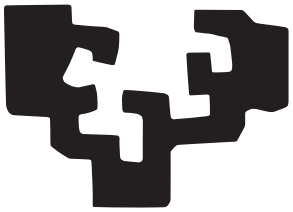
University of the Basque Country (UPV/EHU)

eman ta zabal zazu



Universidad del País Vasco

Euskal Herriko Unibertsitatea

Robotics and Autonomous Systems Group

Faculty of Informatics

Doctoral thesis

Advances in flexible manipulation through the application of AI-based techniques

Ander Iriondo Azpiri

1. Supervisor Elena Lazkano

Department of Computer Science and Artificial Intelligence
University of the Basque Country (UPV/EHU)

2. Supervisor Ander Ansuategi

Intelligent and Autonomous Systems
Tekniker - Basque Research and Technology Alliance (BRTA)

January, 2023

(cc) 2023 ANDER IRIONDO AZPIRI (cc by 4.0)

Ander Iriondo Azpiri

Advances in flexible manipulation through the application of AI-based techniques

Doctoral thesis, January, 2023

Supervisors: Elena Lazkano and Ander Ansuategi

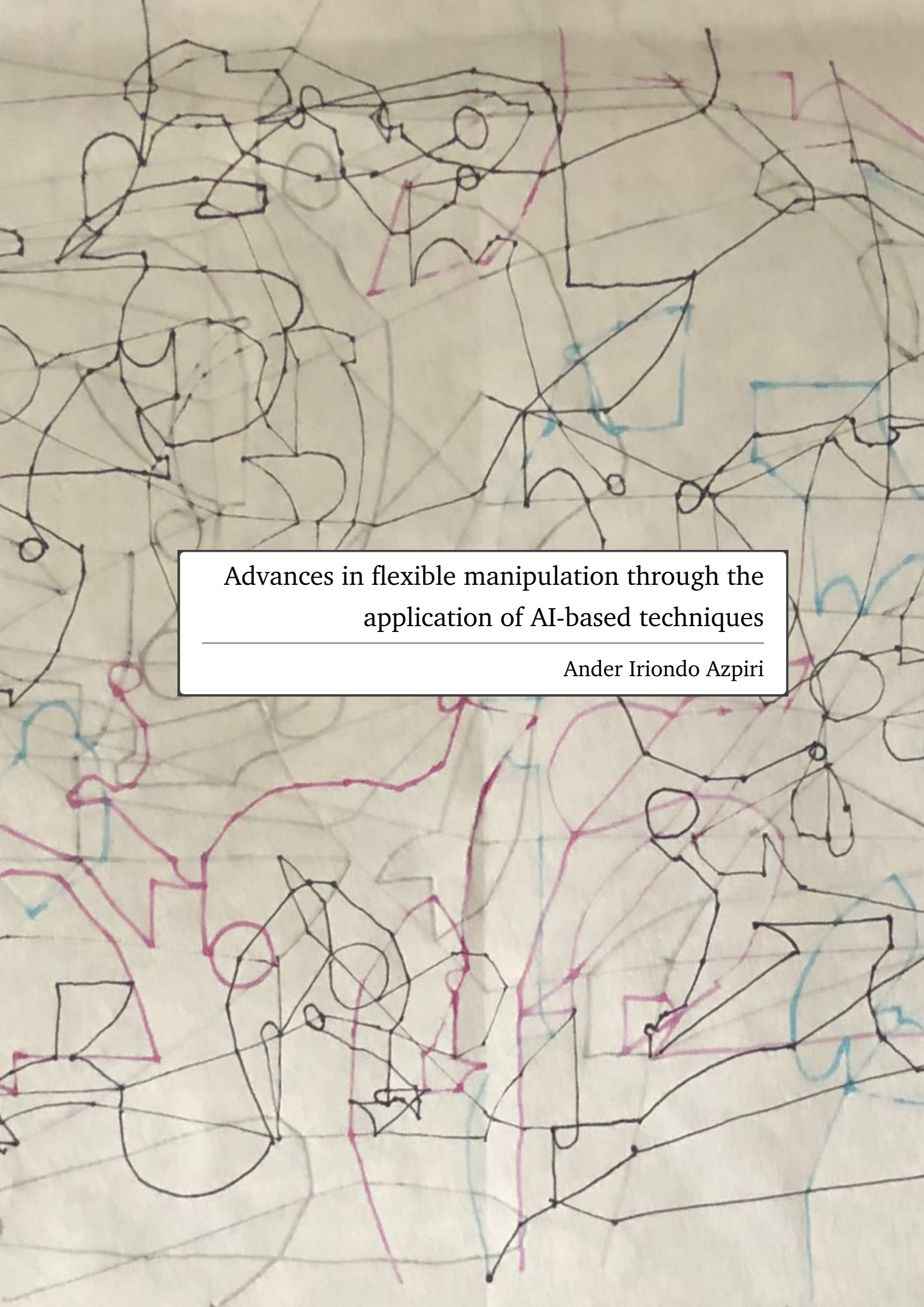
University of the Basque Country (UPV/EHU)

Robotics and Autonomous Systems Group

Faculty of Informatics

Manuel Lardizabal ibilbidea, 1

20018 Donostia



Advances in flexible manipulation through the application of AI-based techniques

Ander Iriondo Azpiri

Abstract

The manufacturing industry was pioneer to incorporate robots to automate costly processes, with the aim of increasing production and reducing costs. Today, robots have a significant presence in almost every industry, generally carrying out simple and repetitive tasks. We are in the transition between Industry 4.0 and 5.0, where in addition to productivity, flexibility is also sought to adjust processes to specific customer needs.

Pick and place are two basic operations in practically any robotic application. Conventional robotic pick and place solutions currently in use in industry are characterised by their efficiency in performing simple, repetitive tasks. However, these are very rigid systems, work in completely controlled environments, and are very costly to reprogramme for other tasks. There are currently tasks in various industrial environments (e.g. order generation in a logistics environment) which require flexible handling of objects and which have not yet been automated due to their nature. The main bottlenecks which hinder their automation are the variety of objects to be handled, the lack of robot dexterity and the uncertainty introduced by uncontrolled dynamic environments.

In contrast, advanced robotics seeks system flexibility as well as productivity. This type of robotics is applied to problems that require advanced decision-making or unstructured environments, using collaborative robots. Artificial intelligence (AI) is playing an increasingly important role in robotics, as it provides robots with the necessary intelligence to solve complex tasks. In addition, AI allows complex behaviours to be learned using real-world experience, thus considerably reducing the cost of programming.

In view of the limitations of current robotic object manipulation systems, the main goal of this work is to increase the flexibility of manipulation systems using AI-based algorithms, providing robotic systems with the necessary capabilities to adjust to dynamic environments without the need for reprogramming. We focus on overcoming the limits in three specific lines of research related to pick and place:

1. The first line of research focuses on studying the feasibility of learning a positioning policy for the base of a mobile manipulator, in order to enable the

arm to pick up an object. Specifically, we propose to learn such behaviour using the experience acquired through interaction with the environment and thus avoiding programming it.

2. The second line of research is focused on developing a grasping point detection system for arbitrary objects. The idea is that such a module will be able to estimate grasping points on a wide variety of objects in bin-picking scenes, thus avoiding the need to configure the system for each reference. Specifically, we propose to directly perform learning on n-dimensional point clouds labelled by an expert, and show that a higher accuracy is obtained than a 2D image-based reference system.

3. The last line of research is based on the development of a dynamic algorithm for the generation of object packing mosaics. Precisely, the proposed algorithm is able to compute mosaics with arbitrary objects in an online way. The validation of the system carried out on a real robotic prototype demonstrates the flexibility of the packaging system to generate mosaics with a wide variety of objects.

Laburpena

Fabrikazio-industria izan zen prozesu astunak automatizatzeko robotak sartu zituen lehenetarikoa, ekoizpena handitzeko eta kostuak murrizteko helburuarekin. Gaur egun, robotek presentzia handia dute praktikoki edozein industriatan, eta, oro har, zeregin sinpleak eta errepikakorrak egiten dituzte. Industria 4.0 eta 5.0 arteko trantsizioan aurkitzen garen momentu hontan, produktibitateaz gain, malgutasuna ere bilatzen da, prozesuak bezeroaren behar espezifikoetara egokitzeko.

Objektuak hartu eta uztea oinarrizko bi eragiketa dira ia edozein aplikazio robotikotan. Gaur egun, "pick and place" aplikazioetarako erabiltzen diren robot industrialek zeregin sinpleak eta errepikakorrak egiteko duten eraginkortasuna dute ezaugarri. Hala ere, sistema horiek oso zurrunak dira, erabat kontrolatutako inguruneetan lan egiten dute, eta oso kostu handia dakarte beste zeregin batzuk egiteko birprogramatzeak. Gaur egun, industria-ingurune desberdinetako zereginak daude (adibidez, logistika-ingurune batean eskaerak prestatzea), zeinak objektuak malgutasunez manipulatzea eskatzen duten, eta oraindik ezin izan dira automatizatu beren izaera dela eta. Automatizazioa zailtzen duten botila-lepo nagusiak manipulatu beharreko objektuen aniztasuna, roboten trebetasun falta eta kontrolatu gabeko ingurune dinamikoen ziurgabetasuna dira.

Aitzitik, robotika aurreratuak sistemen malgutasuna bilatzen du, produktibitateaz gain. Robotika mota hau erabaki aurreratuak hartzea eskatzen duten arazoetan edo egituratu gabeko inguruneetan aplikatzen da, elkarlaneko robotak erabiliz. Adimen artifizialak (AA) gero eta paper garrantzitsuagoa betetzen du robotikaren barruan, robotei zeregin konplexuak betetzeko beharrezko gaitasuna ematen baitie. Gainera, AAk benetako esperientzia erabiliz portaera konplexuak ikasteko aukera ematen du, programazioaren kostua nabarmen murriztuz.

Objektuak manipulatzeko egungo sistema robotikoen mugak ikusita, lan honen helburu nagusia manipulazio-sistemen malgutasuna handitzea da AAn oinarritutako algoritmoak erabiliz, birprogramatu beharrik gabe ingurune dinamikoetara egokitzeko beharrezko gaitasunak emanez. "Pick and place" delakoarekin lotutako hiru ikerketa-lerro espezifikotan mugak gainditzean zentratzen gara lan honetan:

1. Lehen ikerketa-ildoa manipulatzaile mugikor baten oinarrirako posiziona-mendu politika bat ikastearen bideragarritasuna aztertzean datza, besoa ob-

jektu bat hartzeko gaitzeko helburuarekin. Zehazki, portaera hori ikastea proposatzen dugu, ingurunearekiko elkarreraginaren bidez lortutako esperientzia erabiliz eta, horrela, gaitasun horren programazioa saihestuz.

2. Bigarren ikerketa-lerroa objektu arbitrarioentzako heldulekuak detektatzeko sistema orokor bat garatzera bideratuta dago. Horrela, bin-picking eszenetan objetu anitzen heldulekuak zenbatestea da helburua, erreferentzia bakunentzat sistema konfiguratu beharra saihestuz. Zehazki, aditu batek etiketatutako n-dimentsioko puntu-hodeien gainean zuzenean ikasketa egitea proposatzen dugu, eta 2D irudietan oinarritutako sistema bat erabiliz baino zehaztasun handiagoa lortzen dela frogatzen dugu.

3. Azken ikerketa-ildoa algoritmo dinamiko baten garapenean oinarritzen da, objektuak paketatzeko mosaikoak sortzeko. Hain zuzen ere, mosaikoak sortzeko proposatzen den algoritmoa gai da mosaikoak objektu arbitrarioekin eta online kalkulatzeko. Prototipo robotiko erreal batean egindako sistemaren balidazioak agerian uzten du paketatze-sistemaren malgutasuna, mota askotariko objektuekin mosaikoak sortzeko.

Resumen

La industria de la fabricación fue una de las primeras en incorporar robots para automatizar procesos costosos, con el objetivo de aumentar su producción y reducir costes. Actualmente, los robots tienen una presencia importante prácticamente en cualquier industria, generalmente llevando a cabo tareas simples y repetitivas. Nos encontramos en la transición entre la Industria 4.0 y 5.0, donde además de la productividad también se busca la flexibilidad para ajustar los procesos a las necesidades específicas del cliente.

Coger y dejar objetos son dos operativas básicas prácticamente en cualquier aplicación robótica. Las soluciones convencionales de "pick and place" con robots industriales actualmente existentes en la industria se caracterizan por su eficiencia a la hora de realizar tareas simples y repetitivas. Sin embargo, estos son sistemas muy rígidos, trabajan en entornos totalmente controlados, y supone un coste muy alto reprogramarlos para realizar otras tareas. Actualmente existen tareas en distintos entornos industriales (p.ej. preparación de pedidos en un entorno de logística) los cuales requieren de una manipulación flexible de los objetos y que todavía no se han podido automatizar debido a su naturaleza. Los principales cuellos de botella los cuales dificultan su automatización son la variedad en los objetos a manipular, falta de destreza de los robots y la incertidumbre que introducen los entornos dinámicos no controlados.

Por el contrario, la robótica avanzada además de la productividad busca la flexibilidad de los sistemas. Este tipo de robótica se aplica en problemas que requieren una toma de decisiones avanzada o en entornos no tan estructurados, utilizando robots colaborativos. La inteligencia artificial (IA) cada vez juega un papel más importante dentro de la robótica, ya que dota a los robots con la capacidad necesaria para resolver tareas complejas. Además, la IA permite aprender comportamientos complejos utilizando experiencia real, reduciendo así considerablemente el coste de la programación.

Dadas las limitaciones de los sistemas robóticos actuales para manipular objetos, el objetivo principal es aumentar la flexibilidad de estos sistemas de manipulación utilizando algoritmos basados en IA, dotando a los sistemas robóticos con las capacidades necesarias para ajustarse a entornos dinámicos sin necesidad de ser reprogramados. Concretamente, nos centramos en superar los límites en tres líneas de investigación concretas relacionadas con el "pick and place":

1. La primera línea de investigación se enfoca en estudiar la viabilidad de aprender una política de posicionamiento para la base de un manipulador móvil, con el objetivo de habilitar al brazo para coger un objeto. Concretamente, proponemos aprender dicho comportamiento utilizando la experiencia adquirida mediante la interacción con el entorno y así evitar la programación del mismo.

2. La segunda línea de investigación busca desarrollar un sistema de detección de puntos de agarre para objetos arbitrarios. La idea es que dicho módulo sea capaz de estimar puntos de agarre sobre una gran variedad de objetos en escenas de bin-picking, evitando así la necesidad de configurar el sistema para cada referencia. Concretamente, proponemos realizar directamente el aprendizaje sobre nubes de puntos n-dimensionales etiquetados por un experto, y demostramos que se obtiene una precisión mayor que un sistema de referencia basado en imágenes 2D.

3. La última línea de investigación se basa en el desarrollo de un algoritmo dinámico para la generación de mosaicos de empaquetado de objetos. Precisamente, el algoritmo de generación de mosaicos que se propone es capaz de calcular mosaicos con objetos arbitrarios y de manera online. La validación del sistema llevada a cabo en un prototipo robótico real demuestra la flexibilidad del sistema de empaquetado para generar mosaicos con una gran variedad de objetos.

Acknowledgement

I would like to start this report by thanking the people who have helped me during the course of this work over the past four years.

First, I would like to thank my supervisor Elena Lazkano for her help in difficult times and for her involvement in this thesis. I would also like to thank Tekniker for giving me the opportunity to carry out this thesis, and especially to the co-director of this work, Ander Ansuategi and all other colleagues in the unit. You have provided me with the technical help I needed to get out of those moments of blockage that you feel the research is not progressing.

Outside of work, I would like to thank my parents Carmen and Jesus, my brother Eneko, and sisters Lorea and Naroa, and in general the whole family. I would especially like to thank my mother for motivating me to embark on this project when it was not so clear to me. Special mention to my uncle Vicente Iriondo, for showing so much interest in the progress of this work, and for the drawing of the cover of this report. I would also like to thank my partner Marina for supporting me when I could not see the end of the road, especially in this last phase of the work. Finally, I would also like to thank my friends, who, although they had little idea of what I was doing, have been vital in the moments of disconnection.

Many thanks to all of you!

Eskertzak

Txosten hau hasteko, eskerrak eman nahi dizkiet azken lau urteetan lan honetan lagundu didatenei.

Lehenik eta behin, eskerrak eman nahi dizkiot nire zuzendaria izan den Elena Lazkanori, une zailetan laguntzeagatik eta tesi honetan izan duen inplikazioagatik. Teknikerri ere eskerrak eman nahi nizkioke tesi hau egiteko aukera emateagatik, bereziki zuzendarikide izan den Ander Ansuategiri eta baita unitateko beste lankideei. Ikerketak aurrera egiten ez duela sentitzen duzun blokeo-une horietatik ateratzeko behar nuen laguntza teknikoa eman didazue.

Lanetik kanpo, eskerrak eman nahi dizkiet nire gurasoei, Carmen eta Jesus, Eneko anaiari, eta Lorea eta Naroa arrebei, eta, oro har, familia osoari. Batez ere amari, hain argi ez neukanean proiektu honetan murgiltzera motibatu ninduelako. Aipamen berezia nire osaba Vicente Iriondori, lan honen aurrerabidean hainbesteko interesa erakutsi izanagatik eta baita txosten honen azaleko marrazkiagatik. Era berean, eskerrak eman nahi dizkiot nire biokotekidea den Marinari, bidearen amaiera ikusten ez nuenean emandako babesagatik, bereziki tesiaren azken fase honetan. Amaitzeko, eskerrak eman nahi dizkiet nire lagunei, egiten ari nintzenaren ideia handirik ez izan arren, ezinbestekoak izan direlako deskonexio-uneetan.

Eskerrik asko guztioi!

Contents

I The rationale **1**

1 Introduction **3**
1.1 Advanced robotics 4
1.2 PICKPLACE EU project 6
1.3 Research environment 8
1.3.1 Tekniker 8
1.3.2 UPV/EHU 10

2 Motivation **11**
2.1 Pick and place operations 12
2.1.1 The pick 13
2.1.2 The place 15
2.2 Artificial intelligence in robotics 16
2.3 Goal 18
2.4 Research outcomes 18
2.5 Structure of the report 20

3 Studying the feasibility of applying deep reinforcement learning to model a positioning policy for a pick and place operation with an AMMR **23**
3.1 State of the art 24
3.2 Goal 27
3.3 Developed approach 28
3.3.1 Theoretical background 29
3.3.1.1 Introduction to RL 29
3.3.1.2 RL/DRL algorithms 31
3.3.1.3 DRL and robotics 33
3.3.2 Preliminary study in simulation 34
3.3.2.1 Robotic environment 34
3.3.2.2 DRL-based control architecture 36
3.3.2.3 DDPG and PPO as base controller 37

3.3.2.4 Training procedure . . . 41
3.3.2.5 Evaluation metrics . . . 41
3.3.2.6 Results . . . 42
3.3.3 Simulation-to-reality transfer of the learned behaviour . . . 44
3.3.3.1 Robotic environment . . . 44
3.3.3.2 Updated control architecture . . . 46
3.3.3.3 TD3 as base controller . . . 47
3.3.3.4 Training procedure . . . 51
3.3.3.5 Experimentation in simulation . . . 52
3.3.3.6 Validation in the real robotic system . . . 57
3.4 Innovation . . . 61
3.5 Conclusions and future work . . . 63
3.6 Publications . . . 65

4 Affordance-based grasping point detection in point clouds with graph convolutional networks 67
4.1 State of the art . . . 68
4.2 Goal . . . 73
4.3 Developed approach . . . 74
4.3.1 Theoretical background . . . 74
4.3.1.1 Point cloud segmentation with deep learning . . . 74
4.3.1.2 Graph convolutional networks . . . 76
4.3.2 Robotic environment . . . 78
4.3.3 Affordance grasping dataset for suction and gripper . . . 79
4.3.3.1 Suction . . . 80
4.3.3.2 Gripper . . . 81
4.3.4 Predicting object affordances with GCNs . . . 84
4.3.4.1 Data preprocessing for suction . . . 85
4.3.4.2 Data preprocessing for the gripper . . . 86
4.3.5 Evaluation metrics . . . 87
4.3.6 Experimentation . . . 87
4.3.6.1 Known objects . . . 87
4.3.6.2 Novel objects . . . 90
4.4 Innovation . . . 93
4.5 Conclusions and future work . . . 94
4.6 Publications . . . 96

5 Dynamic mosaic planning for a robotic packing system 97
5.1 State of the art . . . 98

5.2 Goal . . . 100
5.3 Developed approach . . . 101
5.3.1 Theoretical background . . . 102
5.3.1.1 Offline IK-PAL . . . 102
5.3.1.2 Empty maximal spaces . . . 105
5.3.2 Workspace . . . 106
5.3.3 Use cases . . . 107
5.3.4 Dynamic bin-packing system . . . 108
5.3.4.1 Picked part monitoring . . . 109
5.3.4.2 Destination bin monitoring . . . 110
5.3.4.3 Dynamic IK-PAL . . . 113
5.3.4.4 Flow of the packaging application . . . 115
5.3.5 Experimentation . . . 116
5.3.5.1 Picked part monitoring . . . 116
5.3.5.2 Destination bin monitoring . . . 117
5.3.6 System validation in the real robotic environment . . . 122
5.4 Innovation . . . 125
5.5 Conclusions and future work . . . 126
5.6 Publications . . . 128

6 Conclusions **129**
6.1 Future works . . . 131

Bibliography **135**

II The research **147**

7 Pick and Place Operations in Logistics Using a Mobile Manipulator Controlled with Deep Reinforcement Learning **149**

8 Affordance-Based Grasping Point Detection Using Graph Convolutional Networks for Industrial Bin-Picking Applications **171**

9 Dynamic mosaic planning for a robotic bin-packing system based on picked part and target box monitoring **199**

10 Learning positioning policies for mobile manipulation operations with deep reinforcement learning **223**

List of Figures

1.1 Advanced robotics application niche, according to [3]. 4
1.2 Forecast for collaborative robot market size, according to Statista. . . . 6
1.3 Order preparation at TOFAŞ and UHS's warehouses. 7

3.1 Reinforcement learning framework. 29
3.2 Taxonomy of RL/DRL algorithms. 32
3.3 Kuka Miiwa AMMR. 35
3.4 Simulated environments in Gazebo. 36
3.5 Implemented control architecture. 36
3.6 Test 1 results. 42
3.7 Test 2 results. 43
3.8 AMMR created at Tekniker. 45
3.9 Simulated environment in Unity. 46
3.10 Proposed control architecture to control the AMMR. 47
3.11 Network architecture of TD3. 48
3.12 Training average rewards and success rates with PPO2, DDPG and TD3 algorithms with the *baseline* and the *proposed* setups. 56
3.13 Mean across-setup reliability rankings of TD3, DDPG and PPO2 algorithms considering the average reward training curves. Rank 1 always means the best reliability. 57
3.14 Map of the real scenario. 58
3.15 Real validation scenario. 60

4.1 Taxonomy of DL-based GPD methods. 69
4.2 Taxonomy of point cloud semantic segmentation methods. 75
4.3 Dilated convolution on a structured graph arranged in a 2D grid and using a 3×3 kernel (top), and dynamic graph convolution in an irregular graph, e.g. 3D point cloud, (bottom). From left to right, dilation rates $d = 1$, $d = 2$ and $d = 4$ are used [107]. 77

4.4 Residual and dense GCNs, where d refers to the dilation rate, k to the number of neighbours, and f to the number of filters. Note that in the residual GCN a vertex-wise addition is used, while in the dense GCN vertex-wise concatenation is used. 78
4.5 Multi-functional gripper attached to the UR robot. 78
4.6 Annotation transformation for suction. 80
4.7 Data preprocessing pipeline. 81
4.8 Gripper annotations in a RGB-D orthographic heightmap. 82
4.9 Annotation transformation for the gripper. 84
4.10 Used model architecture. 84
4.11 Obtained precision scores per confidence percentiles for the suction models with known objects. 88
4.12 Suction result example. Top-left: Point cloud of the scene. Top-right: Ground-truth annotation. Bottom-left: Top-1% predictions. Bottom-right: Top-5% predictions. 89
4.13 Obtained precision scores per confidence percentiles for the gripper models in with known objects. 89
4.14 Gripper result example for $n = 4$ $(90°)$. Top-left: Point cloud of the scene. Top-right: Ground-truth annotation. Bottom-left: Top-1% predictions. Bottom-right: Top-1 grasping point. The vertical orientation for the gripper is determined by the y (green) axis. 90
4.15 Obtained precision scores per confidence percentiles for the suction models wit novel objects. 91
4.16 Suction result example with totally new objects. Top-left: Point cloud of the scene. Top-right: Ground-truth annotation. Bottom-left: Top-1% predictions. Bottom-right: Top-5% predictions. 91
4.17 Obtained precision scores per confidence percentiles for the gripper models with novel objects. 92
4.18 Gripper result example for $n = 2$ $(45°)$. Top-left: Point cloud of the scene. Top-right: Ground-truth annotation. Bottom-left: Top-1% predictions. Bottom-right: Top-1 grasping point. The vertical orientation for the gripper is determined by the y (green) axis. 92

5.1 Products organised in layers. 103
5.2 Generated layers depending on their size. In black a full-size layer. In red and blue, length and width half-size layers respectively. In green a quarter-size layer. 104
5.3 Theoretical representation of a mosaic. 105
5.4 Empty maximal spaces after packing an item inside the container [133].106

5.5 Main components of the TOFAŞ pilot. (1) Inbound area, (2) part monitoring area, (3) outbound area. 107
5.6 Bounding box estimation of the picked part. The monitoring volume and the bounding box are represented as red and green cubes respectively. The blue sphere indicates the origin of the bounding box. 110
5.7 Side view of the generated empty cubes in 2D in an example scene. Green items represent already packed objects and s_1, s_2, s_3, s_4 are the generated EMSs for that scene. 111
5.8 Occupancy grid obtained after each processing phase. 113
5.9 Projections of the cubic solutions that represent the empty space inside the bin. 113
5.10 The scores of the 96 parameter combinations considering the average usable volume ratio, average execution time and the average number of generated EMSs. In red the non-dominated parameter combinations that belong to the Pareto front. The green star represents the scores of the selected parameter combination. 120
5.11 Execution times, usable volume ratios and generated EMSs with respect to scenes with low, medium and high complexity. 121

List of Tables

1.1 Industrial and technological objectives of PICKPLACE. 8

2.1 Research line and outcome mapping. 19

3.1 Variables that define the state of Environment 1. 39
3.2 Variables that define the state of Environment 2. 39
3.3 Variables that define the environment state. 49
3.4 Weights and constant values used for the reward function. 51
3.5 Measurements obtained in the real environment. 60

4.1 Features that define a point for the suction. 86
4.2 Features that define a point for the gripper. 86

5.1 Features of the pallet in the INP file. 102
5.2 Features of the objects in the INP file. 103
5.3 Features of the objects in the OUT file. 105
5.4 Tuning parameters for the picked part monitoring algorithm. 117
5.5 Errors in the bounding box estimation during the picked part monitorisation. 118
5.6 Tuning parameters and estimated mean scores for the destination bin monitoring heuristic. 119

Part I

The rationale

1 Introduction

Historically, humans have always been eager to create automatons to speed up repetitive and forced work. Nowadays there is a great variety of robot types that have been created for different purposes. In particular, the operation of picking up and putting down objects is a basic operation in almost any type of robotic application, but due to the difficulty of automating it, manipulation has traditionally been one of the main lines of research within robotics. The introduction of robots in industry came in the 20th century along with the third industrial revolution. This revolution was mainly due to the invention of the transistor, which enabled the development of programmable logic controllers (PLC), which in turn facilitated the automation of processes in industry. In particular, the manufacturing industry was one of the first to introduce robotic arms on assembly lines, mainly to automate costly and repetitive jobs such as bin-picking, palletising or assembly, in order to increase production and reduce costs.

We are currently living in the transition from the fourth to the fifth industrial revolution. The fourth industrial revolution, also known as Industry 4.0, is mainly characterised by the increasing automation of processes, the use of machines and smart factories, computing and data analysis in the cloud and the integration of artificial intelligence into processes [1]. Thus, the aim is to increase the flexibility of manufacturers so that they can better adjust to customer requirements, as well as to improve the efficiency of companies through the massive collection and processing of data using artificial intelligence. Industry 5.0 aims beyond efficiency and productivity and reinforces the role and the contribution of industry to society. In fact, it places workers' welfare at the heart of the production process and uses new technologies to deliver prosperity beyond employment and growth. In addition, it addresses the training needs of employees to increase the competitiveness of industry [2].

1.1 Advanced robotics

The application area of both conventional and advanced robotics is determined by the following characteristics of production processes (see Figure 1.1): The number of references to be managed, the volume to be produced for each of the references, the need for decision-making during the execution of the task or the level of structuring of the environment [3].

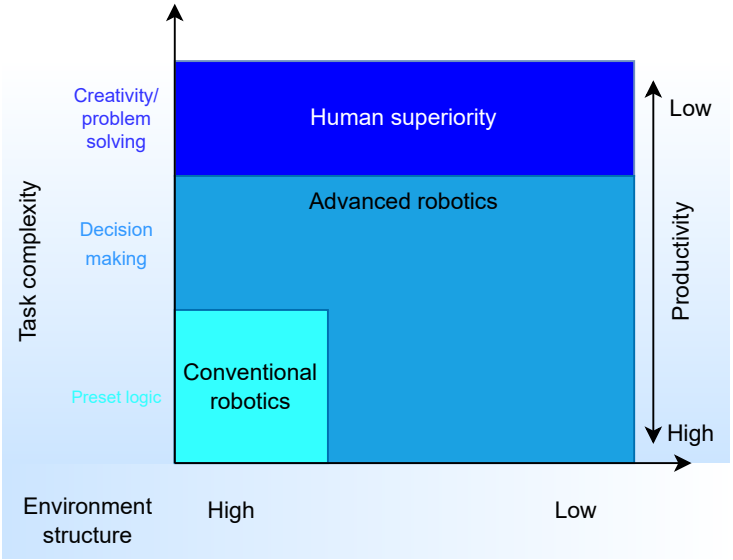


Fig. 1.1.: Advanced robotics application niche, according to [3].

On the one hand, conventional robotics seeks to automate simple and repetitive tasks in highly structured environments. Currently, there are many processes in different sectors of industry that require objects to be handled and which are already automated by robots (e.g. assembly lines in sectors such as automotive, aviation, etc.). The common denominator of these processes is that the objects to be handled are always the same, the environment is highly structured and with very little uncertainty, and generally the tasks performed with these objects are very mechanical and repetitive. Conventional robotics applications, usually implemented as rules based on expert-knowledge, have proven to be very efficient in such tasks in which hardly any process variation occur [4]. From the point of view of productivity, industrial robots have been a step forward especially in labour-intensive tasks that hardly require complex decision making, as they can work 24/7 in a very efficient way at a much lower cost. However, the high efficiency offered by these types of robots directly penalises flexibility, as they are generally very rigid systems and are designed to carry out a single task. In fact, those need to be configured by a robotics

expert, which usually involves considerable material and time cost. Moreover, they are designed to follow a pre-programmed logic in fully controlled environments with strict safety measures, and therefore, without human presence. As stated by Kober and Peters in [5], hard-coded systems are restricted to the situations that the programmer anticipated, and are impractical for scenarios with high uncertainty.

On the other hand, advanced robotics prioritises flexibility of solutions in addition to productivity, and is applied to problems with advanced decision-making and uncertain environments. For this purpose, collaborative robots are generally used, which are much more flexible systems compared to industrial robots. Collaborative robots, unlike the industrial ones, are designed to work under the guidance of an operator in a coordinated and safe manner. This makes them very versatile in the sense that they can be programmed to perform different tasks without the need of expertise in the field of robotics. In addition, they are designed in such a way that they allow a safe collaboration with humans, being able to perceive what is happening around them and act accordingly. These type of robots have gained enormous popularity in recent years and have given rise to numerous studies on the impact on the economy and the employment [6].

As it can be seen in Figure 1.2, according to the statistics portal Statista [1], the size of the collaborative robot market has been growing steadily in recent years and although the Covid19 pandemic caused a slow-down in global economy in 2020, it is expected to continue in the same way in the coming years. Collaborative robots allow the creation of flexible robotic solutions, capable of adjusting quickly and efficiently to changes that may occur in the production chain at a much lower cost. In general, these are much more open systems which can be easily integrated with other systems, thus allowing more versatile and intelligent applications to be created. This type of robot is the most widely used in advanced robotics applications because of the flexibility and safety they offer, and they are in fact a good solution for problems in semi-structured environments that require complex decision making [7].

[1] https://www.statista.com/statistics/748234/global-market-size-collaborative-robots/

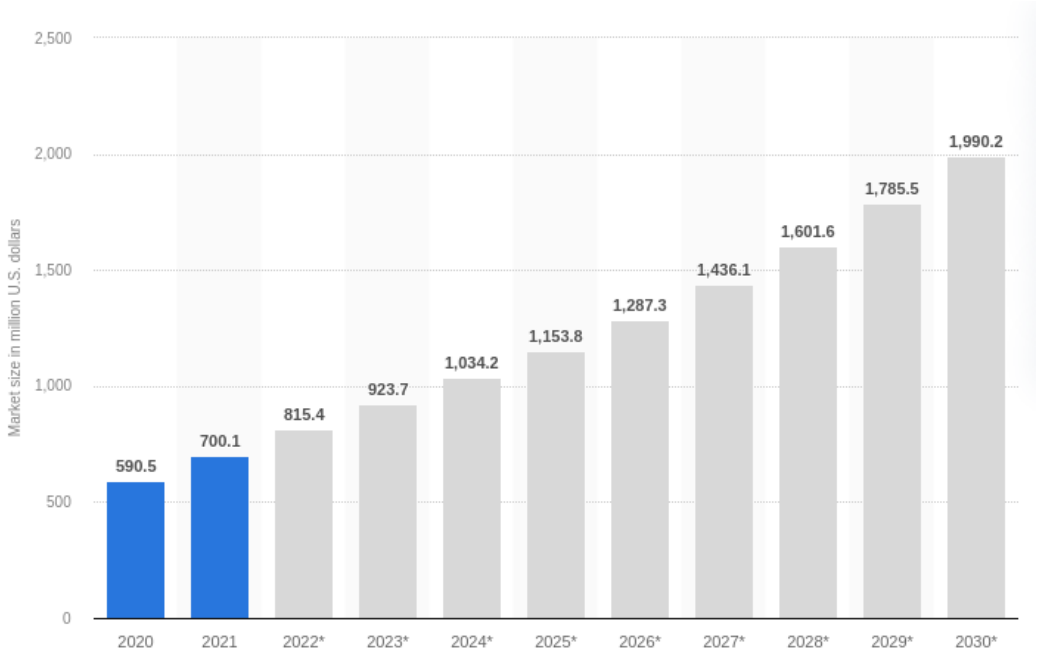


Fig. 1.2.: Forecast for collaborative robot market size, according to Statista.

1.2 PICKPLACE EU project

This research work has been carried out largely in the context of the PICKPLACE [2] project funded by the European Commission. The project's mission was as follows:

> "PICKPLACE focuses on flexible, safe and dependable robotic part-handling in industrial environments. The project proposes a combination of human and robot capabilities in order to achieve this efficient hybrid pick-and-place / pick-and-package solution. It includes dynamic package configuration, flexible grasping strategies using an innovative multi-functional gripper, robust environment perception and mechanisms and strategies for human-robot collaboration."

The consortium was composed of the following partners: CNR-STIIMA, FRAUNHOFER IFF, MONDRAGON ASSEMBLY and TEKNIKER, besides to the ULMA HANDLING SYSTEMS (UHS) and TÜRK OTOMOBIL FABRIKASI A.Ş. (TOFAŞ) end users, the former in the intralogistics sector and the latter in the automotive sector. In the distribution centres installed by UHS, the order pick-and-packaging (the process of collecting items to create a package for shipment) is the last and most challenging

[2] https://pick-place.eu/

task in the warehouse automation. Depending on the type of installation, the warehouse operator takes multiple types of goods at the end of the line and places them in target boxes. In the automotive sector, TOFAŞ deals with more than 63.000 different spare parts that are sent to different dealers in manually prepared packages. This repetitive and monotonous packing process is currently hold in 90% by humans, due to the difficulty to replace the human dexterity to handle parts with high variability in size, shape, weight and material.

The feasibility of fully automated systems is currently limited by many factors such as the huge variability of items to cope with, lack of system robustness and industrial throughput requirements. Human-robot collaboration seeks to combine the robot's efficiency and the human dexterity to optimise the packing process. Figure 1.3 depicts the order preparation at UHS's and TOFAŞ warehouses.

Fig. 1.3.: Order preparation at TOFAŞ and UHS's warehouses.

The technological objectives of PICKPLACE were derived from the analysis of the requirements of the pick-and-package scenarios in warehouses and distribution centres, and give an answer to the technology gap that represents a market barrier, i.e., the lack of flexible solutions that can handle objects of variable size, shape and weight as well as surface properties and stiffness.

In this context, the industrial objectives (IO) and technical objectives (TO) that were identified are shown in Table 1.1.

IO1	Flexibility, dependability and error reduction in pick-and-package performance.
IO2	Human-robot task allocation and layout design to improve working conditions.
TO1	Development of a new generation multi-functional gripper.
TO2	Reactive grasp planning based on cognitive capabilities.
TO3	Robust and efficient bin-picking.
TO4	Human and robot aware dynamic package planning.
TO5	Dynamic robot planning based on cognitive capabilities.
TO6	Reliable environment perception systems and strategies for safe collaboration scenarios.

Tab. 1.1.: Industrial and technological objectives of PICKPLACE.

1.3 Research environment

1.3.1 Tekniker

Tekniker is a research and development centre located in Eibar and currently is member of *Basque Research and Technology Alliance* (BRTA)[3]. The main fields of expertise of the centre include advanced manufacturing, surface engineering, information and communication technologies and product engineering. Tekniker aims to generate knowledge and transfer it to the industry in the form of technological solutions. In fact, as a research centre it serves a wide variety of sectors such as machine tools and manufacturing, renewable energies, aeronautics and space, science industry, bio-medicine, automotive, infrastructures and e-health.

Autonomous and Intelligent Systems

The Autonomous and Intelligent System (SAI) unit has an extensive experience in developing smart systems with a high level of autonomy. The unit focuses on three main lines of work:

- **Robotics:** The unit leads the Tekniker's Robotics specialisation line, defining and coordinating the activities of other research groups in this field. The research in robotics is focused on endowing robots with intelligence to improve navigation and trajectory generation as well as providing the capability to manipulate objects.

[3] https://www.brta.eus/en/home

- **Human-machine interaction:** Mainly focused on robotics applications, it seeks to combine semantic technologies and the understanding of the human actions/intentions to improve human-robot collaboration.

- **Computer vision:** Although its main application is the quality control, it also plays a key role in the other two research lines.

As a member of SAI, during the development of this research work, the author has participated in the following projects related to the application of AI to increase the flexibility of robotic manipulation systems:

- *KOBOT: Flexible and collaborative industrial robotics solutions for logistics operations.* This project was an HAZITEK (industrial R&D project funded by the Basque government) leaded by ULMA Handling Systems [4] (UHS), company leader in the intralogistic sector. In this context, Tekniker developed an AI-based grasping point detection system integrated in a functional prototype for flexible object manipulation.

- *PICKPLACE*: Flexible, safe and dependable robotic part-handling in industrial environments. The project proposed a combination of human and robot capabilities in order to achieve an efficient hybrid pick-and-place / pick-and-package solution. Tekniker played a system integrator role, and in addition, it contributed in the development of the flexible picking and packaging system.

- *MALGUROB*: Development of flexible robotics technologies for the automation of manufacturing processes in the Basque industry.

- *ELKARBOT*: Development of new functionalities in flexible robotics to promote the implementation of robots in Basque industry.

- *TREBEZIA*: Smart and agile BRTA ecosystem to accelerate the implementation of the new generation of robotics in the Basque Country's factories of the future. This basic research ELKARTEK project funded by the Basque Government continued the path of ELKARBOT. The project was leaded by Tecnalia and the consortium was composed of 8 members of BRTA.

- *5R*: Initiative financed by CDTI in the call "Cervera Technology Centres" of the "State Programme for Knowledge Generation and Scientific and Technological

[4] https://www.ulmahandling.com/

Strengthening of the R&D&I System", to establish a collaborative network, equipped with the necessary technology, tools and infrastructures, to act as a driving force for the development and introduction of new robotic technologies in the industrial network of manufacturing.

1.3.2 UPV/EHU

The University of the Basque Country (UPV/EHU) is the public university of the Basque Autonomous Community (Spain). It is an institution with more than 40 years of history with state-of-the-art facilities, which underpins much of the region's scientific and technological progress and success. UPV/EHU has endowed the author of this research with the knowledge in computer sciences and artificial intelligence needed to carry it out. Starting from the degree in computer sciences, it provided me with the basic knowledge on programming and machine learning. Then, the Masters degree carried out in the same faculty, helped me to acquire the skills needed to develop smart robotic applications powered by artificial intelligence.

RSAIT

RSAIT is the research group one of the supervisors belongs to. This group works at the Informatics faculty located at Donostia/San Sebastián, and their main objective is to use data analysis and machine learning methods to increase the autonomy of robots. The experience of RSAIT in the field has been vital in channelling the lines of this research.

2 Motivation

As mentioned before, conventional robotic pick and place systems have proven to be very robust and efficient in highly controlled industrial environments and in tasks that do not require complex decision making. However, there are certain tasks in different industrial environments (e.g. the preparation or return of an order in a logistics centre or warehouse) that require the flexible handling of objects and which, due to their nature, have so far not been automated and are currently carried out by humans. The factors that generally hinder the automation of these operations are the following:

- Variability of objects to be manipulated: There are operations that require the manipulation of a large number of references with highly variable characteristics such as shape, weight or texture. On the one hand, this is a challenge from the point of view of perception, as the robotic system has to be able to estimate the way in which each type of object is picked up. On the other hand, from a hardware point of view, due to the variability in the physical properties of objects, not all robot grippers (e.g. suction, two-finger gripper, magnet...) allow to grasp all type of objects.

- Dynamic environments: In addition to the type of objects, generally the way in which they are placed in the environment induces uncertainty in the operation. Indeed, objects can lay in the environment in a high diversity of poses, mixed together disorderly (e.g. inside a box), occluding each other and making even more difficult the picking process. Furthermore, the industrial environments in which these pick and place operations occur are already dynamic, which makes conventional robotic systems useless.

- Lack of dexterity: The dexterity we humans have allows us to almost unconsciously pick up any type of object, even if it is totally unfamiliar. We unconsciously associate the object to be picked up with other elements that we already know, and on that basis we deduce how the new object should be grasped. Furthermore, as soon as our hands contact the object, we are able to make reactive decisions based on the sensations we perceive in order to

adjust the force with which we grasp the object. Conventional robotic systems are very rigid, and although they solve tasks with a pre-established logic and in controlled environments very efficiently, they are not appropriate in cases where decision-making and the ability to adapt to new situations play an important role.

Summing up, advanced robotics aims to achieve more flexible robotic solutions that are able to adapt to uncertainties in industrial environments in an autonomous and intelligent way.

2.1 Pick and place operations

Pick and place are basic operations in any robotic application, from industrial environments (e.g. machine tending, assembling or bin-picking) to agricultural, healthcare or service environments.

Over the last few years, particularly in the logistics sector, small and medium-sized enterprises (SMEs) were specialised in small sets of product references, and it was common for them to have a very large number of items in stock in order to reduce costs. However, with the recent growth of e-commerce, customer orders tend to be very small and with very varied references, and this has made holding a large stock dangerous. Indeed, logistic centres cannot specialise in specific products and have to work with millions of different references, which introduces a lot of uncertainty in the handling process. As a result, the competitiveness of small and medium-sized companies in this sector has been considerably reduced in recent years. In fact, the cost of maintaining such hard-coded robotic systems causes interruptions in the supply chain and makes it impossible to compete with multinational companies. The same happens for manufacturing companies, where more and more products are customised to specific customer requirements. In both cases, the need for advanced robotic solutions capable of adapting to different tasks and environments is growing.

Traditionally, logistics and manufacturing companies have mainly used manipulator robots (MR) to manipulate goods. However, recently autonomous guided vehicles (AGV), autonomous mobile robots (AMR) and autonomous mobile manipulation robots (AMMR) have gained importance:

- MR: Articulated arms are the most common type of robot used for pick and place operations. These robots are usually attached to a fixed position in the workshop and, thus, their reachability is limited.

- AGV: These kind of wheeled mobile robots are used mainly for transportation in highly structured environments and usually navigate through predefined lanes. AGVs do not have the ability to manipulate objects.

- AMR: These mobile robots are similar to AGVs, but are equipped with sensors and algorithms that enable autonomous navigation. AMRs are also used for transportation of goods and offer a higher level of autonomy and flexibility than AGVs. Yet, they do not have the ability to manipulate objects.

- AMMR: These are AMRs with one or more MRs attached. Thus, the mobile platform is able to autonomously navigate through the workshop extending the reachability of the articulated arm.

This work focuses particularly on pick and place operations with MRs and AMMRs.

2.1.1 The pick

Picking with MRs usually consists of three main phases which involve (1) perception to monitor the environment and estimate the best grasping points in the objects to be picked, (2) grasp planning to calculate the collision-free trajectories considering the obstacles detected by the perception system, and finally (3) the execution of the grasp itself that aims at physically controlling the robot to successfully grasp the item.

Using a specific end-effector and considering its restrictions, grasping points are points on the object that the robot tool has to contact with. Typically, grasping points are identified using a perception system based on computer vision algorithms using 2D/3D images of the scene. Once the grasping points have been estimated, at the grasp planning phase, the trajectory that the robotic arm must follow to reach the grasping points without colliding with the surrounding elements is planned, as well as the tool-related parameters such as the force with which the robot will grip the object. Finally, in the grasp execution phase and as its name indicates, the planned grasp is executed. In certain advanced applications, in addition to simply executing

the plan, reactive control of the robot is sought to adjust to the dynamics of the environment at run time.

When the pick operation is performed by a MR, both the arm itself and the cameras are usually fixed in a specific position in the environment, and calibration algorithms are required to estimate the spatial relationship among them. In turn, this relationship is essential to transform the grasping points from the cameras' reference frame to the robot's coordinate system. In some cases, the camera is attached in the manipulator in order to have several points of view of the scene.

Concerning AMMRs, the picking operation is carried out in a similar way following the aforementioned three phases. However, as the MR is attached to a mobile platform, its position and thus its reachability depend on the location of the AMMR's base. It is then mandatory to position the AMMR in such a way that enables the arm to reach the item. To this effect, several aspects need to be considered:

1. The robot's reach is determined by its kinematics and not only by the distance to the object to be picked.

2. The base has to be positioned in such a way that, taking into account the limited reach of the arm, its kinematics allow it to reach the object to be picked up.

3. To estimate the relative pose between the arm and the object to be picked, it is necessary to know the location of the base with respect to a common coordinate system. Current localisation systems are not perfect and therefore introduce a new factor of uncertainty which makes picking operations more difficult.

The complexity of the picking operation can vary depending on the application. The most common object picking applications in logistics centres are those based on grabbing objects from conveyor belts or containers (i.e. bin-picking). In the former, the items to be picked usually come one at a time and are separated from each other, and the main challenge of this type of application is the difficulty of picking moving objects. In the latter, the objects are stacked inside a box (in an orderly or disorderly manner), and this means that there are usually occlusions causing some objects to be partially visible and others to be completely hidden. In addition, the fact that MRs have to pick up objects from inside a box increases the possibility of

collisions between the robot and the box, and also between the end effector and other objects.

2.1.2 The place

Along with the pick, the place is the operation of leaving an object picked by a MR/AMMR in a desired position. Similarly to the pick, a place operation usually requires a perception system, first to monitor both the picked part and the release area, and then estimate the release pose. Then, in the release planning phase, the collision-free trajectory is calculated for the arm to reach the release pose. Finally, in the release execution phase, the planned path is executed and the uncertainties of the environment are dynamically handled.

The most common types of releases are the following:

1. Drop the object at a fixed point.

2. Place the object in a specific pose.

3. Place the object in an ordered configuration (e.g. an ordered mosaic for palletising or bin-packing).

In the easiest case, the object dropping, the object does not have to be left in a specific pose, hence the robot only has to reach a specific point before releasing it. In that case, the monitoring of the environment is only required to perform a collision free movement. In the second case, in which the item needs to be released in a specific pose, a perception system is needed to monitor both, the picked part and the destination area. This monitoring is usually done with 2D/3D cameras, and it is necessary mainly for the following reasons:

- When the objects to be grasped are known and the grasping points are predefined, as soon as the robot selects a grasping point, it can be estimated what the pose of the object in the robot's gripper will be. However, this is not the case when more flexible grasping point detection systems are used, as these points are not predefined and are calculated on the fly.

- After gripping an object, and depending on its physical properties such as material, texture, etc., the movement of the robot may cause it to wobble, and as a consequence, its pose with respect to the gripper not to be the expected one. This occurs after the objects have been collected, and regardless of the method of identification used.

- Uncertainties that are often present in dynamic industrial environments may have an impact on the target pose and therefore the target pose may vary over time (e.g. the robot has to place an object in a specific pose inside the container, but the pose of the container relative to the robot varies).

Then, during the release planning it is checked whether it is possible for the robot to release the item in the desired pose, then to calculate a collision-free trajectory. Finally, to place objects in an ordered configuration requires planning the target pose. This planning step looks for the optimal release pose for each object taking into account criteria such as space optimisation and stability. The result is a mosaic that indicates the dropping position for each object, and also determines the order in which the objects are picked. Traditional mosaic planners are based on assumptions such that the subset of objects to be packed is known and that there is a 3D model of each element, however this is not always guaranteed when working with a large number of references.

The best known applications of the mosaic planners in logistics centres are palletising and bin-packing. On the one hand, the main objective of palletising is to stack a given set of objects in an orderly manner on a pallet, generally preserving stability and space utilisation criteria. On the other hand, the objective of bin-packing is likewise, but in this case the objects are stacked in an orderly manner inside a box. In this case, the walls of the box usually restrict the accessibility of the arm to it.

2.2 Artificial intelligence in robotics

Robots were first automatic machines created by mankind to effectively automate repetitive processes, they lacked autonomy though. However, this has changed in recent years and more and more advanced applications are being developed. In fact, the same robot can perform multiple tasks and learn from its mistakes and its environment. In this context, artificial intelligence (AI) is playing an increasingly

important role in robotics, and aims to provide robots with the ability to intelligently solve complex tasks.

Machine learning (ML) is a branch of AI which is based on learning patterns in high-dimensional data later to make predictions about new data sets. It has traditionally been used to model knowledge in a wide variety of problems and is based on exploiting existing regularities in previously acquired experience [8]. Such algorithms have also been applied in robotics, mainly with the aim of extracting patterns in the high-dimensional data provided by sensors such as cameras, lasers or similar, which collaborative robots are often equipped with.

In contrast to what has been known as industrial robotics, advanced robotics aims to create autonomous robotic systems able to react to known/unknown events in uncertain scenarios. Furthermore, ongoing research focuses on developing functional robots working in partially modelled scenarios, able to learn from their own mistakes. The major limitation of current industrial robotic applications is the high cost of programming and the poor flexibility they offer. To alleviate these limitations, as detailed in [9], ML can be applied to the following tasks:

- Intelligent and reactive control based on sensory information: The aim is to learn complex robotic behaviours without the need for explicit programming, by means of examples or interaction with the environment.

- Modelling of sensory information. The objective is to model the sensory information of robots to look for patterns and facilitate the robot's decision making.

- Error analysis: Make the robot able to automatically detect and find solutions to possible errors.

- Planning: The aim is to improve decision-making in high-level planning of tasks to be performed, search for sub-task relationships, etc.

2.3 Goal

In view of the limitations of current industrial pick and place systems, the main objective of this research is to push the boundaries and contribute to increase the flexibility of perception and control systems required to perform a pick and place operation, by using AI techniques. Specifically, this work addresses three main lines of research, two of them focused on picking and one on placing:

- **G1:** Learning a positioning policy for an AMMR to carry out a picking operation.
- **G2:** Flexible detection of grasping points in random multi-reference bin-picking scenes.
- **G3:** Dynamic mosaic planning for a robotic packing system with highly variable objects.

2.4 Research outcomes

The following list includes contributions as first author:

- **O1:** Ander Iriondo et al. “Pick and place operations in logistics using a mobile manipulator controlled with deep reinforcement learning”. In: *Applied Sciences* 9.2 (2019), p. 348.
- **O2:** Ander Iriondo Azpiri et al. “Affordance-Based Grasping Point Detection Using Graph Convolutional Networks for Industrial Bin-Picking Applications”. In: *Sensors* 21.3 (2021), p. 816.
- **O3:** Ander Iriondo et al. “Dynamic mosaic planning for a robotic bin-packing system based on picked part and target box monitoring”. In: *The International Journal of Advanced Manufacturing Technology* (2023), pp. 1–21

The following articles have been submitted and are being considered for publication:

- **O4:** Ander Iriondo et al. “Learning positioning policies for mobile manipulation operations with deep reinforcement learning”. In: *International Journal of Machine Learning and Cybernetics* (2023)

In Table 2.1 it can be seen the mapping between each research line with the generated outcomes.

Goal	Outcomes
G1	O1, O4
G2	O2
G3	O3

Tab. 2.1.: Research line and outcome mapping.

The following book chapter has also been published as co-author:

- Loreto Susperregi et al. “RSAII: Flexible Robotized Unitary Picking in Collaborative Environments for Order Preparation in Distribution Centers”. In: *Bringing Innovative Robotic Technologies from Research Labs to Industrial End-users*. Springer, 2020, pp. 129–151

Below are listed the conferences I have attended, including the presentations made at each one:

- **European Robotics Forum 2019, Bucharest.**

  1. Intelligent, flexible and safe operations in future factories. [1]

  2. Pick and Place Operations in Logistics Using a Mobile Manipulator Controlled with Deep Reinforcement Learning. [2]

- **European Robotics Forum 2020, Malaga.**

  1. Artificial Intelligence in robotics for intralogistics applications.[3]

- **European Robotics Forum 2021, Online.**

[1] https://roboception.com/wp-content/uploads/2019/04/ERF2019-Intelligent_-flexible_and_safe_operations-Tekniker.pdf
[2] https://zenodo.org/record/3458219#.Yrsl3i8lMvE
[3] https://drive.google.com/file/d/1ISoi_gXWrXVSLgJmwzi3pNN3PTICSTK8/view

1. Affordance-based Grasping Point Detection using Graph Convolutional Networks for Industrial Bin-Picking applications. [4]

2.5 Structure of the report

As previously mentioned, this work is mainly based on three lines of research. In order to facilitate the understanding of the work carried out, the details of each one of them are collected in a separate chapter, keeping the following structure:

- State of the art
- Goal
- Developed approach
- Innovation
- Conclusions and future work
- Publications

Thus, the rest of the report is structured as follows:

Chapter 3: Studying the feasibility of applying deep reinforcement learning to model a positioning policy for a pick and place operation with an AMMR

This chapter describes the study carried out on the correct positioning of the base of an AMMR to perform a picking operation. Specifically, the feasibility of using deep reinforcement learning (DRL) to learn a controller for the base is analysed, taking into account the reachability constraints of the arm and with the aim of facilitating the programming of such a complex skill.

Chapter 4: Affordance-based grasping point detection in point clouds with graph convolutional networks

[4] https://drive.google.com/file/d/1tdyN69yfzVHM9axG4-LLJDIknoETk5Bw/view

In this chapter we present a study on the flexible detection of grasping points on point clouds, focusing on bin-picking operations in industrial environments. Specifically, we propose a methodology based on graph convolutional networks (GCNs) to predict affordance-based grasping points on n-dimensional point clouds, and its pros and cons are analysed in comparison with methods based on 2D images.

Chapter 5: Dynamic mosaic planning for a robotic packing system

This chapter details the study carried out on the flexible bin-packing of objects in industrial environments. Specifically, it presents an online packing system capable of packing a wide variety of unknown object types.

Chapter 6: Conclusions

This chapter summarises the results obtained during the implementation of the project and answers the research questions raised.

3

Studying the feasibility of applying deep reinforcement learning to model a positioning policy for a pick and place operation with an AMMR

Object manipulation (e.g. bin-picking, assembling) and navigation (e.g. transportation of goods, service robotics) are two of the areas of robotics where most research has traditionally been done. Most existing applications focus on one or the other but there is a shortage of applications that combine the two. Although there are currently several hardware platforms that combine robotic arms with mobile bases (AMMRs), the difficulty and uncertainty involved in combining these two branches of robotics is one of the main reasons for the scarcity of applications.

This work is focused on picking an object from a table. For an AMMR to pick an object, its base must consider the reachability constraints of the arm. Indeed the base must be positioned in such a way that ensures the target object to be within the arm's range. The particular kinematic of each robotic arm limits its reachability, and thus, not all poses of the base near the object in question are valid.

Traditionally, such mobile manipulation operations have been solved using analytical planning and control methods [15]. These methods require explicit programming of the skills, which can be very costly and error-prone particularly in problems where decision making is complex. The performance of these models depends on how well the reality fits the assumptions made by the model. Due to the impossibility of predicting all the casuistry that may occur in dynamic and unstructured environments, these methods are generally impractical.

However, other works such as the one proposed by Stulp et al. in [16] challenge the practicality of analytic methods and raise the following questions: *"Does well-in-reach always imply that the target can really be reached, given the hardware and control software of the robot?, Can we have a least-commitment realisation of 'places' such that the robot can refine a 'place' as it learns more about the context (e.g. clutteredness) of the surroundings?, How can such a concept of 'place' take into account uncertainties about the robot's self-localisation and the estimated target position?*". As the authors mention, explicit programming to account for these factors is tedious, and more flexible solutions are needed.

Following that idea, the main objective of this work is to reduce the effort required to programme such complex behaviour by using AI-based methods that allow to learn it from experience.

3.1 State of the art

Well known traditional planning and control methods (such as those offered in the ROS navigation stack [17] for navigation and MoveIt! [18] for manipulation) have been widely used to build mobile manipulation skills [19]. Using these traditional planning and control libraries, it has been possible to develop robust mobile manipulation behaviours, which follow a logic pre-set by the programmer.

For instance, Dömel et al. focus on fetch and carry operations in industrial environments [20]. In that work, both the arm and the base are considered as independent systems. Although an reachability study is carried out to estimate the poses for the base that allow a successful manipulation, it is done offline, and thus, ignoring the environment's dynamics. In the approach proposed by Xu et al., a inverse-kinematics dataset is used to estimate the feasible positions for the AMMR's base to solve a pick-and-place operation with objects stored in trays. However, also in this work the generation of this database is carried out offline and ignoring the environment's dynamics. In addition, authors only consider a limited set of poses, since the continuous space of feasible poses is discretised to generate the dataset. In other methods such as [22, 23, 24] both the base and the arm are treated as a single kinematic chain, and also rely on traditional path planning and control methods to schedule the task. On the one hand, the computational cost of such a high dimensional planning is very high. On the other hand, the applications are designed to follow a

pre-established logic and although they attempt to model uncertainty in dynamic environments, the system is limited to the casuistry foreseen by the programmer. In addition to the approaches mentioned above, there are other methods, which do not consider the AMMR base and arm as a single system, but also not as fully independent systems. For instance, in works such as [25, 26] use the force feedback perceived on the arm is used to estimate velocity commands for the base.

As previously mentioned, methods based on such planning and control algorithms generally require explicit scheduling of behaviour. This process is very costly, particularly in applications where advanced decision making is required, due to the difficulty of foreseeing all the cases that may occur. In addition, such solutions are generally very inflexible and error-prone due to the impossibility of modelling all the uncertainty of dynamic industrial environments when those are programmed.

Alternatively, data-driven methods seek to alleviate the cost of programming complex behaviours and aim to learn such behaviours using real experience, i.e. data [27]. Early data-driven approaches to modelling behaviour used ML techniques such as deep learning (DL) or RL. For example, in [28] Lin and Goldenberg used DL to model a motion controller for an AMMR using real-world experience. This approach allowed them to use deep neural networks to model the uncertainties of the environment, which led to a more robust controller compared to traditional ones. Other works such as [16] use ML-based classifiers to learn the positioning of a mobile manipulator. In that work Stulp et al. propose to learn a concept of "place" (areas where the base of the mobile manipulator has to be placed to guarantee the reach of the arm to the object) for mobile manipulation operations using ML. Instead of learning a controller for the robot, the authors propose to learn a probabilistic ML model using experience obtained through trial and error. Their goal is to predict offline poses of the base which allow the arm to reach the object. As the authors claim, the explicit modelling of the problem is no longer required since the learnt models are grounded in real experience.

However, one of the disadvantages of supervised methods is the need to acquire and annotate data manually. In contrast, RL allows autonomous experience acquisition to learn behaviours by trial and error and only guided by a reward signal. In addition, this technology allows complex control policies to be learned. One of the first works to apply RL to mobile manipulation was [29], which were able to automate behavioural learning.

More recently, the combination of DL and RL, better known as DRL, has allowed to tackle complex decision-making problems that previously were unfeasible with DL or RL separately. It combines the ability of DL to model very high dimensional data with the ability of RL to model decision-making agents through trial and error [30]. In fact, DRL has become the *de facto* technology for learning complex decision-making through interaction with the environment, and has been successfully applied in multiple areas such as computer vision, robotics or games [31]. Particularly, DRL has allowed to obtain remarkable performance in robotics, specially in applications that require advanced decision-making. Applications range from manipulation [32, 33, 34], to autonomous navigation [35] and locomotion [36, 37].

Kalashnikov et al. demonstrate the potential of DRL applied to robotic manipulation in [38]. In that work the authors propose a vision-based self-supervised DRL framework, dubbed *Qt-opt*, to learn a hand-eye coordinated grasping policy. In fact, the policy is learnt using real grasping experience acquired from 7 robotic arms. DRL has been also applied to learn the manipulation of nonrigid objects such as cloths [39] in simulation. The method proposed by Jangir et al. is based on the Deep Deterministic Policy Gradient (DDPG) [40] DRL algorithm. More recently, Kim et al. [41] use the Twin Delayed Deep Deterministic Policy Gradient (TD3)[42] DRL algorithm to solve the path planning problem with 2/3-DoF manipulators in simulation, and show that TD3 can be used to plan smoother paths compared to traditional algorithms such as Probabilistic Roadmap Planning [43].

Concerning DRL-based applications for robotic navigation, Tai et al. solve the problem of mapless navigation using an asynchronous variant of DDPG [44]. Specifically, they learn a velocity controller for the mobile base, using as input 10-dimensional laser sensor readings, besides to the relative position of the robot with respect to the target. The controller is first trained in simulation and then deployed in the real robot. The large-scale 3D navigation of unmanned aerial vehicles is also tackled in [45], using a recurrent variant of the DDPG algorithm, called RDPG. In that work, Wang et al. use RDPG to solve the 3D large scale navigation in simulation as a Partially Observable Markov Decision Process (POMDP), with partially observable and uncertain states.

Robotic locomotion skills have been also successfully learnt using DRL. For instance, in works such as [46, 47] authors apply TD3 to learn controllers for biped and quadruped robots in simulation. Contrary to the general approach, Haarnoja et al. showed that is possible to learn complex robotic locomotion skills using real-world experience using their Soft Actor Critic (SAC) algorithm [36].

More recently, in the course of this research work, and due to the success that DRL has had in solving both manipulation and navigation control problems, work has also been done on mobile manipulation, thus combining these two worlds to generate more flexible solutions. For instance, Kindle et al. propose a whole-body control framework based on DRL to learn the picking of an object from a shelf [48]. Specifically, the authors learn to jointly control the base and the arm of the AMMR in simulation using the Proximal Policy Optimisation (PPO) [49] DRL algorithm. Wang et al. also propose a whole-body control approach for a non-holonomic mobile manipulator [50]. The authors use the PPO agent to learn a mobile manipulation task following a simulation-to-reality approach. Another approach is the one proposed by Honerkamp et al. that use both SAC and TD3 agents to learn the kinematic feasibility of mobile manipulation operations [51]. However, their system was only tested in simulation.

3.2 Goal

As just mentioned, methods based on traditional planning and control algorithms have proven to be useful in problems where the environment is relatively controlled and when they follow a simple and pre-established logic. The programming of this type of behaviour based on rules is very costly as it requires a high level of knowledge in the subject, and it is the programmer who has to foresee all the situations that may occur in reality.

Alternatively, data-driven methods allow to learn complex robotic behaviours based on experience gained through trial and error interacting with the environment. The technique par excellence in this field is DRL, which has given excellent results in modelling robotic behaviours such as object manipulation or navigation, which require a high level of decision making. However, at the time that this first research was carried out, the application of such algorithms to mobile manipulation was a totally unexplored world.

In this context and in relation to the TO5: "Dynamic robot planning based on cognitive capabilities" of the PICKPLACE project, the following research questions were stated:

- **RQ1:** Can DRL be used to model a mobile manipulation behaviour such as picking?

- **RQ2:** If so, is the behaviour learnt in simulation transferable to the real robot?

As it is common in RL-based applications, learning through the interaction with the environment is usually done in simulation. As it is the algorithm that controls the robot, especially in the early stages of learning its behaviour is often random, making it very costly to train the controller in the real world. In addition, it usually takes multiple runs before enough experience is gathered to learn stable behaviour, and this makes real-world training practically unfeasible.

Taking into account the aforementioned considerations, the first goal was to study the feasibility of applying DRL to a mobile manipulator for picking operations in simulation. In case of satisfactory results, the transfer of the learnt behaviour to the real robot was set as a second objective.

3.3 Developed approach

We aimed at learning a positioning policy for the AMMR's base that positions itself in the correct way to allow the arm to pick the object, taking into account the limited scope of the arm. To do so, we proposed to use a DRL agent to model the AMMR's base controller. The idea was to use the feedback from the arm planner to reward or penalise it depending on whether the picking was successful or not. Our intuition was to motivate the base controller to navigate to areas where the probability of successful picking was high.

As previously mentioned, we followed a two-phase approach. In a first approach, a DRL-based control architecture was developed for simulated environments, and DDPG and PPO algorithms were selected to learn the positioning policy. The objective in this first approach was to validate the feasibility of the approach, and therefore, the validation was also carried out in simulation. In the second approach, both the simulation and the control architecture were first improved with the deployment of the controller to the real robotic environment in mind. Then, the TD3 algorithm was selected to model the positioning policy. This algorithm was compared with those used in the first approach from the point of view of performance and reliability in

the simulated environment. Finally, the TD3-based base controller was validated in the real robotic environment.

3.3.1 Theoretical background

3.3.1.1. Introduction to RL

According to [52], a RL solution to a control problem is defined as a finite-horizon Markov Decision Process (MDP). At each discrete time-step t the agent observes the current state of the environment $s_t \in S$, takes an action $a_t \in A(s_t)$, receives a reward $r : S \times A \to \mathbb{R}$ and observes the new state of the environment s_{t+1}. At each episode of T time-steps, the environment and the agent are reset to their initial status. The goal of the agent is to find a policy, deterministic $\pi_\theta(s)$ or stochastic $\pi_\theta(a|s)$, parameterised by θ under which the expected reward is maximised. Figure 3.1 depicts a typical RL framework.

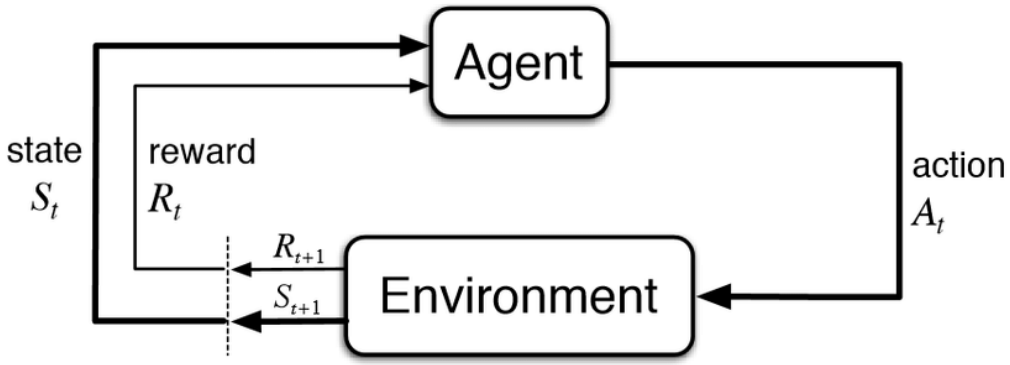


Fig. 3.1.: Reinforcement learning framework.

A trajectory τ is a sequence of states and actions in the environment (see Equation 3.1). Trajectories are also known as episodes or rollouts. The first state of the environment s_0 is randomly sampled from the start-state distribution which is usually denoted as ρ_0 (see Equation 3.2).

$$\tau = (s_0, a_0, s_1, a_1, ...) \tag{3.1}$$

$$s_0 \sim \rho_0(\cdot) \tag{3.2}$$

What happens to the environment between the state at time t, s_t, and the state at time $t+1$, s_{t+1}, are called state transitions and those are governed by natural laws of the environment. These transitions only depend on the most recent action given by the policy π and can be deterministic (Equation 3.3) or stochastic (Equation 3.4).

$$s_{t+1} = f(s_t, a_t) \tag{3.3}$$

$$s_{t+1} \sim P(\cdot|s_t, a_t) \tag{3.4}$$

The reward function has vital importance in RL and its most common definition can be seen in Equation 3.5.

$$r_t = R(s_t, a_t) \tag{3.5}$$

The goal of a RL agent is to maximise the cumulative reward over a trajectory. Usually the infinite-horizon discounted return is used, which is the sum of all rewards obtained by the agent, but discounted by how far off in the future they are obtained. To this end a discount factor $\gamma \in (0, 1)$ is used. The infinite-horizon discounted return is defined in Equation 3.6.

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t \tag{3.6}$$

Having said that, the main goal of RL is to find a policy which maximises the expected return. Supposing that we have an environment with both stochastic state transitions and stochastic policy, the probability of a T-step trajectory is defined in Equation 3.7.

$$P(\tau|\pi) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t) \tag{3.7}$$

Thus, the expected return $J(\pi)$ is defined as shown in Equation 3.8.

$$J(\pi) = \int_{\tau} P(\tau|\pi)R(\tau) = \underset{\tau\sim\pi}{E}[R(\tau)] \tag{3.8}$$

The central optimisation problem of RL is shown in Equation 3.9, where π^* is the optimal policy.

$$\pi^* = \arg\max_{\pi} J(\pi) \tag{3.9}$$

3.3.1.2. RL/DRL algorithms

The best known categorisation of RL/DRL algorithms is based on how they deal with learning a policy, and it can be seen in Figure 3.2. On the one hand, model-free algorithms assume that there is no model of the system on which they are going to act, and base the learning on real observations obtained from the environment. On the other hand, model-based algorithms assume the existence of a model of the environment and therefore, instead of using real observations, they make queries to that model. This implies that all transitions between states are known and therefore it is possible to estimate which are the optimal actions to reach the desired state of the environment. In robotics applications, it is practically impossible to model a system as dynamic as a real robotic environment, and that is why model-free algorithms are the most widely used.

Early approaches to RL methods used discrete action and state spaces. The most common approach was to use Q-learning based algorithms. These algorithms are based on estimating the value of executing each possible action in each state (since both are finite), and finally executing deterministically the action that obtains the highest expected value. To that end, an approximator $Q_w(s,a)$ of the optimal action-value function $Q^*(s,a)$ is used, which is usually optimised using the Bellman equation (see Equation 3.10). Algorithms such as DQN [53], HER [54], QR-DQN [55] and C51 [56] are based on the idea of Q-learning.

$$Q^*(s_t,a_t) = \underset{s_{t+1}\sim P}{E}[r(s_t,a_t) + \gamma \max_{a_{t+1}} Q^*(s_{t+1},a_{t+1})] \tag{3.10}$$

The deterministic policy $\pi(s) = a$ is defined in Equation 3.11, which at each state s, selects the action a out of the possible actions that gives the highest Q-value.

$$a = \arg\max_{a} Q_w(s,a). \tag{3.11}$$

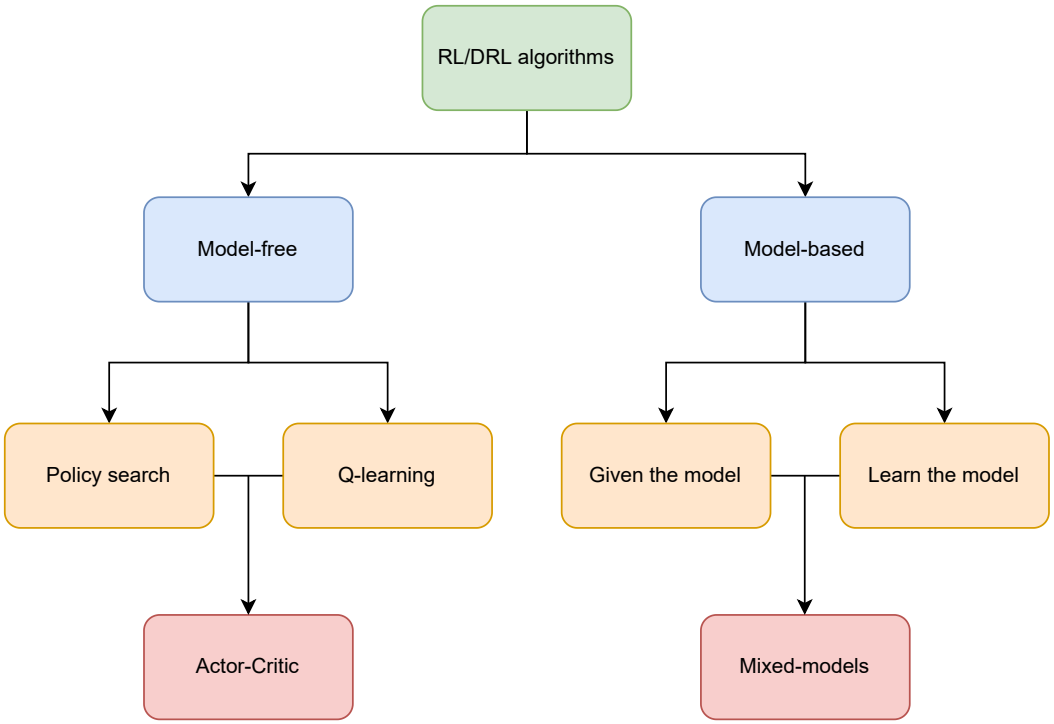


Fig. 3.2.: Taxonomy of RL/DRL algorithms.

Q-learning-based methods assume that a deterministic agent can be optimal. However, if we wanted to learn an agent that plays the game of rock/paper/scissors, for instance, a deterministic agent is very unlikely to be optimal and action probabilities need to be considered. Moreover, concerning to robotics, continuous spaces (e.g. Cartesian space or joint space) are used, but discretisation of these spaces does not usually give good results.

Taking these limitations into account, policy search algorithms were born, which directly seek to optimise policies. This kind of methods try to learn a explicit mapping between states and actions $\pi_\theta(a|s)$, and their goal is to maximise the expected return by taking small steps in the direction of the policy gradient $J(\pi_\theta)$. Learning such a mapping, for example by using function approximators which have trainable parameters, allows (1) stochastic policies to be modelled and (2) continuous state/action spaces to be tackled. This kind of learning usually also involves to learn an approximator $V_\theta(s)$ of the on-policy value-function $V^\pi(s)$ (see Equation 3.12), which is used during the policy optimisation process. The estimation

of the value-function is also based on the Bellman equation. Algorithms such as PPO [49], TRPO [57] and A3C [58] are examples that follow the idea of policy search.

$$V^{\pi}(s_t) = \underset{a_t \sim \pi, s_{t+1} \sim P}{E}[r(s_t, a_t) + \gamma V^{\pi}(s_{t+1})] \tag{3.12}$$

Actor-critic methods combine both Q-learning and policy search methods and are able to take advantage of the strengths of both of them. Best known examples of this kind of methods are DDPG [40], TD3 [42] and SAC [36]. Although algorithms such as PPO or A3C do not use the idea of Q-learning, these are also usually considered as actor-critic algorithms as they use value-functions V to help the optimisation of the policy.

Depending on how they acquire experience, RL/DRL algorithms are divided into two groups, on-policy and off-policy. On-policy algorithms expect that the experience used to optimise their behaviour policy is generated by the same policy. Off-policy methods, instead, can use the experience generated by another policy to optimise it's behaviour policy. Those methods are said to be able to better explore the environment than on-policy methods because they use a more exploratory policy to get experience. Q-learning methods usually optimise the Q-function off-policy, meaning that each update can use the data collected at any time during training, regardless of how the agent was choosing to explore the environment when the data was obtained. Concerning policy search algorithms, to get an unbiased estimate of the policy gradient, the trajectories need to be sampled from the current policy, and thus, policy search methods are usually on-policy.

3.3.1.3. DRL and robotics

The combination of DL and RL has resulted in DRL, which allows us to tackle problems with complex decision making that were previously unfeasible. Basically, what DRL proposes is to use deep neural networks to model policies π_θ, value functions V_w and action-value functions Q_w. Neural networks are excellent function approximators and the idea is to combine them with RL algorithms to learn complex behaviours.

On the one hand, the use of neural networks has allowed algorithms based on Q-learning to tackle problems with continuous state spaces. Although these algorithms

are designed to work with a discrete set of actions, it was a breakthrough and allowed more intelligent agents to be learned. For example, Mnih et al. proposed the DQN algorithm and used it to learn control policies on the ATARI video game directly from pixels [59].

On the other hand, as far as policy search algorithms are concerned, the application of neural networks to model deterministic $\pi(s)$ or stochastic $\pi(a|s)$ policies allowed to directly map continuous state spaces to continuous action spaces. This made it possible to model agents that were able to control systems through actions in continuous spaces, and gave excellent results in areas such as robotics. For example, Lillicrap et al. were among the first to combine ideas from Q-learning and policy search to learn continuous control policies, and demonstrated excellent results in robot control [40].

In our approach, a DRL algorithm is used to learn a positioning policy for the mobile manipulator's base, which drives it to zones that allow a successful picking of an object on a table by sending continuous velocity commands. Those DRL algorithms need a huge amount of experience to learn complex robotic skills and thus, it is unfeasible to train them acquiring experience in the real world. In addition, the actions taken by the robot in the initial learning iterations are nearly random, and both the robot and the environment might end up damaged as a result. Therefore, DRL algorithms are usually trained in simulation. Learning in simulation enables a faster experience acquisition and avoids material costs. However, in current simulators it is practically impossible to perfectly model all the dynamics of a real robotic environment and therefore there is a gap, known as the sim-to-real gap, which makes it difficult to apply the behaviours learned in simulation to the real environment. Although there are methods such as *domain randomisation* [60], this is yet an unsolved issue.

3.3.2 Preliminary study in simulation

3.3.2.1. Robotic environment

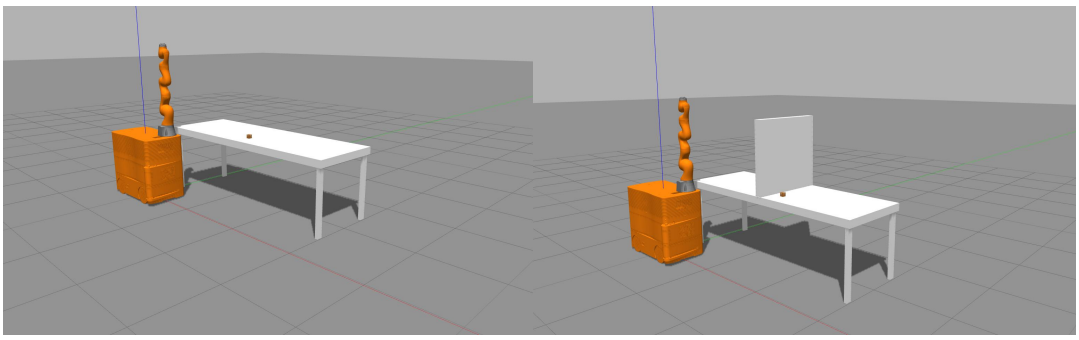
The AMMR used in the preliminary study was the Kuka Miiwa collaborative industrial robot, consisting of an omnidirectional base and a collaborative 7-DoF Kuka Iiwa arm (see Figure 3.3). This robot has been previously used in industrial applications such as [20]. The first proof of concept was only carried out fully in simulation and the

development was based on open source robotic tools such as the ROS middleware [61] and the Gazebo simulator [62], which is perfectly integrated into the ROS ecosystem.

Fig. 3.3.: Kuka Miiwa AMMR.

The rather simplistic preliminary simulated environments are depicted in Figure 3.4. The environments consisted of the AMMR, a table and the object to be picked up placed on the table, and the goal in all environments was common: the base should learn to position itself in zones near the table ensuring the arm could reach the object. From one environment to the other, a wall was introduced in the middle of the table to reduce the feasible poses for the base to pick up the object and thus make the task more difficult.

As the simulation was integrated in ROS, different services were implemented to interact with the simulation (e.g. randomising the pose of the object or the robot). To control the robot's omnidirectional base via velocity commands, the plugin *gazebo planar move* was used, which also publishes the robot's odometry.



(a) Environment 1 simulated in Gazebo. **(b)** Environment 2 simulated in Gazebo.

Fig. 3.4.: Simulated environments in Gazebo.

3.3.2.2. DRL-based control architecture

The implemented ROS-based modular control architecture is shown in Figure 3.5. All the modules that make up the system were built up as ROS nodes. The developed

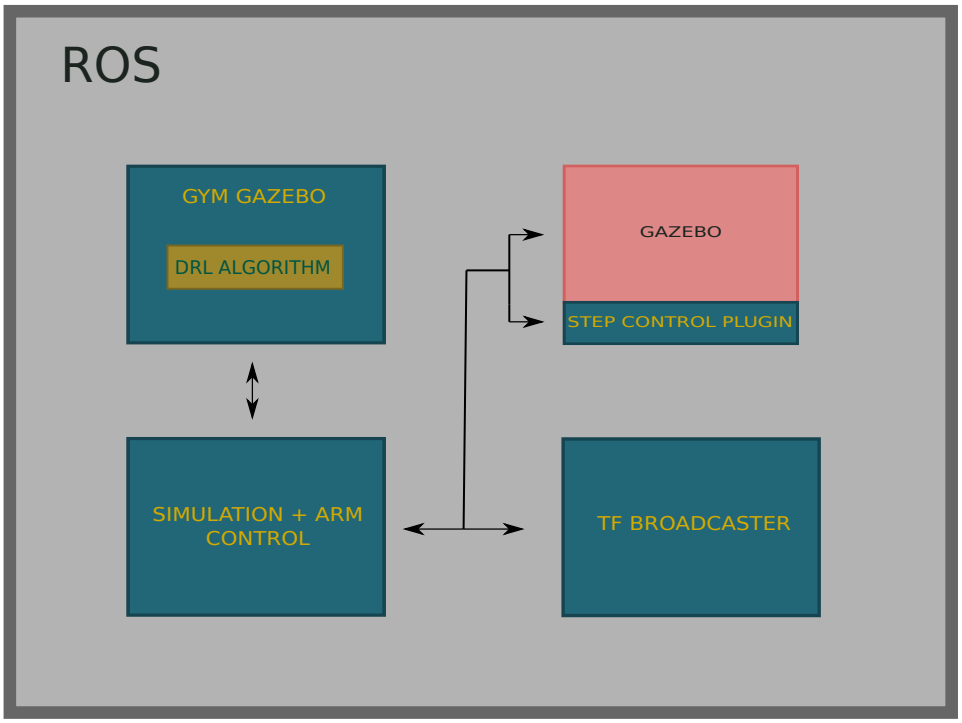


Fig. 3.5.: Implemented control architecture.

sub-modules were the following:

- *OpenAI Gym* is a library designed to develop and compare DRL algorithms by providing a standard API to communicate between learning algorithms and

environments. Indeed, this communication interface has become a standard to design DRL algorithms and environments and most of the libraries follow it [63]. Gym Gazebo is an extension of Gym and enables to create robotic environments in Gazebo and offers the same simple interface to the algorithm to be able to interact with these environments [64]. Therefore, the DRL algorithms in charge of controlling the mobile base were wrapped using this library, and the services to interact with the simulation were developed following this standard.

- The simulation and arm control node acted as a bridge between the DRL-based controller and the simulation, and was in charge of controlling the entire training flow. Besides, it also carried out the path planning and control of the arm using MoveIt!, since those were done using traditional planning and control algorithms and not using DRL. This library enabled us planning collision free trajectories since it used an internal planning scene where objects could be added to be taken into account when trajectories were planned.

- The TF broadcaster node published the relationships between all the coordinate systems present in the robotic environment and offered the possibility to dynamically modify them at run time.

- The simulation in Gazebo was already implemented as a ROS node and we developed the step control plugin to control the speed of the simulation. The learning process of the DRL-based controller was carried out in an episodic manner during a fixed number of discrete time-steps, and therefore it was important to accelerate the simulation as much as possible to reduce the training times.

3.3.2.3. DDPG and PPO as base controller

DDPG and PPO were the agents selected to model the base controller for the AMMR, both of them implemented in the *Stable baselines* library [65]. The former is a deterministic and off-policy algorithm, while the latter is stochastic and on-policy.

PPO follows the actor-critic architecture and it is based on the Trust-Region Policy Optimisation (TRPO) [57] algorithm. This algorithm aims to learn a stochastic policy $\pi_\theta(a_t|s_t)$ that maps states into Gaussian distributions over actions. In addition, the

critic is a value function $V_w(s_t)$ that yields the mean expected reward in state s_t. This algorithm has the benefits of TRPO and in general of trust region based methods but it is much more simple to implement it. The intuition behind trust-region based algorithms is that at each parameter update of the policy, the output distribution can't diverge too much from the original distribution. Both the policy $\pi_\theta(a_t|s_t)$ and the value function $V_w(s_t)$ are implemented as *Multi Layer Perceptron* (MLP) neural networks. On the one hand, at each time-step the actor receives the state observation s as input, and predicts the velocity command a that maximises the future expected reward. As it is an stochastic algorithm, each variable of the action is encoded as the mean and standard deviation representing a Gaussian distribution. On the other hand, the value function gets as input the state and outputs the expected average reward for that state, which is then used to optimise both the actor and the critic.

DDPG combines elements of value function based and policy gradient based algorithms, following the actor-critic architecture. This algorithm aims to learn a deterministic policy $\pi_\theta(s) = a$ and it is derived from the deterministic policy gradient theorem [66]. Following the actor-critic architecture, DDPG uses an action-value function $Q_w(s, a)$ as critic to guide the learning process of the policy and it is based on the deep Q-network (DQN) [53]. Similar to PPO, in this case also both the actor and the critic are modelled as MLPs. However, the policy $\pi_\theta(s) = a$ receives as input the state observation s and predicts a deterministic action a. In the case of the critic $Q_w(s, a)$, it gets as input both the state observation and the action, and predicts a single Q value indicating the expected future reward of executing the action a_t in the state s_t.

State/action spaces and reward functions Agents described previously were applied to learn two tasks with different complexity. In the case of the first test, the objective was to learn a base controller that, by sending velocity commands, would position the AMMR's base so that the object was within reach for the arm. In the second case, although the objective was similar, a wall was introduced in the middle of the table to see if the controller was able to learn to discard the poses behind the wall. Besides, in the first environment the target pose was random, while in the second environment it was fixed. Therefore, the vector of features that made up the state of the system was different in each case. The environment state feature vector and reward function used in each environment were the following:

Environment 1: The environment state defined in Equation 3.13 is composed of the features defined in Table 3.1.

Robot's position in world coordinate system	$p_r = [x_r^w, x_r^w]$
Robot's rotation on z axis in world coordinate system	yaw_r^w
Robot's linear velocities on x and y axes	v_x, v_y
Robot's angular velocity in z axis	ω_z
Object's position in world coordinate system	$p_{obj} = [x_{obj}^w, y_{obj}^w]$
Distance between the robot and the object	$d(p_r, p_{obj})$
Remaining time steps to end the episode	t

Tab. 3.1.: Variables that define the state of Environment 1.

$$s = [x_r^w \;\; y_r^w \;\; yaw_r^w \;\; v_x \;\; v_y \;\; \omega_z \;\; x_{obj}^w \;\; y_{obj}^w \;\; d(p_r, p_{obj}) \;\; t] \in \mathbb{R}^{10} \tag{3.13}$$

The reward function can be seen in Equation 3.14.

$$r(s_t, a_t) = \frac{1}{d(p_r, p_{obj})} \cdot (1 - collision) - 0.5 \cdot \Delta v - 0.5 \cdot v\_high(v_r) + 100 * success \tag{3.14}$$

$$v\_high(a_t) = \sum_{i=1}^{3} a_{t_i}, \text{ if } a_{t_i} > k \tag{3.15}$$

Environment 2: The environment state defined in Equation 3.16 is composed of the variables defined in Table 3.2.

Robot's position in object's coordinate system	$p_r = [x_r^{obj}, x_r^{obj}]$
Robot's rotation on z axis in object's coordinate system	yaw_r^{obj}
Robot's linear velocities on x and y axes	v_x, v_y
Robot's angular velocity in z axis	ω_z
Distance between the robot and object's coordinate system origin	$d(p_r, \mathbb{O})$
Remaining time steps to end the episode	t

Tab. 3.2.: Variables that define the state of Environment 2.

$$s = [x_r^{obj} \;\; y_r^{obj} \;\; yaw_r^{obj} \;\; v_x \;\; v_y \;\; \omega_z \;\; d(p_r, \mathbb{O}) \;\; t] \in \mathbb{R}^8 \tag{3.16}$$

The reward function can be seen in Equation 3.17.

$$r(s_t, a_t) = \frac{1}{d(p_r, \mathbb{O})} \cdot (1 - collision) - 0.5 \cdot \Delta v - 0.5 \cdot v\_high(v_r) + 100 * success \tag{3.17}$$

The main difference was related to the pose of the target object. Regarding the former, the fact that the pose of the object was variable made it necessary for both the robot and the object to be referenced with respect to an external coordinate system. In the latter, however, this was not the case and the robot was referenced with respect to the object's coordinate system. Therefore, the major difference in the definition of the state was that in the former it was necessary for the agent to know the target pose. The action space was common in both environments and it can be seen in Equation 3.18. It was composed of the linear and angular velocities predicted by the agent and to be sent to the robot.

$$a = [v_x, \ \ v_y, \ \ \omega_z] \in \mathbb{R}^3 \tag{3.18}$$

As far as the reward functions are concerned, in both cases they were composed of four main components:

1. Distance reward: The idea was to reward the robot for driving near the target object. The closer to the object the higher the reward, unless the robot collided with the table. In that case, the distance reward was cancelled.

2. Velocity penalty: The robot was penalised for driving fast.

3. Acceleration penalty: The robot was penalised for high positive and negative accelerations, with the idea of not allowing the robot to make sudden movements.

4. Grasp reward: In the last time-step of the episode a grasp trial was carried out. In the case of success, the maximum possible reward was given to the robot with the idea of encouraging it to drive to the area that allowed a successful grasp.

Intuition says that it should be sufficient to give positive rewards when the robot meets the goal of picking up an object. However, this generally yields poor results and it is necessary to reward the robot for meeting sub-goals, in order to guide learning. This concept is known as reward shaping [67]. In our case, in addition to rewarding the agent when the main goal is met, we use other criteria such as distance to the goal or speed/acceleration to "shape" the reward function.

3.3.2.4. Training procedure

As previously mentioned, the experimentation was carried out in two environments of varying difficulty. In each of the environments, training was performed with both DDPG and PPO. The duration of each training period was 5M discrete time-steps. The training was performed in an episodic manner, where at the beginning of each episode of $T = 512$ discrete time-steps (about 4s) the initial position of the robot was randomly selected. In the case of the first environment, the target pose was also randomly selected at the first time instant of each episode $t = 0$.

The process was as follows: At each time instant of the episode of duration T, the DRL-based base controller sent velocity commands to the robot base based on observations of the state of the environment. Taking into account both the state of the environment and the predicted action, the corresponding reward was calculated. At the last time instant of the episode $t = T - 1$, the robot's navigation was stopped and an attempt was made to plan a trajectory with the arm to the target object. If the planning was successful, it meant that the object was within the range of the AMMR's arm, and therefore received the maximum possible reward. Thus, a new attempt was made every T instants of time. In addition, each time the system entered a terminal state, a new episode was started. The following terminal states were defined:

- The robot collides with the table
- Robot poses out of limits
- $t = T - 1$

3.3.2.5. Evaluation metrics

A 5-episode evaluation period was conducted every 20 training episodes. In each evaluation period, the following metrics were used to quantify the performance of the learned agent:

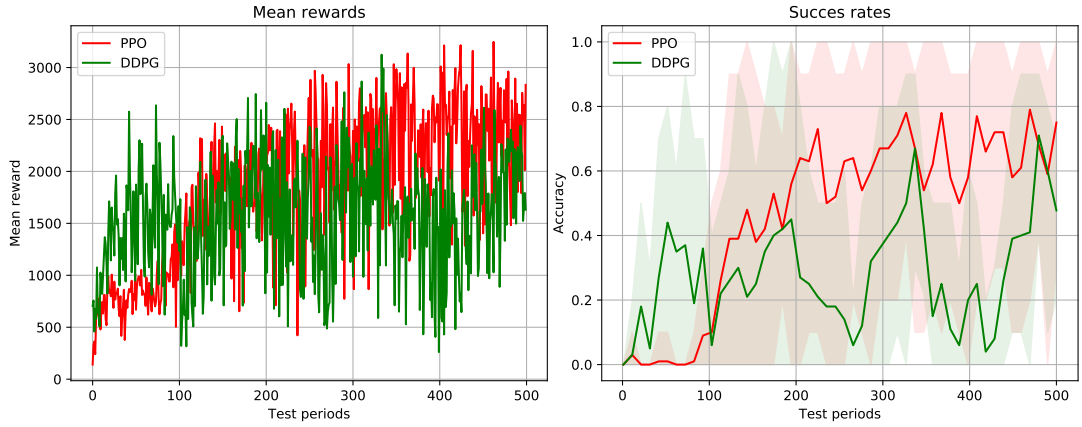
- Average accumulated rewards: In each evaluation episode the accumulated reward (i.e. the sum of the rewards during the T time-steps of the episode)

was computed. Finally, the average accumulated reward was calculated among the 5 episodes of each evaluation period.

- Success rates: Per each evaluation period of 5 episodes, a success rate was computed which told us how many times the learnt base controller was able to drive the mobile manipulator's base to areas where the target was in the scope of the arm. For illustration purposes we show the average success rates considering 10 evaluation periods.

3.3.2.6. Results

The goal of the first test was to learn a speed controller to drive the mobile manipulator's base close to the table, the arm to be able to plan a trajectory up to the object. The results obtained with both PPO and DDPG algorithms are depicted in Figure 3.6. On the one hand, Figure 3.6a shows the average accumulated rewards obtained in each of the 500 evaluation periods. On the other hand, the obtained success rates are depicted in Figure 3.6b. Due to the unstable nature of DRL algorithms, the success rates obtained vary considerably. Thus, to better understand the results, the mean value and the maximum/minimum values are shown per 10 test periods.

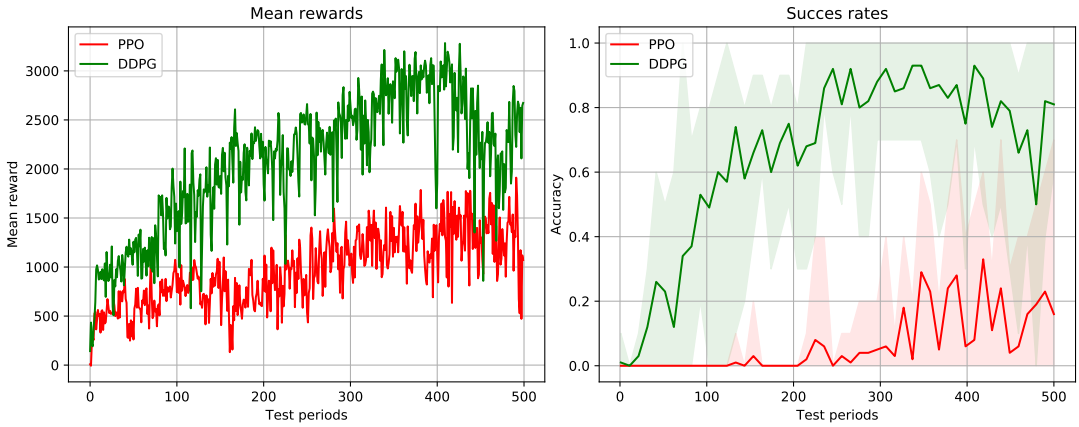


(a) Average accumulated rewards. (b) Success rates.

Fig. 3.6.: Test 1 results.

The algorithm that achieved the most stable behaviour, reaching an average success rate of 80%, was PPO. In the case of DDPG, although at one point in the learning process it reached an average success rate of over 60%, it showed a forgetful behaviour.

The objective of the second test was similar but with a higher degree of complexity. The results obtained with both algorithms can be seen in Figure 3.7. The average accumulated rewards obtained during learning are shown in Figure 3.7a, and similarly to the first test they also oscillate. However, both the mean cumulative rewards and success rates shown in Figure 3.7b indicate that in this case the behaviour learned by DDPG far outperformed PPO. In fact, the latter achieved at most an average success rate of about 30%.



(a) Average accumulated rewards. **(b)** Success rates.

Fig. 3.7.: Test 2 results.

The results show that a relatively high rate is possible in mobile manipulation tasks. However, the unstable nature of the DRL algorithms make it difficult to obtain a robust behaviour, mainly due to the following causes:

- The sensitivity of DRL algorithms to hyperparameters hindered finding the best parameter combination to get a robust and stable behaviour. Despite the fact that an optimisation over those hyper-parameters could be made, the large training times made this process very expensive and the default values proposed in the literature were used.

- The definition of the reward function was another key point in the learning process, as it was entirely guided by this signal. In fact, DRL algorithms are highly sensible to it and an incorrect definition of the function leads to undesired behaviours. A logical approach could be to give a reward to the robot in the last step only if the arm plans a trajectory, but unfortunately, using only sparse rewards did not work well. In several goal reaching applications either with arms or mobile bases, a distance dependant reward is proposed.

In our application, this encourages the robot to navigate close to the object so that the arm is able to plan a trajectory. After some tests, we saw that a non-linear distance function accelerates the learning process.

Concerning the algorithms tested in this work, the results obtained showed that the behaviour of the algorithms was dependant on several properties of the environment such as the state/action codification and the reward function definition. In addition, the network architecture used to encode both policies and value-functions had a large effect on the learning process. Despite the same network architecture and hyperparameters were used in both tests, the results obtained were very different. Therefore, each environment needs the algorithm to be tuned in the best way to solve the problem and that is why the learnt policies are hardly reproducible.

3.3.3 Simulation-to-reality transfer of the learned behaviour

In view of the promising results obtained in the first approach, the second objective was defined as the deployment of the controller learned in simulation into the real robot.

3.3.3.1. Robotic environment

The AMMR selected for this second iteration can be seen in Figure 3.8, which was developed at Tekniker by combining a Segway omnidirectional mobile base, with a collaborative Kuka Iiwa arm with 7 degrees of freedom.

To get closer to the final goal of deploying the AMMR base controller into the real robot, it was first necessary to develop a much more realistic simulation of the workshop where the controller was to be validated. In other words, the goal was to reduce the simulation-to-reality gap. This simulated environment was developed in Unity [68] as it is widely used in different sectors to create environments with a high degree of realism. The simulated environment can be seen in Figure 3.9. The simulation was integrated in the ROS framework using the ROS# library [69] and following the OpenAI Gym interface. The robots were modelled in the URDF format to be easily loaded to the simulated world.

Fig. 3.8.: AMMR created at Tekniker.

In addition to the simulated environment itself, a simulated arm driver was also implemented to integrate with MoveIt!, besides to a speed driver for the base equivalent to Gazebo's *gazebo planar move*. One of the main limitations of that plugin is that it only takes into account the kinematics of the robot and therefore when commanding the base with speed commands the accelerations are not taken into account. Therefore, in the new implementation for Unity, the base's dynamics were introduced in a simplified way, with the aim of bringing the simulation closer to reality.

It is widely known that real localisation systems are noisy. However, the developed base driver, which also publishes the robot's odometry, does not introduce any localisation uncertainty. To overcome this lack and make the system aware of imprecise localisation, during the training process we added Gaussian noise to the

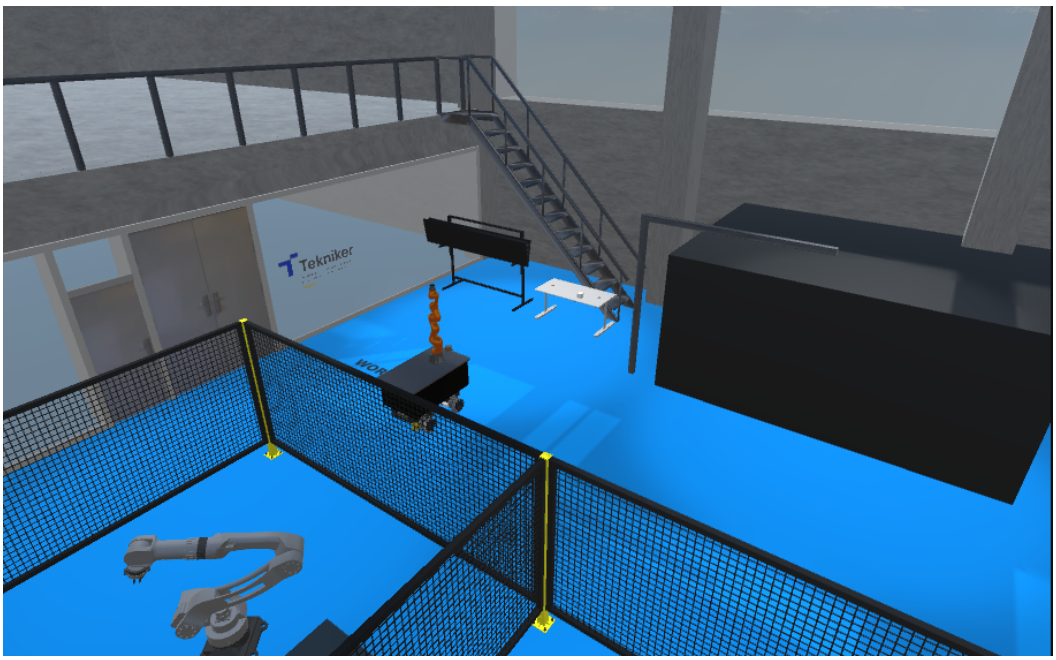


Fig. 3.9.: Simulated environment in Unity.

robot's pose given by the simulated system at each time step. The modifications made in the localisation are detailed in Equation 3.19.

$$\begin{aligned} x_r^w &= x_r^w + \mathcal{N}(0.0, 0.02) \\ y_r^w &= y_r^w + \mathcal{N}(0.0, 0.02) \\ \alpha_r^w &= \alpha_r^w + \mathcal{N}(0.0, \frac{\pi}{100}) \end{aligned} \tag{3.19}$$

3.3.3.2. Updated control architecture

The upgraded control architecture follows the same idea as the first approach and is depicted in Figure 3.10.

In this new iteration, the simulated environment was also developed following the standards defined by *OpenAI Gym* to facilitate the communication between the agent and the environment. The idea was to use this standard so that later the jump to the real environment would be direct. The major difference was in the agent acting as the brain, which was upgraded to one of the state-of-the-art algorithms. Specifically, the TD3 agent was used to model the base controller, which is an evolution of DDPG. The optimisations introduced to the system related to the upgraded DRL-based agent are detailed in the following section.

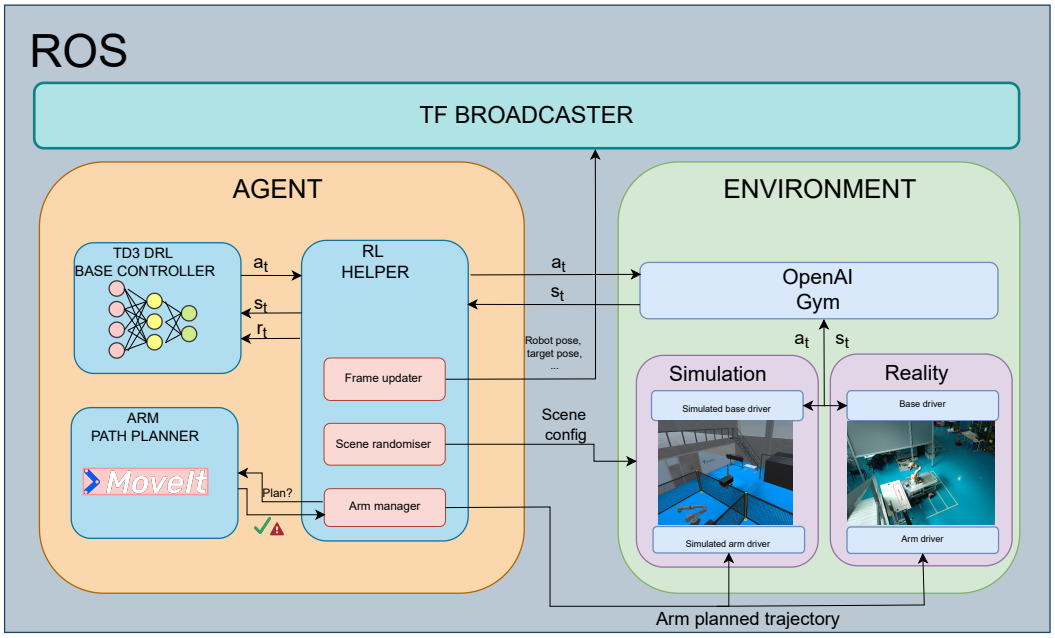


Fig. 3.10.: Proposed control architecture to control the AMMR.

In addition to the base controller, the control architecture is composed of another two key modules: The RL helper and the Arm path planner. On the one hand, the Arm path planner node determines at each instant whether the target is reachable for the robotic arm or not. Then, this information is used to calculate the reward given to the robot. On the other hand, the RL helper node is in charge of orchestrating both the learning and the execution of the agent and acts as a bridge to communicate with the environment. In addition to coordinating both arm and base control, it also takes care of other essential tasks such as randomising the scene using the "scene randomiser" library, or updating the relationships between coordinate systems via the "frame updater" library.

3.3.3.3. TD3 as base controller

To model the agent, the TD3 actor-critic algorithm was selected, which solves some major stability concerns of its predecessor DDPG. Although DDPG has shown great performance learning some robotic skills, it usually tends to be very sensitive with respect to the tuning hyper-parameters. Similarly to its predecessor, this DRL algorithm also learns a deterministic policy $\pi_\theta = a$, dubbed actor. TD3 also uses action-value functions Q_w, called critics, to guide the learning process. In addition, target networks, π' and Q' are used for both the actor and critic respectively, which

are updated slower, leading to consistent targets y during temporal difference (TD) learning [70]. On the one hand, the TD error shown in Equation 3.20 is used to optimise the critics. On the other hand, the actor is optimised according to the Deterministic Policy Gradient (DPG) theorem [66], which can be seen in Equation 3.21. The training process is carried out off-policy, getting training experience from a replay buffer, and thus, removing temporal correlation. The network architecture of TD3 is depicted in Figure 3.11.

$$\begin{aligned} \tilde{a} &\leftarrow \pi_{\theta'}(s) + \epsilon, \qquad \epsilon \approx clip(\mathcal{N}(0, \tilde{\sigma}), -c, c) \\ y &\leftarrow r + \gamma min_{i=1,2} Q'_{w'_i}(s_{t+1}, \tilde{a}) \\ w_i &\leftarrow min_{w_i} \frac{1}{N} \sum (y - Q_{w_i}(s, a))^2 \end{aligned} \tag{3.20}$$

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum \nabla_a Q_{w_1}(s, a)|_{a=\pi_\theta(s)} \nabla_\theta \pi_\theta(s) \tag{3.21}$$

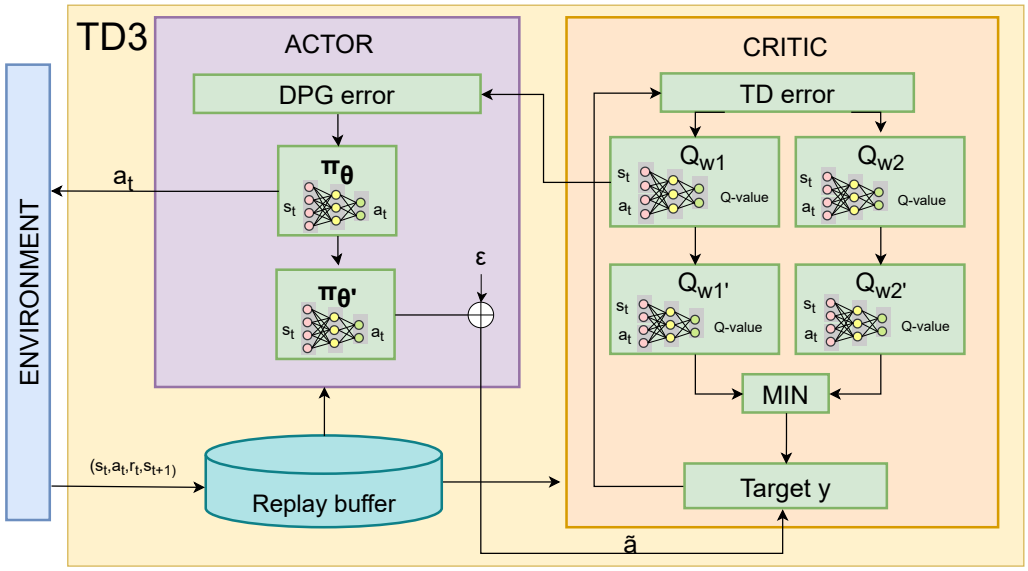


Fig. 3.11.: Network architecture of TD3.

The main improvements introduced by TD3 that alleviate stability issues are the following:

1. TD3 learns two Q-functions, $Q_{w_1}(s, a)$ and $Q_{w_2}(s, a)$, instead of one (hence "twin"), and uses the smaller of the two Q-values to form the targets in the Bellman error loss functions.

2. TD3 updates the policy (and target networks) less frequently than the Q-function.

3. TD3 adds noise to the target action, to make it harder for the policy to exploit Q-function errors by smoothing out Q along changes in action.

State/action spaces In addition to the agent itself, improvements were also made both to the state observation and reward function. The environment state $s \in \mathbb{R}^{13}$ (see Equation 3.22) was composed of the variables defined in Table 3.3.

The pose of the base of the arm with respect to the world that is composed of the x and y coordinates and α rotation, where α is the rotation in z axis	$p_r^w = [x_r^w, y_r^w, \alpha_r^w]$
The predicted action in the previous time-step	$a^{t-1} = [v_x^{t-1}, v_y^{t-1}, \omega_z^{t-1}]$
The observed linear and angular velocities	$v' = [v'_x, v'_y, \omega'_z]$
The position of the target with respect to the world	$p_{target}^w = [x_{target}^w, y_{target}^w]$
The distance between the base of the arm and the target	d_r^{target}
The normalised time-step in the episode	$\frac{t}{T}$

Tab. 3.3.: Variables that define the environment state.

$$s = [p_r^w,\ \ a^{t-1},\ \ v',\ \ p_{target}^w,\ \ d_r^{target},\ \ \frac{t}{T}] \in \mathbb{R}^{13} \tag{3.22}$$

The main difference in the state observation compared to the one used in the preliminary test was the discrimination between the speed predicted by the agent a^{t-1} and the observed speed v'. As previously mentioned, the simulation developed in Gazebo did not include accelerations, and therefore, the velocity commands received by the robot were executed instantaneously. Consequently, both the predicted and observed velocities were always the same. As this is not the case here, we decided to include both velocities in the state.

Reward function The upgraded reward function is shown in Equation 3.23 and it was composed of the following components:

- Distance reward, *D*: The closer the robot was from the target, the higher was the distance reward. When it got closer than d_{thresh}, the distance reward

became constant to avoid crashing with the table. This component is described in Equation 3.24.

- Velocity penalty, *V*: The agent was penalised with a discount factor for moving with high velocities, particularly when the robot was closer than d_{thresh} from the goal (see Equation 3.25). This discount factor was applied to *D*.

- Collision penalty, *C*: If there was a collision between the robot and any element in the environment, this variable took the value 1. Otherwise, its value was 0.

- Acceleration penalty, *W*: The agent was penalised for predicting consecutive actions that would require high acceleration (Equation 3.26). To this end, the L^2 norm of the difference between a^t and a^{t-1} was used.

- Grasp reward, *G*: The robot tried to plan a trajectory up to the target in the last time-step of the epoch $t = T - 1$. If the planning succeeded, this variable took the value 1 and, instead, it took the value -1. In the other time-steps of the episode its value was 0.

$$r = w_d \cdot D \cdot V \cdot (1 - C) - w_c \cdot C - w_w \cdot W + w_g \cdot G \tag{3.23}$$

$$D = \begin{cases} c_1, & \text{if } d_r^{target} < d_{thresh} \\ \frac{1}{d_r^{target}}, & \text{if } d_r^{target} \geq d_{thresh} \end{cases} \tag{3.24}$$

$$V = \begin{cases} (c_2 - min(\|v'\|_2, c_3))^{\beta}, & \text{if } d_r^{target} < d_{thresh} \\ (c_4 - max(\|v'\|_2, c_5))^{\beta}, & \text{if } d_r^{target} \geq d_{thresh} \end{cases} \tag{3.25}$$

$$\text{where, } \beta = \frac{c_6}{max(d_r^{target}, c_7)}$$

$$W = \|a^t - a^{t-1}\|_2 \tag{3.26}$$

The weights w_d, w_c, w_w, w_g were used to determine the importance of each component of the reward function. Both the weights of each component and the constant values were experimentally obtained and are shown in Table 3.4.

w_d	1.0	w_c	1000.0
w_w	10.0	w_g	1000.0
c_1	100.0	c_2	1.0
c_3	1.0	c_4	2.0
c_5	0.5	c_6	2.0
c_7	0.65	d_{thresh}	0.8

Tab. 3.4.: Weights and constant values used for the reward function.

The most significant improvement in this updated version of the reward function was the use of a penalty factor for high speeds, relative to the proximity to the target. In the Gazebo simulation, when accelerations were not considered, the robot navigated to the target regardless of the magnitude of the speed, due to its ability to stop from one instant of time to the next. In the Unity simulation, however, when accelerations of the base are considered and in order to prevent collisions with the table due to a fast approaching to the goal, the discount factor mentioned above was introduced. The main goal of this discount factor was to give higher rewards for moving slowly near the target.

3.3.3.4. Training procedure

In the same way than in the preliminary study, the training of the agent was done following an episodic scheme during 4M discrete time-steps. Each training episode consisted of $T = 512$ discrete time-steps, with a time-step duration of $T_s = 30ms$. The duration of each episode was around 15s. At the beginning of each episode, both for training and validation, the initial poses of the robot and the target were randomly selected.

In this way, at each episode the robot had T time-steps to complete the positioning of the robot's base in a suitable zone for the arm to plan a trajectory up to the target. To that end, at each time-step the base controller observed the environment state, predicted the optimal velocity command, and after executing the action, it received a reward signal. At time-step $t = T - 1$ the path planning trial was done with the arm and the robot was rewarded or penalised depending on whether the planning succeeded or not.

Whenever the agent reached a terminal state, the simulation was reset, and a new episode started. The same three terminal states were considered:

1. A collision occurred between the robot and any other element in the scene.

2. The localisation of the robot fell outside the limits defined in the observation space.

3. The robot reached the last time-step of the episode $t = T - 1$.

The training algorithm of the agent can be seen in Algorithm 1.

3.3.3.5. Experimentation in simulation

In order to demonstrate the improvements introduced in both the definition of the state observation and the reward function for the new environment simulated in Unity, a comparison was made with the previous configuration, henceforth *baseline*. Thus, the PPO, DDPG and TD3 algorithms were trained using both the *baseline* and the *proposed* setups. We call setup the set of reward function, state and action spaces, and neural architectures. As a result, 6 training processes were carried out in total.

To that end, while the agents were being trained, a 5-episode evaluation period was performed at all $1e4$ training steps to assess the quality of the agent learnt at that time. In each evaluation episode, first both the initial poses of the robot and the target were randomly selected, and then the agent was executed to carry out a picking trial. Taking into account these evaluation periods, the performance and reliability of each algorithm/setup combination was analysed. On the one hand, performance was used to see how fast each algorithm converged based on the chosen setup. On the other hand, the reliability of the model showed how stable the learning process was and what risk there was in the short and long term of a large drop in performance. The algorithm used to evaluate the agents can be found in Algorithm 2.

Metrics To measure the performance of the algorithm/setup combinations while training the same metrics as in the first proof-of-concept were used:

Algorithm 1 Training of TD3-based agent

Initialise critic networks Q_{w_1} and Q_{w_2}, and actor network π_θ with random parameters w_1, w_2, θ.
Initialise target networks $w'_1 \leftarrow w_1$,$w'_2 \leftarrow w_2$, $\theta' \leftarrow \theta$
Initialise replay buffer $\mathbb{B}$
$env \leftarrow gym.environment()$
$i \leftarrow 0$
while $i < train\_steps$ **do**
    randomiseScene()
    $s_t \leftarrow env.reset()$
    for $t = 0$ .. $T - 1$ **do**
        Select action with exploration noise $a_t \approx \pi_\theta(s_t) + \epsilon$, $\epsilon \approx \mathcal{N}(0, \sigma)$
        $s_{t+1}, r_t, done \leftarrow env.step(a_t)$
        if $t = T - 1$ **then**
            $plan? \leftarrow doPickingTrial()$
            if $plan?$ **then**
                $r_t \leftarrow r_t + G$
            end if
        end if
        Store transition tuple (s_t, a_t, r_t, s_{t+1}) in $\mathbb{B}$
        if done **then**
            break
        end if
    end for
    if i mod $train\_freq$ **then**
        Sample minibatch of N transitions (s_t, a_t, r_t, s_{t+1}) from $\mathbb{B}$
        $\tilde{a} \leftarrow \pi_{\theta'}(s) + \epsilon', \epsilon' \approx clip(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
        $y \leftarrow r + \gamma min_{i=1,2} Q'_{w'_i}(s_{t+1}, \tilde{a})$
        Update the critics $w_i \leftarrow argmin_{w_i} \frac{1}{N} \sum (y - Q_{w_i}(s, a))^2$
        if i mod d **then**
            Update θ by the deterministic policy gradient
            $\nabla_\theta J(\theta) = \frac{1}{N} \sum \nabla_a Q_{w_1}(s, a)|_{a=\pi_\theta(s)} \nabla_\theta \pi_\theta(s)$
            Update the target networks:
            $w'_i \leftarrow \tau w_i + (1 + \tau) w'_i$
            $\theta'_i \leftarrow \tau \theta_i + (1 + \tau) \theta'_i$
        end if
    end if
    $i \leftarrow i + t$
    $s_t \leftarrow s_{t+1}$
end while

Algorithm 2 Evaluation of TD3-based agent

Initialise actor network π_θ with trained parameters θ.
$accumulated\_rewards \leftarrow []$
$successes \leftarrow 0$
$env \leftarrow gym.environment()$
for $i = 0$ .. $eval\_episodes$ **do**
    $accumulated\_reward \leftarrow 0$
    randomiseScene()
    $s_t \leftarrow env.reset()$
    for $t = 0$ .. $T - 1$ **do**
        Select optimal action $a_t = \pi_\theta(s_t)$
        $s_{t+1}, r_t, done \leftarrow env.step(a_t)$
        if $t = T - 1$ **then**
            $plan? \leftarrow doPickingTrial()$
            if $plan?$ **then**
                $r_t \leftarrow r_t + G$
                $successes \leftarrow successes + 1$
            end if
        end if
        $accumulated\_reward \leftarrow accumulated\_reward + r_t$
        if done **then**
            break
        end if
    end for
    $accumulated\_rewards.append(accumulated\_reward)$
end for
$success\_rate \leftarrow \frac{successes}{eval\_episodes}$
$average\_accumulated\_reward \leftarrow average(accumulated\_rewards)$

- Average accumulated rewards: In each evaluation episode the accumulated reward (i.e. the sum of the rewards during the T time-steps of the episode) was computed. Finally, the average accumulated reward was calculated among the 5 episodes of each evaluation period.

- Success rates: Per each evaluation period of 5 episodes, a success rate was computed which told us how many times the learnt base controller was able to drive the mobile manipulator's base to areas where the target was in the scope of the arm. For illustration purposes we show the average success rates considering 5 evaluation periods.

We also quantitatively measured the reliability of the algorithms during training. To do this, we used the metrics proposed in [71], which focus on measuring the risk and dispersion of the DRL algorithms. To compare the reliability of the agents during the training curves, we used the following metrics:

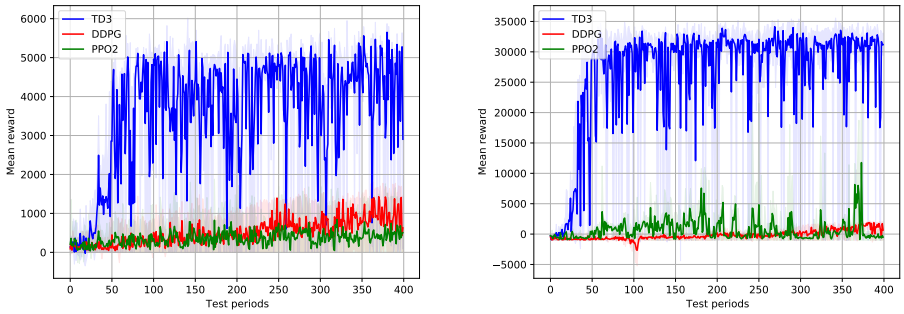
- Dispersion across Time (DT): Measures the dispersion of the average rewards during training. The dispersion was measured using the inter-quartile range (IQR), in this case the distance between the $75th$ and $25th$ percentiles. This metric measures short-term variability that corresponds to noisy training.

- Short-term Risk across Time (SRT): The goal of this metric is to measure the most extreme reward drop from one evaluation step to the next. To that end, the conditional value at risk (CVaR) or expected shortfall [72] of the differences in the rewards between successive evaluations was measured. CVaR measures the expected value below the α-quantile in the distribution formed by the reward differences. In our experiments we used the default value of $\alpha = 0.05$.

- Long-term Risk across Time (LRT): This metric measures the long-term risk during each training run by measuring the most extreme reward drop with respect to the peak, also called *drawdown* [73]. In this case, CVaR is applied to the *drawdown*.

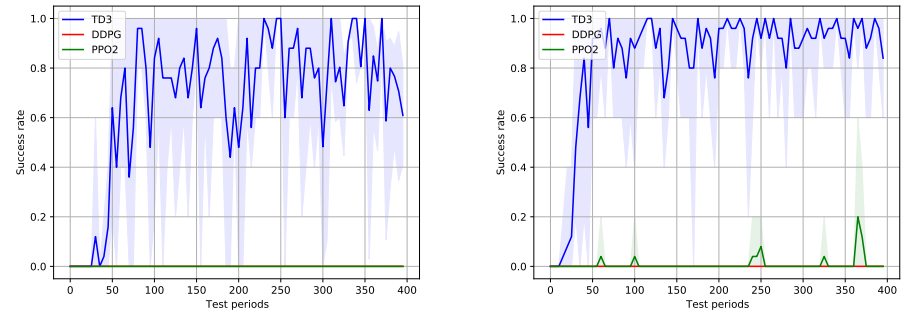
Due to the fact that each setup had a different reward range, to compute the across setup reliability metrics, those were converted into rankings as proposed in [71]. In fact, the rankings were first calculated per setup and finally, the across-setup mean rankings were obtained.

Results Figure 3.12 shows the cumulative average rewards and success rates obtained during training with the PPO, DDPG and TD3 algorithms.



(a) Average accumulated rewards per each evaluation period, using the *baseline* setup.

(b) Average accumulated rewards per each evaluation period, using the *proposed* setup.



(c) Average success rates per 5 evaluation periods, using the *baseline* setup.

(d) Average success rates per 5 evaluation periods, using the *proposed* setup.

Fig. 3.12.: Training average rewards and success rates with PPO2, DDPG and TD3 algorithms with the *baseline* and the *proposed* setups.

As it can be observed in Figures 3.12a and 3.12b, in both the *baseline* and the *proposed* setups TD3 outperformed DDPG and PPO algorithms. As the reward scale in both setups was different, it was difficult to visually assess in which setup the learned behaviour was better. In addition, as a consequence of the reward shaping it is not straightforward to know at each point if the robot has been able to successfully solve the task. Thus, Figures 3.12c and 3.12d show the average success rates obtained while training. Success rates prove the superior performance of TD3 during the learning process comparing to DDPG and PPO. Although the scale of the reward is different in both setups, its clear that the proposed setup, together with TD3, leads to a more stable learning process.

The rankings of the dispersion and risk metrics are depicted in Figure 3.13. The DT metric indicates that the algorithm with the highest dispersion across time was

TD3, which can also be seen in the average accumulated rewards in Figures 3.12a and 3.12b. Although the algorithm with the lowest dispersion was DDPG, success rates in Figures 3.12c and 3.12d indicate failure in solving the task. The SRT and LRT metrics indicate that TD3 was the algorithm with the lowest short-term and long-term risk.

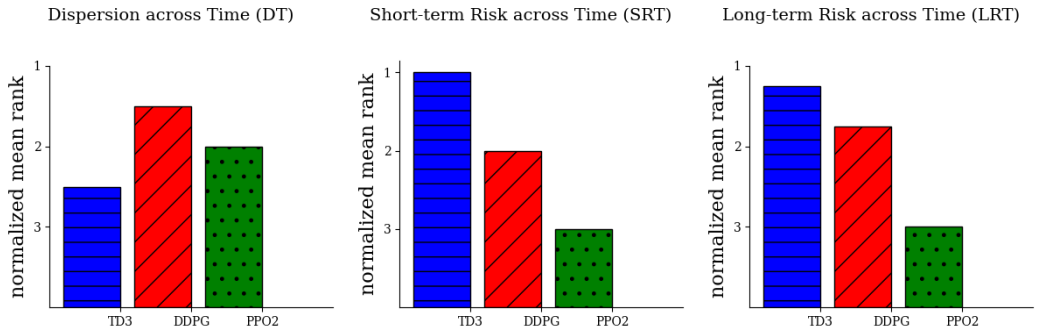


Fig. 3.13.: Mean across-setup reliability rankings of TD3, DDPG and PPO2 algorithms considering the average reward training curves. Rank 1 always means the best reliability.

3.3.3.6. Validation in the real robotic system

Setup for the real environment Before deploying the learnt controller in the real robot, there were some aspects to be considered. The localisation of the robot was one of the most important issues. In the real robotic system, the pose of the robot is usually provided by the odometry, which is known to contain a cumulative error. Thus, we used a 2D localisation system to correct the robot's pose given by the odometry using a map of the environment.

Specifically, we used the Adaptive Monte Carlo localisation (AMCL) algorithm as the 2D localisation system for the robot [74]. This probabilistic algorithm represents the localisation of the robot on a map as a particle filter. In fact, it fuses multiple data sources such as the odometry or laser scans to estimate the position of the robot in the map. We created the map of the real workshop using the *Gmapping* [75] algorithm and it can be seen in Figure 3.14. However, one of the main limitations of AMCL was that in our system the corrected odometry pose was estimated at a very low frequency ($3 - 7Hz$) due to the computational load of the sensor fusion process.

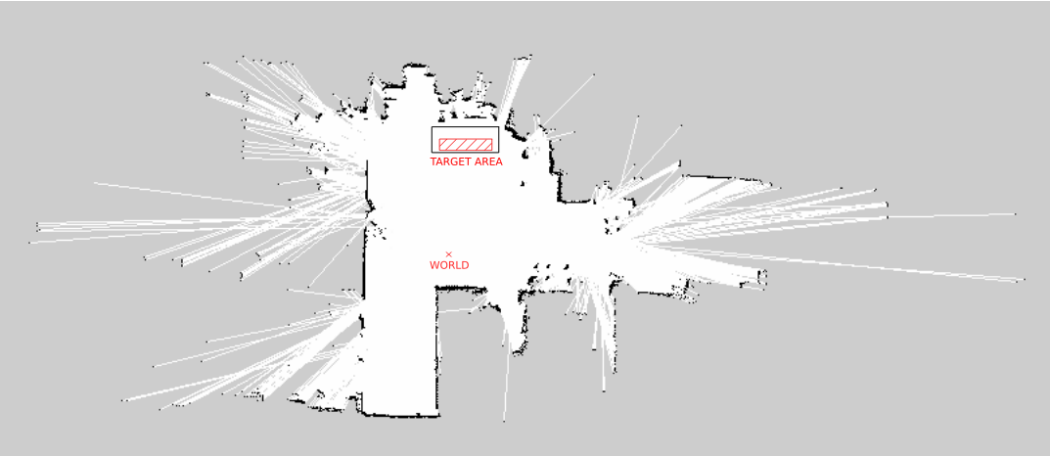


Fig. 3.14.: Map of the real scenario.

Arm path planning to the target Regarding the path planning with the arm, in the same way as in the simulation, the planning for the real mobile manipulator's arm was done using MoveIt!. However, the drift between the real and the estimated robot's pose provided by AMCL caused the estimation of the relative pose between the base of the arm and the target not to be accurate enough. Consequently, the planned path could be invalid for manipulating the item and an on-board vision system would be needed to correct the localisation error and successfully pick the part.

The main objective of our research was to demonstrate that the DRL-based agent was able to position the mobile manipulator in areas where the target object was within the range of the arm. Therefore, in addition to using the base location given by AMLC to perform the arm planning, we also used the real location calculated manually for verification purposes. The aim of doing this was to remove the localisation error from the system to see if the object was really within range.

Validation procedure To carry out the validation we selected the TD3 agent in conjunction with the *proposed* setup, due to the increased performance and reliability shown during the experimentation in simulation. The real robotic environment can be seen in Figure 3.15. The validation consisted of 20 trials where the robot's initial pose was randomly selected in each trial and, instead, the position of the target was randomly selected once per 5 trials. In the same way as in the simulation, each trial took T time-steps of navigation where the base controller was in charge of driving the robot's base to zones that ensured the target to be within the reach of the arm. At time-step $t = T - 1$, the arm tried to plan a trajectory up to the target

from the location of the robot in the environment at that moment, and the following measurements were taken:

1. The robot was set in a random initial pose and the DRL-based base controller was executed for T time-steps.

2. After T time-steps of navigation, once the robot was already positioned, at time-step $t = T - 1$ the arm made a trial to plan a trajectory up to the target. A successful plan meant that the target object was within the reach of the robot. This first planning was done using the AMCL localisation. Finally, a success rate was computed considering the results of the plans.

3. Due to the localisation uncertainty, it was impossible to deduce whether the target was in the arm's range. Therefore, with the aim of removing the localisation error, the real localisation of the robot was manually measured. This let us check whether the target was really reachable by the arm according to the arm's planner, and was used as ground truth.

The following measurements were taken in each trial:

- Difference between the observed and the measured real distance between the base of the arm and the target (Δd).

- Difference between the observed and real rotation on the z axis of the base of the arm with respect to the target ($\Delta\alpha$).

- Whether the base of the robot collided with the table when approaching the target ($coll.$).

- The attempt was considered successful if (1) there were no collisions during the positioning and (2), the robot positioned itself in such a way that the arm was able to plan a trajectory to the object. The planning was done using both the localisation given by AMCL ($amcl\_success$) and the real measured localisation ($real\_success$).

Validation results Table 3.5 shows that in 75% of the trials the robot successfully planned a trajectory to the target considering the localisation given by AMCL ($amcl\_success$). The mean errors between the estimated and manually measured

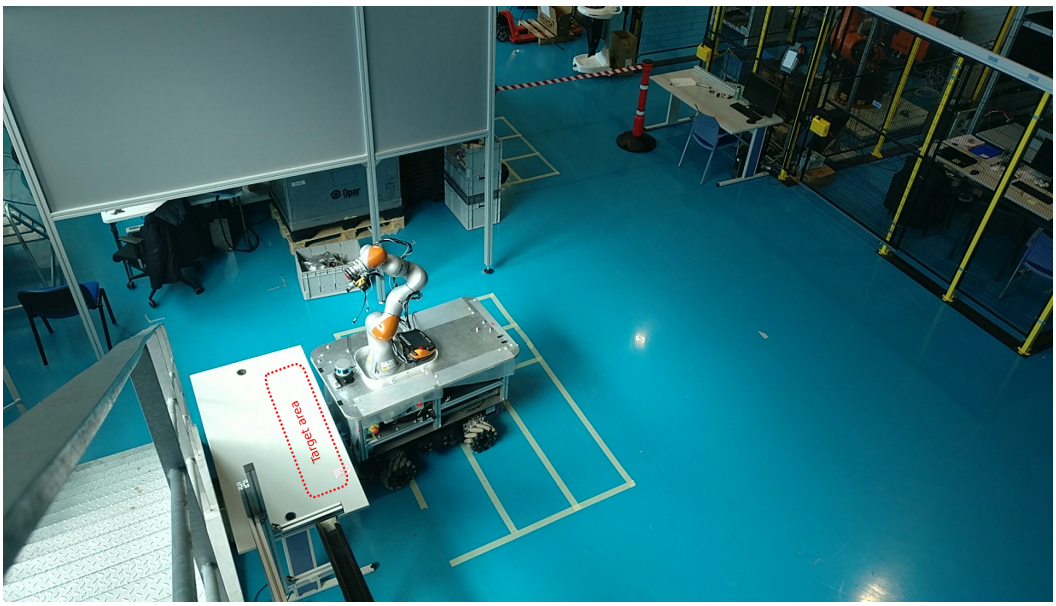


Fig. 3.15.: Real validation scenario.

Trial	Δd (m)	$\Delta\alpha$ (rad)	$coll.$	$amcl\_success$	$real\_success$
1	0.051	0.088	False	True	True
2	0.092	0.060	True	False	False
3	0.021	0.091	False	True	True
4	0.048	0.072	True	False	False
5	0.187	0.178	True	False	False
6	0.018	0.082	False	True	True
7	0.047	0.065	False	True	True
8	0.003	0.125	False	True	True
9	0.032	0.260	False	True	True
10	0.083	0.059	False	True	True
11	0.034	0.061	False	True	True
12	0.056	0.057	False	True	True
13	0.037	0.047	False	True	True
14	0.019	0.040	False	True	True
15	0.050	0.066	True	False	False
16	0.040	0.081	False	True	True
17	0.088	0.062	False	True	True
18	0.052	0.090	False	True	True
19	0.087	0.097	True	False	False
20	0.053	0.078	False	True	True
$Avg.$	0.055 ± 0.039	0.084 ± 0.05	25%	75%	75%

Tab. 3.5.: Measurements obtained in the real environment.

robot-target distances and rotations at time-step $t = T - 1$, $0.055\ m$ and $0.084\ rad$ respectively, are not small enough to ensure a successful manipulation.

Therefore, to assess whether the controller was able to drive the robot to successful picking areas, in each test a new planning was done relying on the manually measured real localisation of the robot ($real\_success$). In that case, the success rate was also 75%, which meant that whenever the arm succeeded planning using the AMCL localisation, it also succeeded with the real localisation. The main decrease in the performance was caused by small brushes with the table where the target was located on it.

3.4 Innovation

Given the success that DRL has had in learning robotic behaviours in other fields, we sought the feasibility of learning a mobile manipulation behaviour without explicit programming. At that time the application of DRL algorithms to learn mobile manipulation operations was a totally unexplored world. The research carried out focused on the operation of picking an object on a table with a AMMR. When performing any picking operation, the positioning of the mobile manipulator base is of vital importance, as the reach of the robotic arms is limited. That said, we proposed to use a DRL algorithm to model the controller of the base of the mobile manipulator, which was optimised in such a way that it took into account the constraints of the arm to learn to position itself in areas that would ensure that the object to be picked was within the arm's reach.

Specifically, our approach was based on learning a speed controller for the base, which, knowing the location of the object to be picked up, it was able to drive the robot's base to areas that allowed the arm to reach the object to be picked up. In RL based solutions, the reward function guides the learning process by rewarding or penalising the actions performed by the robot. Thus, the intuition behind our approach was to reward the robot controller for reaching areas that enabled the arm to pick up the object. In that way, the idea was to incentivise the robot to navigate to areas where the object was more likely to be within the arm's reach. To do this, following a typical RL scheme, the robot was given a number of discrete time steps to navigate to a point close to the target. After executing the navigation, the arm's planner made an attempt to plan a trajectory to the target. If the result was successful, it indicated that the object was within the arm's range, and in this case the reward that the robot received was the maximum possible. If the arm's planning fails, the controller was penalised.

Naturally, the training of the controller was carried out in simulation due to the large amount of experience needed to successfully learn robotic behaviours and to safeguard the integrity of both the environment and the robot. In a first attempt, DDPG and PPO2 algorithms were used to model the base controller, both based on the Actor-Critic architecture. The training of the controllers required:

- The development of the simulated environment.
- A correct definition of the reward function, which would appropriately reward each action performed by the robot.
- The definition of the neural architecture of the algorithms, together with their hyperparameters.
- The definition of the state/action spaces.

In this first approach, as well, the validation was carried out in the simulated environment. Although the behaviour of the controller was somewhat unstable during the training process, the success rates obtained at the end of the training were sufficient to demonstrate the validity of the approach.

Considering the promising results obtained in the first approach, a second step aimed at deploying and validating the technology in a real robot-environment system. At a later date to our first contribution, some publications attempted to apply DRL to perform whole-body control of mobile manipulators. These works were rather different to out approach and, thus, we chose to continue developing our system, relying on DRL to learn only the base controller and using more traditional algorithms to conduct the arm's path planning and control.

Although the behaviour to be learned and the methodology to be followed were the same, in this second approach multiple improvements were introduced in all aspects, from the simulation to the DRL algorithms. On the one hand, when the simulated environment was implemented, new features were taken into account, with the aim of making the simulated environment more realistic. For example, in the simulation of the first approximation, the accelerations of the mobile manipulator were not simulated, which meant that it was very far from reality. On the other hand, the improvements introduced in the simulation were also accompanied by improvements in the reward function, the state/action spaces, and the algorithm that modelled the base controller of the mobile manipulator. Specifically, the TD3 algorithm was

used, which is an evolution of the previously used DDPG algorithm and introduces several new features that improve learning stability. To demonstrate that the changes introduced contributed to improve the learned behaviour, a comparison was made between TD3 and DDPG/PPO2 algorithms used in the first approach, from the point of view of performance and reliability. Finally, once the simulation validation was completed, both the developed software architecture and the trained model were deployed and validated on the real robot.

3.5 Conclusions and future work

DRL is a technology that allows complex robotic behaviours to be learned without the need for explicit programming. According to the results obtained, and in response to our first research question RQ1, it is possible to use this technology to model a picking behaviour with a mobile manipulator. More specifically, in the work carried out, it is shown that a controller can be learnt for the base that by sending velocity commands is able to drive it to zones that ensure the reachability of the arm to the target.

The first proof of concept was carried out in a simplistic simulation, with the sole objective of demonstrating that the proposed approach was valid. Although the results obtained demonstrated the validity of the approach, the main conclusion drawn from the work carried out was the fragility shown by the algorithms with respect to the hyperparameters of the DRL model, the selected neural architecture and the reward function. This was clearly seen in the success rates obtained during learning, that fluctuated considerably indicating that the learned behaviour was not stable. Another important limitation of the technology is the cost of acquiring the necessary experience to train these algorithms. Although this process as usual was carried out in simulation, experience acquisition remains a bottleneck. Despite a plugin being used to speed up the simulation time, the duration of each training was still several hours.

In the second approach, several updates were introduced in both the simulation and the learning algorithm. The idea was to bring the simulation closer to the real environment to facilitate deployment and to adjust the algorithm to achieve more stable learning process. The results obtained in the simulation benchmark, where we compared the TD3 algorithm with those previously used (DDPG and PPO2),

showed that the changes introduced lead to a substantial improvement in both the stability of the training and the overall success rate. In addition to improving the performance of the robot, we proved that the changes introduced increased the reliability of the system.

Finally, the validation on the real robot of the behaviour learnt in simulation served to answer the research question RQ2. The success rate obtained, although significantly lower than in simulation, served to demonstrate the validity of the behaviour in the real system and thus the feasibility of the approach in general. However, several factors came to light that could be the cause of the decrease in the success rate and these are as follows:

- Simulation to reality gap: Despite the efforts made to correctly simulate the physical properties of the robot, the gap between simulation and reality was still very large. Therefore, each action predicted by the controller had a different effect in the simulation and in reality. However, since the actions predicted by the controller were based on real observations, and due to their reactive nature, this effect was considerably diminished.

- Noisy localisation estimation: State of the art localisation systems are not 100% accurate. However, localisation played a very important role as the DRL controller made its decisions largely based on this information. Although the error made by the navigation system was not very large, it was critical especially when the robot navigates very close to obstacles.

- Learning to stop: Although the learned controller successfully stopped the robot when it got close to the target in simulation, this was not the case in reality. Indeed, the controller learnt to send near to 0 velocity commands to stop the robot in simulation, but the real robot struggled to stop and oscillated. This happened due to the noisy localisation estimates, that made the controller believe that the robot was slightly moving when it was actually still. Therefore, the controller tried to correct this error sending opposite velocity commands, which caused the robot to oscillate. This effect was aggravated by the sim-to-real gap. Although we added noise to the localisation to simulate this effect while training, the robot failed to properly stop and this oscillation near the goal sometimes caused the robot to brush the table.

To overcome the limitations mentioned above, as a first step, the manipulator must have an on-board vision system to be able to accurately estimate the relative pose

between the arm's end effector and the target before executing the grasp. In addition, the main future work lines will be focused on adding more sensing capabilities to the agent to increase the safety of the navigation, and on reducing the simulation-to-reality gap. On the one hand, the use of sensors such as lasers, cameras etc. will let the agent sense the dynamic elements in the environment to safely navigate to the target. On the other hand, domain randomisation techniques will let us reduce the simulation-to-reality gap. Due to the difficulty in properly simulating the physical properties of both the robot and the environment, domain randomisation techniques suggest randomising these physical properties in simulation, assuming that the real-world properties are a particular case of the randomised variables.

3.6 Publications

This line of research produced the following manuscripts, the former already published and the latter still under consideration:

- Ander Iriondo et al. "Pick and place operations in logistics using a mobile manipulator controlled with deep reinforcement learning". In: *Applied Sciences* 9.2 (2019), p. 348

- Ander Iriondo et al. "Learning positioning policies for mobile manipulation operations with deep reinforcement learning". In: *International Journal of Machine Learning and Cybernetics* (2023)

Both articles are included in the PART II of the work.

4

Affordance-based grasping point detection in point clouds with graph convolutional networks

Grasping point detection (GPD) has attracted the attention of researchers since early days of robot manipulation. The detection of the gripping points consists of finding the best areas (position and orientation) within the objects to be gripped conditioned by the tool being used. The complexity of estimating the grasping points lies mainly in the arrangement and typology of the objects.

Regarding the organisation of the objects, the following casuistry is defined:

- Structured arrangement: The objects to be grasped are fully organised inside the box.
- Semi-structured arrangement: There is a predictability in the way the objects are arranged and the objects are relatively well separated.
- Random arrangement: The products are arranged randomly in the box and do not follow any pattern.

Therefore, the less structured the scene, the higher the difficulty to detect grasping points. In this work we focus on the bin-picking problem, which is very common in industrial environments, warehouses and logistics centres. Specifically, we address the problem of random bin-picking, where objects are randomly arranged inside a box. From the point of view of object typology, random bin-picking can be divided into the following categories:

- Mono-reference: Objects are randomly placed inside the box and belong to the same reference which is generally known.

- Multi-reference: Objects are randomly placed but belong to different categories, there might even be unknown objects. The number of references is determined by the application but can be very high.

GPD has traditionally been approached as a computer vision detection problem. In the past, hand-designed features such as SIFT [76], SURF [77] or ORG [78] were used to first recognise the object and then perform model-based grasp planning. However, while such methods are robust with small subsets of known objects, they lack generalisability and are time-consuming. Unlike other traditional computer vision algorithms, these detection methods are used online and, to meet cycle time constraints, computational speed is required. In dynamic and unstructured environments such as warehouses or distribution centres, a large number of items need to be handled, and thus, it is unfeasible to configure systems for each reference and more flexible solutions are needed. The work presented in this chapter focuses precisely on flexible part handling in industrial environments.

4.1 State of the art

According to Sahbani et al., GPD applications are divided into analytic and data-driven [79]. In mono-reference applications, the objects are generally known and their 3D model can be easily extracted. There, 3D model matching analytic algorithms such as ICP [80] offer a robust solution to estimate the pose of the objects in the scene [81]. Thus, the grasping points are predefined in the 3D model of the object, and once the object is located in the scene, these predefined points are used to grasp the object. In multi-reference applications, by contrast, these ad-hoc solutions are not practical since the number of references to be manipulated can be huge, which would imply the system to be adjusted specifically for each of the references. In addition, in environments such as big warehouses, the set of objects to be handled may change from time to time and this would require a manual readjustment of the system. For this reason, detection systems must be flexible, in the sense that they must be able to (1) handle objects with enormous variability in shape, colour, texture, etc. and (2) generalise to previously unseen situations and thus avoid manual readjustments.

Data-driven methods have demonstrated to overcome the mathematical complexity of analytic grasping methods and have demonstrated an increased flexibility [82].

These methods focus on learning representations from previously acquired grasping experience, and the generated models can be used to estimate grasping points in new queries. Specifically, DL-based methods have recently gained popularity due to their demonstrated ability to learn from large amounts of data and generalise to new situations. Those methods have shown state-of-the-art performance on a wide range of computer vision, audio and natural language processing problems and have also been successfully applied to detect grasping points [83]. A taxonomy of DL-based GPD methods is shown in Figure 4.1.

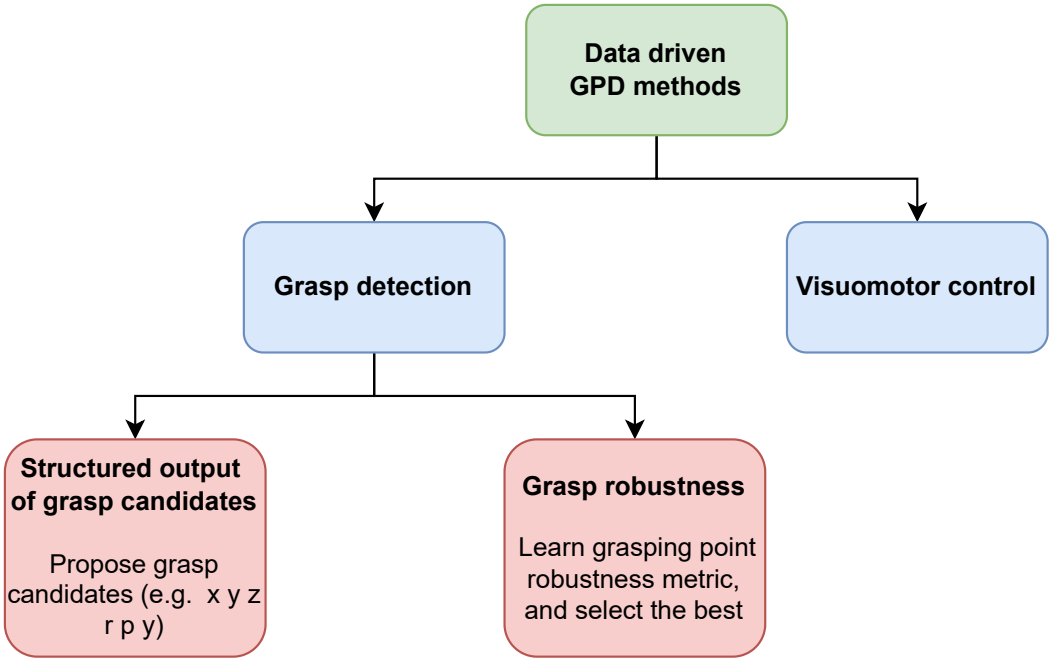


Fig. 4.1.: Taxonomy of DL-based GPD methods.

Early DL-based approaches for GPD such as [84], first identify the objects in the scene and then heuristically estimate the candidate grasping points using the 3D data of the item. Identification-based methods have proven to be useful and efficient with small subsets of objects, but struggle when the number of references is high, not being able to generalise to never seen parts. Instead, affordance-based methods have outstanding capabilities of predicting affordable areas in items just focusing on the shape, colour and texture of the objects. The fact that these methods do not depend on object identification makes them highly generalisable to new objects, meaning that they offer a very flexible grasping solution.

Preliminary affordance-based GPD methods such as [85, 86, 87, 88], only focused on gripper end-effectors and used convolutional neural networks (CNN) to map 2D RGB-D images with graspable regions. In these works each grasping point candidate

was detected as a rectangle over the RGB-D image, indicating the grasping position, orientation and opening of the fingers of the gripper. The main disadvantage of this kind of methods is that they work in a sliding window manner, meaning that an inference is made per each object in the scene. This considerably increases the inference time when the number of objects is high. In addition, sliding window approaches fail predicting grasping points in cluttered scenes, since they loose the context of the scene due to the fact that the inference is made over a small portion of the entire image. However, the context of the scene provides relevant information for example in a bin-picking application, since occlusions, entanglements etc. between objects can be identified. In fact, this global context information can be used to decide which is the most suitable object to pick up.

To overcome the aforementioned limitations of the sliding window methods, alternate methods try to estimate grasping candidates by predicting pixel-wise affordances. In an incipient approach, Detry et al. proposed to learn grasping affordances as continuous density functions [89]. In fact, they used real grasping experience to learn the affordance-based models. Unfortunately, the method was designed to work with separate objects. More recently, one-shot approaches based on fully convolutional networks (FCN) [90] have gained importance, as they are able to detect grasping candidates with their respective affordance scores at a pixel level in a single inference. In a key work, Zeng et al. propose an affordance-based GPD system based on FCNs to detect the best grasping areas for a multi-functional gripper composed of a suction cup and a parallel jaw gripper [91]. The fact that it does not learn the identities of the objects, but looks at the shape, colour and texture of the objects, makes the system capable of predicting grasping zones also on unseen objects.

Instead of directly predicting grasping points, another widely used approach is to learn a robustness metric on candidate grasping points. These works define robustness as the probability that a candidate grasping point is satisfactory. On the one hand, the general approach is to use analytical grasping point sampling algorithms (e.g. antipodal, approach-based etc.) [92] to obtain a large set of candidate grasping points. On the other hand, they use DL to learn a model that assigns a robustness score to each candidate grasping point. After assigning a quality score to each candidate, the one with the highest probability of success is selected. For example, Pas et al. use an approach-based grasping point sampling algorithm and learn a binary classifier to discriminate feasible candidates from invalid ones for a two-fingered gripper [93]. Another well-known work is [94] where they first use an antipodal candidate sampling algorithm, and then use DL to learn a continuous

robustness metric between [0-1]. In fact, Mahler et al. use a grasp quality CNN (GQ-CNN) to learn to assign a robustness score to each candidate, also for a two-fingered gripper.

Another completely different approach is to learn visuomotor control policies to directly predict the robot's actions to pick up objects instead of explicitly predicting grasping points. Thus, these methods avoid the need for an additional robot controller. For example, Mahler and Goldberg were able to learn a deep policy for picking up objects in cluttered scenes in [95]. In [96] the authors trained a large CNN to predict the probability of success of movements in the gripper task space, only using RGB images. The learning process was carried out with 14 real robotic manipulators accumulating experience during two months of continuous execution. However, the cost of gaining experience in reality makes the solution hardly transferable to industry. Similar to the procedure proposed by Levine et al., in [38] Kalashnikov et al. used RL to learn high-precision control policies for picking up objects in cluttered scenes, also using RGB images. This approach also used the experience of multiple real robots to optimise the policy neural network. To avoid the cost of real setups with multiple robots and to decrease the experience acquisition time, in [97] James et al. proposed to learn visuomotor control policies in simulation. To reduce the gap between simulation and reality, the training process is usually performed with domain randomisation [98]. Such methods increase the complexity of the problem as they directly learn to control the robot arm, and are not practical for industry due to the huge amount of data needed to learn such picking behaviours.

Previously mentioned methods that rely on DL use architectures that are designed to work only with 2D images, and therefore are not able to extract 3D features from non-euclidean data (e.g. point clouds). However, the use of these 3D features plays a key role on predicting grasping points, particularly in affordance-based models where the shape and the spatial arrangement of the items provide a lot of information. Recently, graph convolutional networks (GCN) [99] have shown to be capable of extracting features on non-Euclidean data organised as graphs, such as meshes or point clouds [100]. The most popular architectures are PointNet [101] and its evolution PointNet++ [102], that have been applied to detect grasping points among many other applications [100].

GPD methods based on GCNs can be divided into the following groups:

1. **Sampling-based methods** usually rely on 3D models of the objects to generate training data in simulation. First, grasping point candidates are sampled using criteria such as the grasping angle or force [92], in random scenes generated using the 3D models of the items and the robot's gripper. Then, a quality score is analytically calculated for each candidate and is used as a label. Finally, the grasping point dataset generated is used to train a GCN-based model that assigns a quality score to each sampled grasping point in cluttered scenes where 3D models are not available. For instance in [103] authors used an antipodal grasp sampling algorithm to generate the grasping point candidates for a gripper end-effector, and trained PointNet to infer their quality. As shown by Mousavian et al. variational autoencoders (VAE) can also be used to sample grasping points in point clouds [104]. In that approach, the evaluation of each generated candidate was done in a second phase using a PointNet-based architecture. The main weaknesses of the methods that divide the detection problem in grasp sampling and grasp evaluation phases are: (1) Both time and computational cost are dependent on the number of grasping points sampled, and (2) assume the existence of 3D models of the objects to generate training data for the GCNs that are later used to evaluate each sampled grasping point.

2. **Single-shot methods** take the raw single-view point cloud of cluttered scenes as input and directly train a GCN to generate candidate grasping points alongside a quality score. Thus, the computational cost is constant independently of the number of generated grasping points. For instance in [105] PointNet++ network was used to implement a single-shot grasp proposal network. In that work the dataset used for training was automatically generated in simulation relying on 3D models of the items. In addition, the training was done with single objects and therefore global factors of the scene such as occlusions or entanglements were not considered, but are vital to properly define grasping points. Also in [106] authors implemented a single-shot grasp proposal network based on PointNet++, that directly inferred 6-DoF grasping poses and their quality over single-view point clouds. First, the grasping point candidates were analytically sampled in simulation using the 3D models of single items and without considering the global arrangement. Then, considering the synthetically generated cluttered scenes, the unfeasible grasps were discarded. Finally, single-view point clouds of the scene were acquired and annotated using the sampled grasping points. Although both in training and testing the model worked with single-view point clouds, the 3D models needed to generate the training dataset can be expensive to obtain in setups where the number of references is high.

4.2 Goal

Our proposal focuses on the use case of multi-reference random bin-picking, which consists of emptying a box full of objects that belong to different totally disordered references. This use case is common in the management of return orders in warehouses. The uncertainty introduced by the variability in the type of objects and their random position causes traditional grasping point detection methods to fail, and makes the need for more flexible methods evident.

Although there are methods that use RGB-D images, it is not clear to what extent these take advantage of the 3D information provided by the depth image, as CNNs are designed only to extract 2D features.

Alternatively, algorithms based on GCNs have been applied to predict grasping points directly on n-dimensional point clouds, thus solving limitations of 2D solutions thanks to their ability to extract higher dimensional spatial features. Existing applications focus on detecting grasping points in relatively simple scenes where objects are placed on a table. However, such techniques have not been applied to random bin-picking problems, in which the degree of clutter is much higher, making the operation of picking objects in a box difficult. Moreover, existing work in the literature only focuses on detecting grasping points for a two-finger gripper.

Seeing the promising results offered by GCNs in learning from point clouds, our intuition was that using n-dimensional point clouds and tackling it as a segmentation problem would improve the performance of affordance-based grasping point detection compared to the RGB-D image-based models.

Thus, with the goal of facing the TO3: "Robust and efficient bin-picking" of the PICKPLACE project, this was the research question we asked ourselves:

- **RQ3:** Does the use of n-dimensional point clouds help to predict the affordance of objects in a random bin-picking problem for both suction and two-finger end effectors?

4.3 Developed approach

Zeng et al. [91] developed a FCN-based model to get pixel-wise affordance scores from 2D RGB-D images in a multi-reference bin-picking context, the setup we are interested in. Thereby, our proposal aims to extend this work to be able to handle n-dimensional data to boost the performance of GPD.

We selected the *Deep GCN* [107] architecture originally thought for point cloud scene segmentation, and adapted it to obtain object affordance scores in multi-reference bin-picking scenes. For training and evaluation purposes, we created a dataset with multiple annotated RGB-D bin-picking scenes. The dataset was annotated once and we used it to train and test both methods (FCNs and GCNs). Although the training of the FCN was straightforward, to train the GCN the RGB-D data needed to be transformed into point cloud. To that end, we developed a bin-picking oriented pipeline, which made the learned GCNs applicable to other picking scenarios. Finally, we compared both the FCN and the GCN based methods from the perspective of performance and generalisability, using the metrics proposed in [91].

4.3.1 Theoretical background

4.3.1.1. Point cloud segmentation with deep learning

3D data can be represented in different formats such as depth images, meshes, point clouds and volumetric grids. Point clouds are sets of 3D data points usually represented by three coordinates in a Cartesian or other coordinate system [108]. These unstructured 3D points usually represent the geometry of 3D objects and are used in many scene understanding tasks such as classification, object detection and semantic segmentation in 3D scenes [100]. In fact, it is one of the most used format as it preserves the raw geometric information in 3D space without any discretisation.

The taxonomy of DL-based point cloud segmentation algorithms proposed by Guo et al. in [109] can be seen in Figure 4.2.

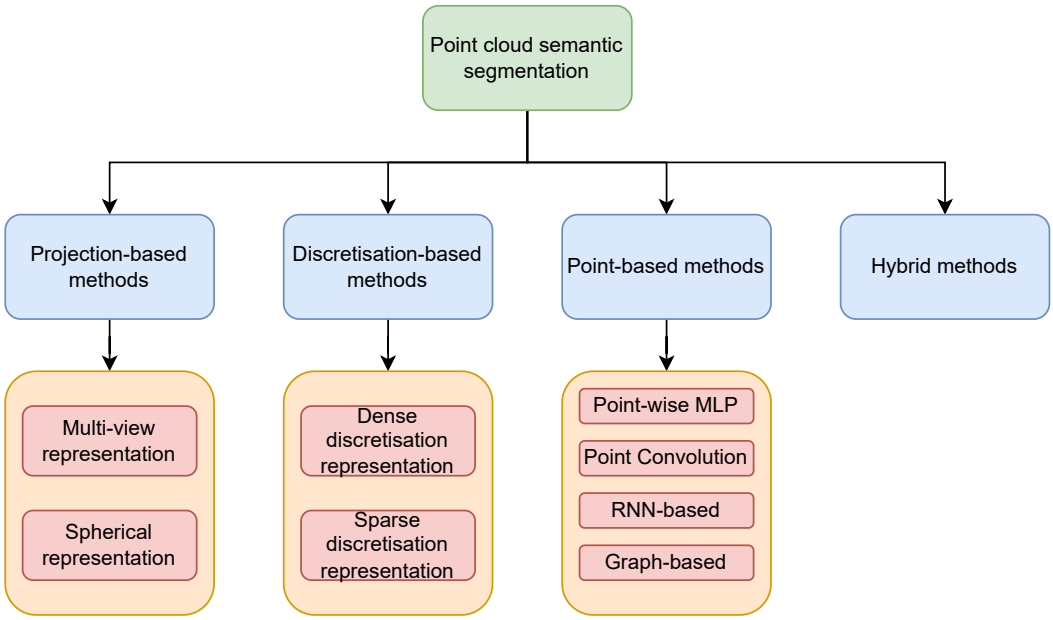


Fig. 4.2.: Taxonomy of point cloud semantic segmentation methods.

According to Guo et al. the most used point cloud semantic segmentation approaches are the following:

1. Projection-based methods: These methods convert 3D point clouds into 2D images (i.e. depth images), including multi-view and spherical images, and use 2D CNNs such as FCNs to perform the segmentation.

2. Discretisation-based methods: These methods convert the point cloud data into a dense/sparse discrete representation such as voxels. Doing that, 3D CNNs can be applied to learn a voxel-level segmentation.

3. Point-based methods: These methods directly work on raw irregular point clouds. The fact that point clouds are unstructured and orderless makes standard 2D/3D CNNs not applicable.

4. Hybrid methods: These methods aim to learn multi-modal features from 3D scans by combining multiple data sources such as point clouds and RGB images. Hybrid methods usually use either discretisation-based or point-based methods to extract geometric features from the 3D data, later to fuse, for instance, with other 2D CNN streams.

GCNs belong to the point-based family as they directly learn about unordered and unstructured point clouds. Specifically, they fall into the graph-based subfamily, as they internally represent the point cloud as a graph structure, with the aim of extracting the underlying 3D geometric shapes and structures.

4.3.1.2. Graph convolutional networks

Graphs are popular data structures that are used in many applications such as social networks, natural language processing, biology or computer vision [110]. A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined by an unordered set of vertices $\mathcal{V}$ and a set of edges $\mathcal{E}$ indicating the connections between vertices. Vertices are represented associating each vertex $v \in \mathcal{V}$ with a feature vector $h_v \in \mathbb{R}^D$, where D indicates the dimension of the feature vectors. So, the graph is represented concatenating the feature vectors of the set of unordered vertices $h_\mathcal{G} = [h_{v_1}, h_{v_2}, h_{v_3}, ..., h_{v_N}] \in \mathbb{R}^{DxN}$, where N indicates the number of vertices in the graph.

In GCNs, the most used operations at each layer are *aggregation* and *update*. These operations receive the input graph $\mathcal{G}_l = (\mathcal{V}_l, \mathcal{E}_l)$ and they output the graph $\mathcal{G}_{l+1} = (\mathcal{V}_{l+1}, \mathcal{E}_{l+1})$ at the l-th layer. On the one hand, the aggregation function is used to collect the information of the neighbour vertices (e.g. max-pooling aggregator). On the other hand, the update function applies a transformation to the collected data by the aggregate operation, in order to generate new vertex representations (e.g. using MLPs). At each layer, the aforementioned operations are applied to each vertex $v \in \mathcal{V}_l$ and new vertex representations are generated [111]. A general graph convolution operation $\mathcal{F}$ at l-th layer is shown in Equation 4.1, where $\mathcal{W}_l^{agg}$ and $\mathcal{W}_l^{update}$ are learnable weights for aggregation and update functions respectively.

$$\mathcal{G}_{l+1} = \mathcal{F}(\mathcal{G}_l, \mathcal{W}_l) = Update(Aggregate(\mathcal{G}_l, \mathcal{W}_l^{agg}), \mathcal{W}_l^{update}) \tag{4.1}$$

Most GCNs have fixed graph structures and only per-vertex features are updated at each iteration. However, more recent works demonstrate that changing graph structures contribute to better learn graph representations. For instance, Wang et al. proposed a neural network module dubbed *EdgeConv* [112] that finds k nearest neighbours in the feature space to reconstruct the graph at each EdgeConv layer. The authors claim that this architecture is suitable for classification and segmentation tasks in point clouds.

Furthermore, as reported in recent work, traditional GCNs cannot go as deep as CNNs due to the high complexity in back-propagation and they are no more than 3 layers deep [99, 113]. State-of-the-art suggests multiple changes in GCN architectures to overcome the aforementioned shortcomings [114, 107, 115]. The *Deep GCN* model architecture developed by Li et al. in [107] proposes multiple changes that allow to effectively learn deep GCNs that are up to 56 layers deep. Moreover, this architecture has shown great performance in semantic segmentation tasks in the S3DIS indoor 3D point cloud dataset [116].

The adaptations that allow deep GCNs are twofold:

- **Dilated aggregation:** The convolution operation that is applied after each GCN layer relies on the *Dilated* k-NN to find dilated neighbours (i.e. dilated convolution), getting as a result a *Dilated Graph*. The idea is to increase the receptive field of the network (see Figure 4.3). Having a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with a *Dilated* k-NN and d dilation rate, the *Dilated* k-NN returns the $k \times d$ nearest neighbours in the feature space, ignoring every d neighbours. The l_2 distance is used in the feature space.

- **Residual and Dense connections:** Based on the success of models with residual connections between layers such as ResNet [117], or dense connections such as DenseNet [118], these concepts have been translated to GCNs, allowing much deeper models (see Figure 4.4).

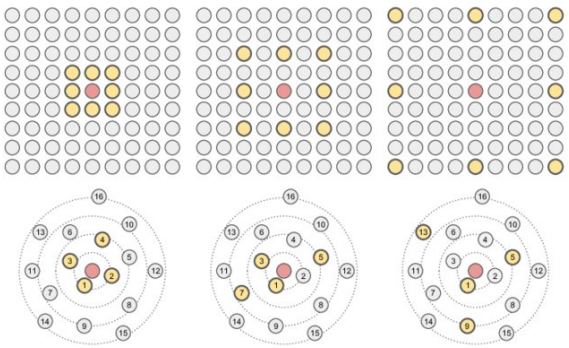


Fig. 4.3.: Dilated convolution on a structured graph arranged in a 2D grid and using a 3×3 kernel (top), and dynamic graph convolution in an irregular graph, e.g. 3D point cloud, (bottom). From left to right, dilation rates $d = 1$, $d = 2$ and $d = 4$ are used [107].

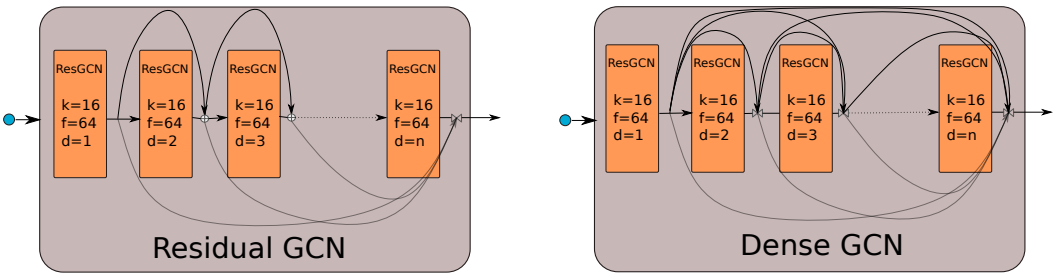


Fig. 4.4.: Residual and dense GCNs, where d refers to the dilation rate, k to the number of neighbours, and f to the number of filters. Note that in the residual GCN a vertex-wise addition is used, while in the dense GCN vertex-wise concatenation is used.

4.3.2 Robotic environment

In the context of the *PICKPLACE* European project, a multi-functional gripper was developed which is depicted in Figure 4.5. This multi-functional gripper was composed of the following retractable end effectors: A magnet, a suction cup and a two finger gripper. All of them were equipped with tactile sensors that were designed to softly handle any kind of product.

Fig. 4.5.: Multi-functional gripper attached to the UR robot.

Although in this work the gripper was not physically used, it was important to define its particularities since the annotation of the dataset to be used to learn the object affordances highly depended on these specifications. For instance, the quality of a grasping point can drastically change depending on the maximum width between the fingers of the gripper. In this work, both, suction and gripper end effectors were considered.

4.3.3 Affordance grasping dataset for suction and gripper

Although automatic dataset generation/annotation methods are widely used in simulation, we opted to capture real-world bin-picking scenes to generate the dataset, thus avoiding the sim-to-real gap. During the annotation process, we believed that the experience and the criteria of the human is vital to analyse the particular casuistry of each scene.

We used a *Photoneo Phoxi M* camera to generate our dataset, which has been widely used in many bin-picking applications and has demonstrated a great performance [119, 120]. We positioned the camera overhead the bin to reduce occlusions. To generate the dataset, we used a set of 37 rigid and semi-rigid opaque parts that were selected in the context of the *PICKPLACE* EU project. In spite of the fact that we included some transparent and shiny objects, we opted to focus our analysis on opaque parts. As a result, we collected a dataset composed of 540 scenes, and the following elements were stored per each scene:

- Intrinsic parameters of the camera.
- RGB-D images of the scene. The grey scale images obtained by the *Photoneo* were cloned to obtain a three-channel colour image. Using the intrinsic parameters, images can be easily transformed to a single-view point cloud.
- Bin localisation with respect to the camera.
- Point cloud of the scene.

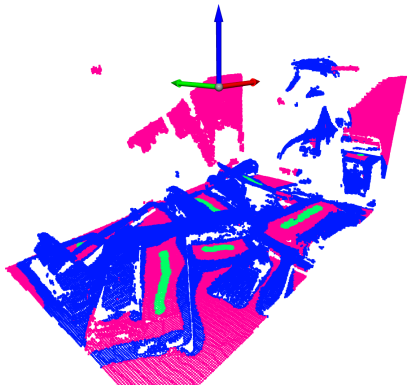
As RGB-D images can be easily converted to single-view point clouds, the annotation process was carried out only in RGB-D data.

4.3.3.1. Suction

For the suction end effector, a grasping point is defined by a 3D contact point in the object to be picked, along with its normal vector. The normal vector determines the orientation of the suction end effector. A sample annotated RGB-D scene for the suction is depicted in Figure 4.6, together with its transformation to point cloud. During the annotation we followed the following criteria: Pixels located on flat surfaces and near the centre of the object were considered as good. All other points belonging to the objects were considered as bad. Pixels that do not belong to the objects were considered neutral. All this process was done taking into account the weight distribution of each object, occlusions and that some objects could be stepped on/clipped with others. In these cases the criterion was adjusted to each situation.

To transform the RGB-D data into a single-view point cloud we followed the processing pipeline shown in Figure 4.7. We aimed to abstract from the camera pose in the scene by always working in the bin coordinate frame. This also gives the possibility to work with multiple extrinsically calibrated cameras.

(a) Suction annotation in RGB-D data. Green and red pixels belong to good and bad grasping areas respectively. Remaining pixels are considered as neutral.



(b) Same scene converted to semantically annotated point cloud. Green, blue and pink points belong to good, bad, and neutral grasping areas respectively.

Fig. 4.6.: Annotation transformation for suction.

The main processing steps that were executed in the pipeline are the following:

1. Transform the RGB-D image into point cloud using the intrinsic parameters of the camera.

2. Localise the bin in the point cloud using a previously developed 3D model matching method [14].

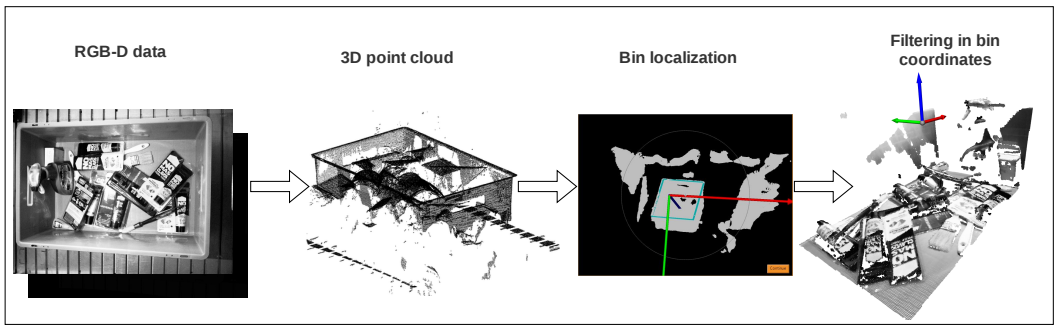


Fig. 4.7.: Data preprocessing pipeline.

3. Transform the point cloud into the bin's coordinate frame and filter the points that lay outside of it.

4. Simplify the point cloud by applying a voxel downsampling method, with a voxel size of 2mm.

4.3.3.2. Gripper

A gripper's grasping point is defined by:

1. A 3D point in the space that indicates the position where the centre between the fingers of the gripper has to be placed.

2. The orientation of the gripper in the vertical axis.

3. The opening of the fingers.

To generate the gripper data, the original RGB-D scenes were transformed into orthographic heightmaps, and top views of the scenes were obtained, due to the fact that only vertical gripper actions were taken into account. For that purpose, the following steps were carried out:

1. Transform the RGB-D image into point cloud using the intrinsic parameters of the camera.

2. Localise the bin in the point cloud using the same method than for the suction.

3. Transform the point cloud into the bin's coordinate frame and discard the points that fall outside.

4. Orthographically project the resulting point cloud and store it again as RGB-D.

An example of an orthographically projected scene is shown in Figure 4.8.

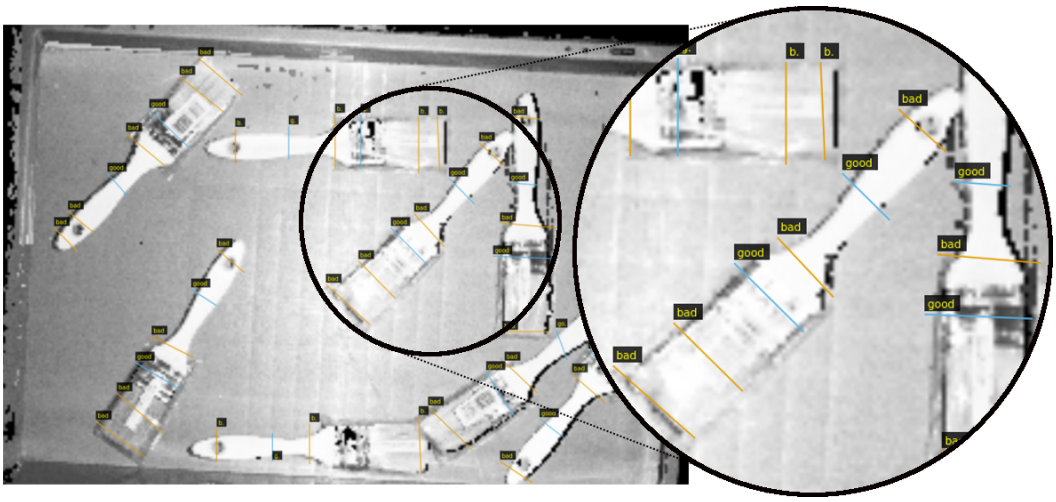


Fig. 4.8.: Gripper annotations in a RGB-D orthographic heightmap.

The labelling process for the gripper was first done using straight lines and then converted to pixel-wise labels. Following that approach, only hard negative grasping points were annotated as bad. The centre of the line indicates the 3D point in the space where the gripper should move to, and the orientation of the line with respect to the horizontal axis of the image indicates the grasping angle. The opening of the gripper fingers is computed on-line during execution, taking into account the local geometry of the grasping area.

Rotations in z axis were only considered, to perform vertical grasps. For the sake of simplifying the problem, the z axis was divided into n fixed angles, in our case $n = 16$. To decide to which discrete angle each annotated line belongs to, the angle θ between each line and the horizontal axis of the heightmap was computed. Thus, for each annotated RGB-D scene for the gripper, 16 pixel-wise annotations were obtained (one mask per angle). The transformation from line annotations into masks was done in the following way:

1. The θ angle between each line and the horizontal (y) axis indicates the discrete n orientation in z axis that the annotation belongs to.

2. The scene RGB-D image is rotated n times with an increment of $360°/n$ with respect to the z axis of the bin.

3. The centre pixel of each annotated line is computed and included in the corresponding annotation mask among the n possible orientations.

To convert the RGB-D data for the gripper into point cloud, the followed procedure was opposite to the one used for the suction. In this case the annotated RGB-D orthographic heightmap was already transformed into the bin coordinate frame and, thus, the conversion of RGB-D images to 3D point clouds in the bin coordinate frame was straightforward. For the suction effector a single labelled point cloud was obtained from each annotated RGB-D image. However, from each annotated scene for the gripper, n labelled point clouds were obtained, one per each rotation angle in the z axis.

The transformation from 2D annotations into 3D was made in the following way:

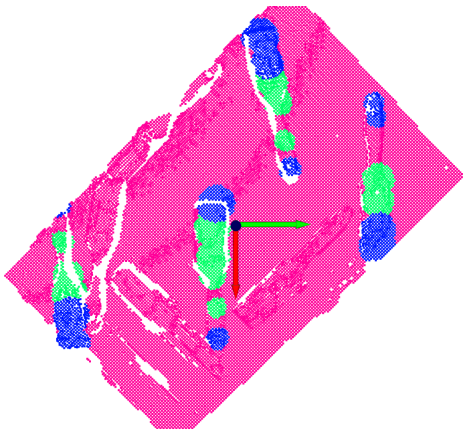
1. The angle θ between each line and the y axis (horizontal axis of the orthographic heightmap) indicates to which discrete n orientation in the z axis the annotation belongs to.

2. The scene point cloud is rotated n times with an increment of $360°/n$ with respect to the z axis of the bin.

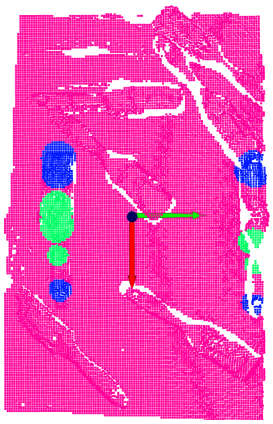
3. The 3D coordinate of the centre pixel of each annotated line is computed in the corresponding point cloud among the n rotated clouds.

4. The length of each line indicates the radius of the circumference around the centre pixel that was annotated.

An example of the gripper annotation conversion to 3D can be seen in Figure 4.9 with two of the 16 generated annotated clouds depicted. Specifically, $n = 6$ and $n = 12$ discrete orientations in the z axis have been selected for illustration purposes. In these examples the y axis (green) indicates the orientation of the gripper.



(a) $n = 6$ annotated 3D point cloud, that corresponds to $\theta = 135°$.



(b) $n = 12$ annotated 3D point cloud, that corresponds to $\theta = 270°$.

Fig. 4.9.: Annotation transformation for the gripper.

4.3.4 Predicting object affordances with GCNs

The GCN-based architecture used to predict object affordances is depicted in Figure 4.10.

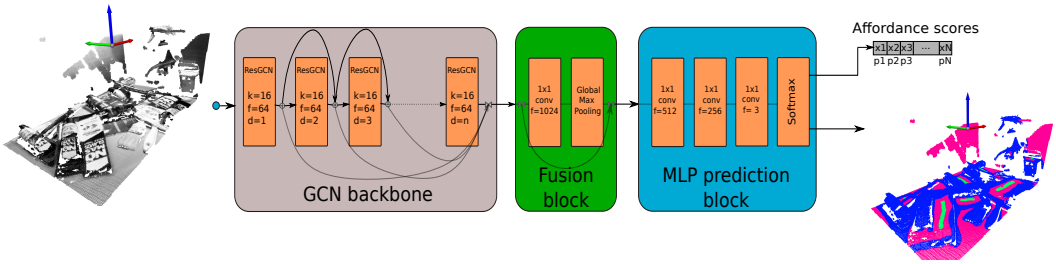


Fig. 4.10.: Used model architecture.

As mentioned before, the selected model was based on the *Deep GCN* architecture proposed in [107]. First, the model gets as input the n-dimensional point cloud data. Then, the GCN backbone is in charge of extracting features from the input data. Afterwards, the fusion block fuses the features extracted by the backbone and global features are extracted. Finally, the prediction block predicts point-wise labels.

With the aim of obtaining affordance scores instead of point-wise labels, we modified this latter block. Therefore, the output tensor of the network has the following shape: $(N_{batch}, N_{pts}, N_{classes})$, where N_{batch} indicates the number of batches fed to the network, N_{pts} denotes the number of points per batch, and $N_{classes}$ the number of classes. In our case $N_{classes} = 3$ (good, bad and neutral grasping points). Originally, to obtain the point-wise labels, the *argmax* operation was performed across the last

axis of the output tensor. To obtain good grasping affordances, however, we only select the channel corresponding to the "good" category, as the *softmax* operation outputs the probability distribution over the 3 classes. In our case the affordances are predicted in the first channel of the last axis, as shown in Equation 4.2.

$$affordances = [N_{batch}, N_{pts}, 0] \tag{4.2}$$

In that way the points are ordered depending on their affordance score. The higher the affordance score, the higher is the likelihood of this grasp being a success. We used the same general model architecture both for suction and gripper end effectors.

Before training the GCN-based models, the 3D dataset was preprocessed in a specific way for the suction and gripper. The details of the preprocessing steps carried out are detailed in Sections 4.3.4.1 and 4.3.4.2.

4.3.4.1. Data preprocessing for suction

Since the number of points that GCNs can analyse is limited due to computational complexity issues and hardware restrictions, the point clouds need to be downsampled. In spite of the fact that the general approach in the literature was to split the input point cloud in partial blocks to sequentially feed the model (using a reduced number of points per block), we decided our model to be a single-shot detector and thus it must process the whole scene in a single inference. Therefore, we increased the number of points to be processed in each batch and set it to 8192. We used a random sampling method to reduce the sampling time compared to other complex sampling methods. For suction, each input point was defined by a 9-dimensional feature vector that can be seen in Equation 4.3. These features are defined in Table 4.1.

$$[p_{bin}, rgb, n] \in \mathbb{R}^9 \tag{4.3}$$

Here our intuition was that the usage of the normal vector information (not available in 2D) would lead to a performance boost, as most suction grasping points are located in flat and regular surfaces. In addition, as the annotated data was limited, we applied data augmentation to generate more training data, by randomly rotating the

The point coordinates with respect to the bin coordinate system	$p_{bin} = [x_{bin}, y_{bin}, z_{bin}]$
RGB values normalised to $[0 - 1]$	$rgb = [r, g, b]$
Normal vector computed using the points within a radius of $0.05m$ before the random sampling	$n = [n_x, n_y, n_z]$

Tab. 4.1.: Features that define a point for the suction.

scene with respect to the bin's vertical axis. This was possible because the suction data had no implicit end-effector rotation information.

4.3.4.2. Data preprocessing for the gripper

Similar to the preprocessing steps followed for the suction, again 8192 points were randomly sampled for each scene. For each annotated heightmap, as many rotated point clouds as n discrete angles were obtained. Thus, n inferences were performed to detect grasping points in a scene for the gripper. Each input point was defined by a 6-dimensional feature vector as shown in Equation 4.4. The features are described in Table 4.2.

$$[p_{bin}, rgb] \in \mathbb{R}^6 \tag{4.4}$$

The point coordinates with respect to the bin coordinate system	$p_{bin} = [x_{bin}, y_{bin}, z_{bin}]$
RGB values normalised to $[0 - 1]$	$rgb = [r, g, b]$

Tab. 4.2.: Features that define a point for the gripper.

Because gripper grasping points usually are in irregular surfaces, our intuition this time was that normal vector information would not contribute to better learn grasping point representations. Thus, we opted to exclude it. Aside from that, in this case it was not possible to augment the data rotating it vertically, as the rotation of the data had implicit information of the grasping orientation.

4.3.5 Evaluation metrics

As authors claim in [91], a GPD method is robust if it is able to consistently find at least one grasp proposal that works. Thus, the metric used to measure the robustness of the methods is the *precision* of the predictions with respect to manual annotations. The precision was computed with respect to multiple confidence levels:

- $Top-1$ *precision:* For each scene, the pixel (for the FCN) and the point (for the GCN) with highest affordance score was taken into account to measure the precision.

- $Top-1\%$ *precision:* In this case the pixels/points were sorted according to their affordance scores and those within the 99th percentile were selected to measure the precision.

In both cases, a grasping proposal was considered as a true positive if it was manually annotated as good grasping area, and as a false positive if was manually annotated as bad grasping area.

4.3.6 Experimentation

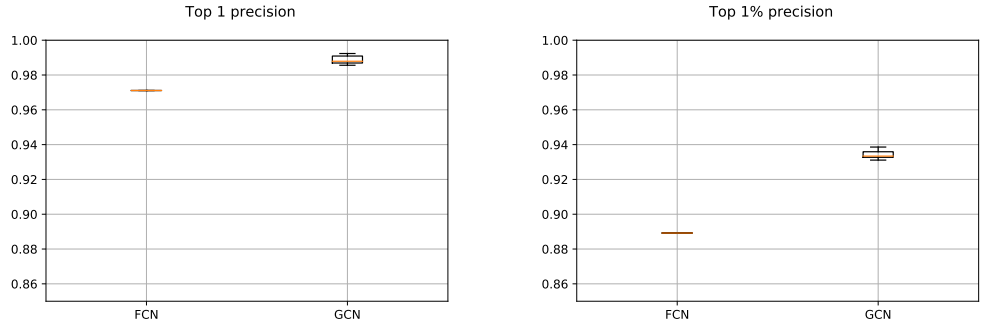
Two tests were carried out to demonstrate that the implemented GCN-based method together with the use of n-dimensional point clouds improves the implementation based on FCNs and 2D images.

4.3.6.1. Known objects

The dataset generated using 37 known objects was used to carry out this test, 80% of the scenes for training and the remaining 20% for validation. Thus, although the training and validation scenes were different, the objects that appeared in them were common.

The precisions obtained concerning the suction model can be seen in Figure 4.11, for the GCN and FCN respectively. As far as the former is concerned, due to the random sampling performed to reduce the dimensionality of the point clouds, 5 inferences

were made to measure the average behaviour. In the case of the FCN, however, as the input data were fixed, only a single inference was made.



(a) Top-1 scores.

(b) Top 1% scores.

Fig. 4.11.: Obtained precision scores per confidence percentiles for the suction models with known objects.

From the results it can be concluded that the GCN had a clearly superior performance compared to the FCN, considering both the Top-1 and Top-1% precision scores. On the one hand, we can say that the use of n-dimensional point clouds together with the introduction of normal vectors helped to improve the accuracy of the predictions compared to the 2D method. In addition, the arbitrary rotation of the n-dimensional scenes served to considerably increase the volume of training data, which also contributed to the improvement of the results. In the case of images, however, although visual augmentations could be applied, performing such spatial augmentations is not trivial.

An example of an inference of the GCN with a scene obtained from the test split is depicted in Figure 4.12. For illustration purposes, Top-1% and Top-5% precisions have been selected.

The results obtained with the gripper model are shown in Figure 4.13. We followed the same methodology as in the case of the suction, with 1 and 5 executions for the FCN and GCN respectively. As it can be seen in Figure 4.13a, the GCN based method outperformed the FCN, giving at least a valid grasping point per each scene with high precision. However, Figure 4.13b shows that the FCN was more precise considering the highest 1% of the predicted affordances. This means that the GCN was more overconfident than the FCN, sometimes assigning high affordance scores to points that did not deserve it.

A visual example of the results obtained for the scene with vertical rotation $n = 4$ can be seen in Figure 4.14.

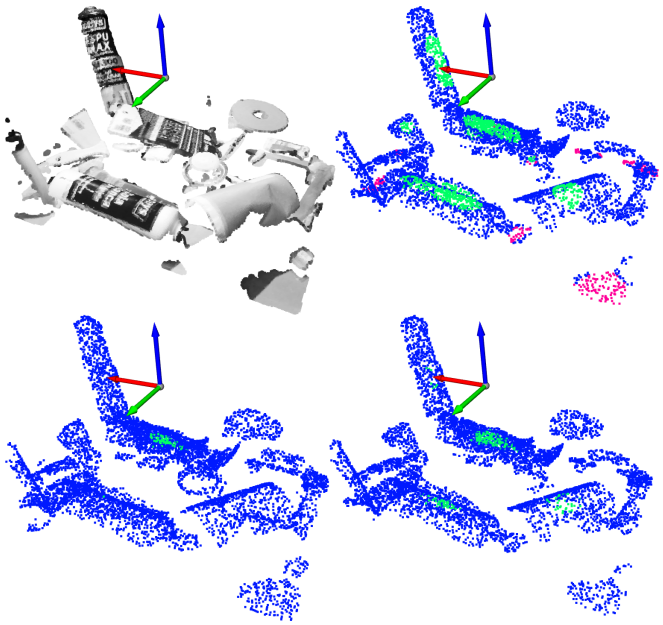
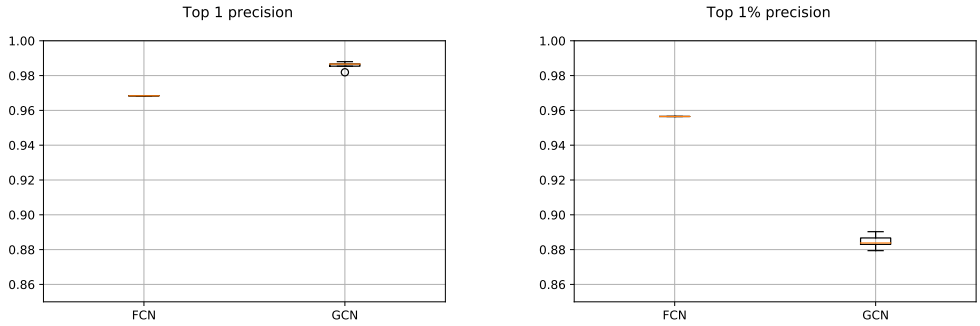


Fig. 4.12.: Suction result example. Top-left: Point cloud of the scene. Top-right: Ground-truth annotation. Bottom-left: Top-1% predictions. Bottom-right: Top-5% predictions.



(a) Top-1 scores.

(b) Top 1% scores.

Fig. 4.13.: Obtained precision scores per confidence percentiles for the gripper models in with known objects.

In this example it can be appreciated that the GCN model was able to correctly assign high affordance scores also to objects that were not annotated as good or bad. In addition, sometimes the model showed an overconfident performance predicting relatively high affordance scores to points that were not good grasping points. This could happen due to the fact that only hard negative samples were annotated as bad.

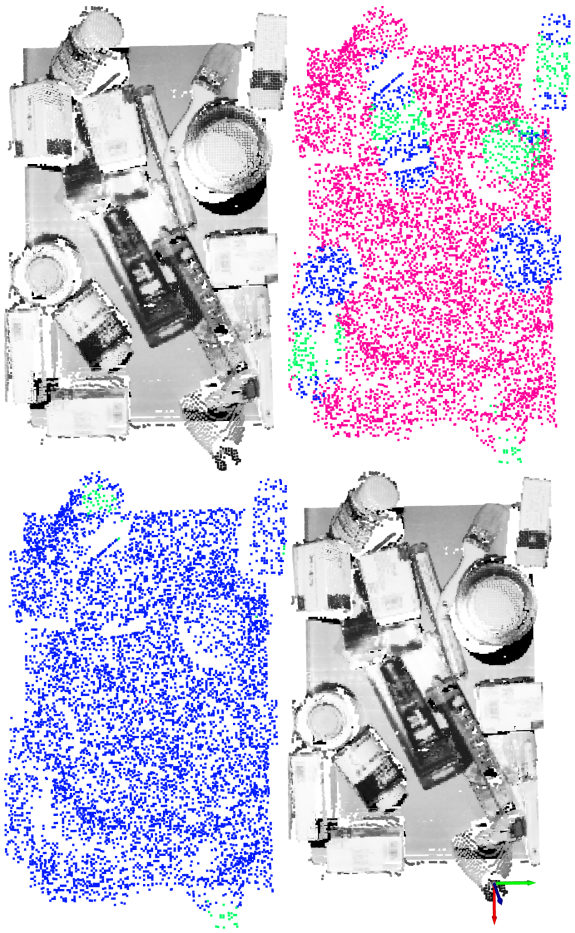


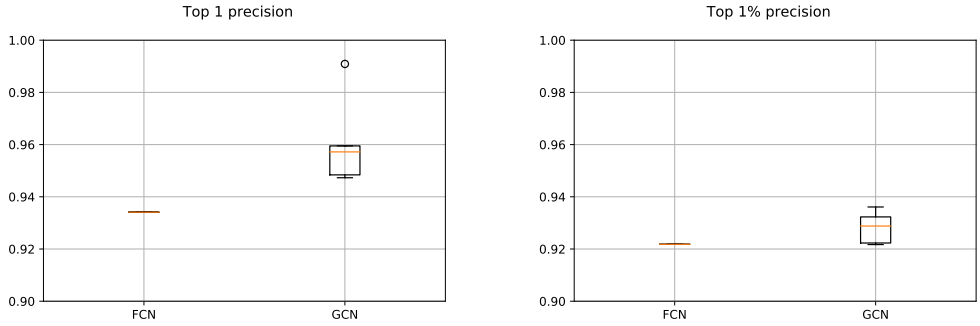
Fig. 4.14.: Gripper result example for $n = 4$ (90°). Top-left: Point cloud of the scene. Top-right: Ground-truth annotation. Bottom-left: Top-1% predictions. Bottom-right: Top-1 grasping point. The vertical orientation for the gripper is determined by the y (green) axis.

4.3.6.2. Novel objects

The main objective of this test was to evaluate the generalisation ability of the models against 100 completely new scenes composed of objects not used to generate the training dataset.

The evaluation methodology used in this test was exactly the same as for the known objects. The precisions obtained in the case of the suction model are shown in Figure 4.15. As it can be seen, both Top-1 and Top-1% precision scores were clearly superior using the GCN. This indicates that the data augmentation performed helped not to over-fit the training data, leading to better performance in scenes composed of randomly arranged new objects. Concerning the FCN, although no data

augmentation was applied, it showed a surprisingly good generalisation capability considering the reduced size of the training dataset.



(a) Top-1 scores.

(b) Top 1% scores.

Fig. 4.15.: Obtained precision scores per confidence percentiles for the suction models wit novel objects.

A graphical example of the predicted affordances with Top-1% and Top-5% confidences are shown in Figure 4.16.

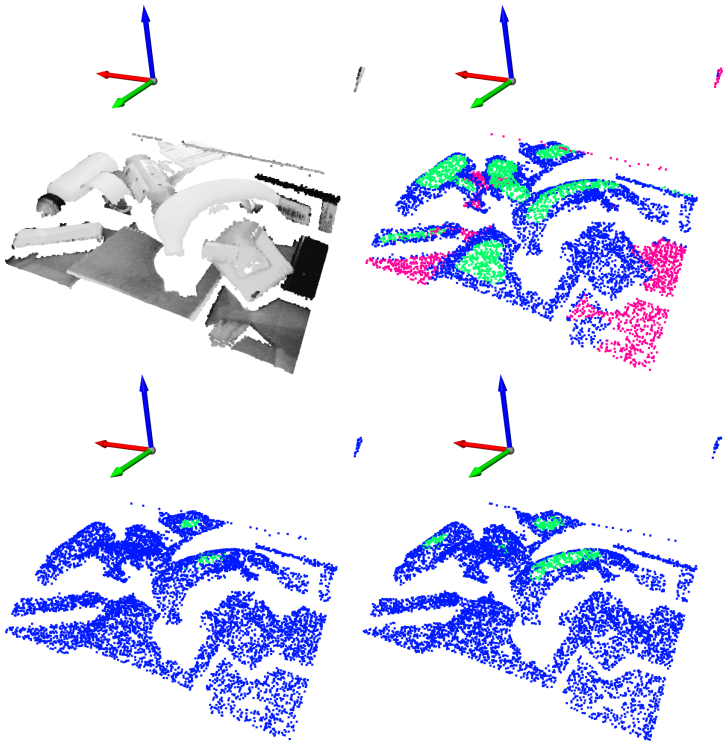
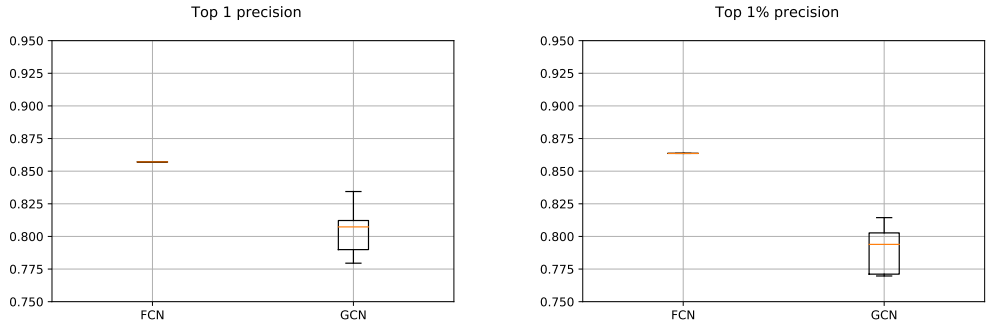


Fig. 4.16.: Suction result example with totally new objects. Top-left: Point cloud of the scene. Top-right: Ground-truth annotation. Bottom-left: Top-1% predictions. Bottom-right: Top-5% predictions.

The results corresponding to the gripper are depicted in Figure 4.17. As it can be seen, the behaviour of the GCN got much worse when evaluated with new objects



(a) Top-1 scores.

(b) Top 1% scores.

Fig. 4.17.: Obtained precision scores per confidence percentiles for the gripper models with novel objects.

and performed worse than the FCN. This may be a symptom of focusing too much on learning the characteristics of the specific objects used to generate the dataset. Although the random sampling used with the n-dimensional point clouds helped to introduce variability into the data, it was not sufficient to learn generic enough representations to generalise to new objects. A visual example of the affordances obtained in a scene with rotation $n = 2$ can be seen in Figure 4.18.

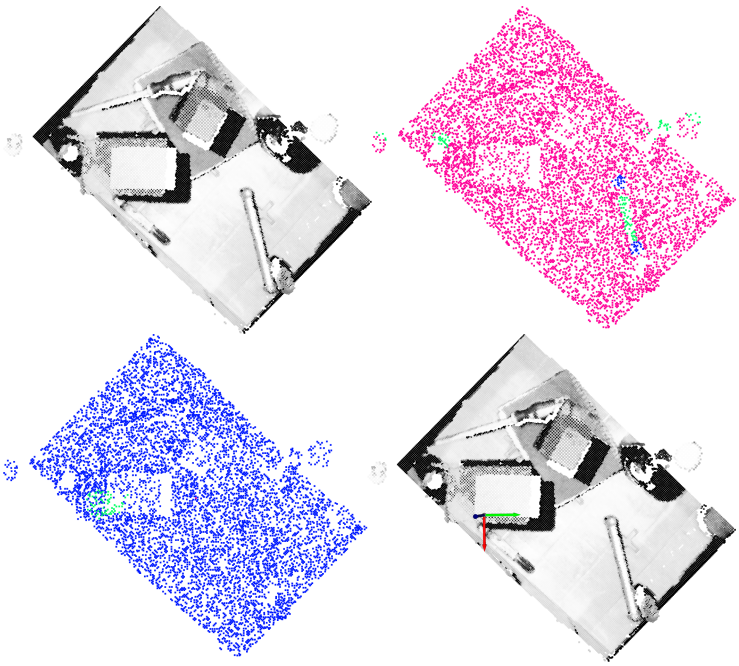


Fig. 4.18.: Gripper result example for $n = 2$ $(45°)$. Top-left: Point cloud of the scene. Top-right: Ground-truth annotation. Bottom-left: Top-1% predictions. Bottom-right: Top-1 grasping point. The vertical orientation for the gripper is determined by the y (green) axis.

4.4 Innovation

Algorithms such as those based on GCNs have performed excellently to learn about n-dimensional point clouds, thus solving some of the limitations of algorithms based on 2D data. Most approaches based on both sampling and one-shot use 3D models of the objects to generate labelled data in simulation. Although this facilitates the generation of labelled data, especially in certain industrial environments such as logistics centres or warehouses, it is not always possible to obtain 3D models of the objects. Besides, due to the difficulty of simulating real environmental conditions, the sim-to-real gap can be a problem when deploying the model in a real scenario. Furthermore, in the works analysed, the labelling of grasping points is generally carried out automatically on unitary objects, ignoring the situations that can occur in more complex scenes. However, in complex scenes there may be occlusions and objects may be entangled with each other, and thus, human judgement is fundamental to determine which the best points are but difficult to automate.

Seeing the success of the methodology proposed by Zeng et al. in [91] in the Amazon Robotics Challenge, we followed their idea to train a single-shot model to predict object affordances not in images but in point clouds. These are the main contributions:

- The state-of-the-art methods only focus on two-finger grippers. In contrast we do consider more than one gripper to make the solution more flexible, being able to pick up a wider variety of objects.

- The analysed works focus on picking operations of objects on a table. Alternatively, we propose a bin-picking oriented pipeline which makes the solution agnostic to both, the location of the box in the scene and the number of cameras used.

- In [91] they approach the problem of detecting the affordance of objects as an image segmentation problem using FCNs, and their one-shot system is able to assign a score to each pixel indicating the probability that it is a good grasping point. Instead, we propose to use a GCN oriented to the segmentation of point clouds and adapt it to predict the affordance of objects for two-finger gripper and suction graspers. Specifically, we selected the architecture presented by Li et al. in [107], which solves several limitations such as over-smoothing or

over-fitting of traditionally used GCNs [121, 113] and allows creating deep GCNs.

- To show that the use of n-dimensional point clouds helps to predict grasping points more precisely in multi-reference random bin-picking scenes, we compared our GCN-based method to a state-of-the-art FCN. The idea was to generate a single labelled dataset that would be valid for training both the image-based model and the point cloud-based model, in order to see which of the two obtained better results. To that end, a dataset of RGB-D images was generated, composed of bin-picking scenes with different complexities. Since the transformation of RGB-D images to single-view point clouds is trivial using the intrinsic parameters of the camera, the labelling was performed only on the RGB-D images. Unlike the general trend of automating the labelling using simulated environments, the annotation process was carried out by hand as human judgement is vital to account for occlusions and entanglements in complex scenes, which is difficult to automate.

To evaluate the performance of our system in comparison to the image-based solution, in a first test we evaluated the generalisation ability of the models with known objects but in scenes with completely new configurations. In a second test, the generalisation ability of the models was also assessed but with unknown objects in random configurations.

4.5 Conclusions and future work

In the research carried out, a GCN-based model was developed to predict object affordances using single-view n-dimensional point clouds of bin-picking scenes, both for suction and two-finger gripper end effectors. The intuition was that the use of higher dimensional data would lead to a performance boost, comparing to 2D FCN models that use RGB-D images. In addition, a bin-picking data processing pipeline was developed to make the solution generic for any bin-picking application. To demonstrate both the performance and the generalisation capability of the developed model, we compared it to a baseline FCN-based approach in a random bin-picking application. For this purpose, a single dataset was generated for both models and the labelling was manually carried out indicating which were the affordable zones for a suction tool and a two-finger gripper. The idea behind this comparison was to

train the models with the same dataset, the first one with RGB-D images directly and the second one with the same data transformed into point clouds, finally to answer the research question RQ3.

According to the findings, in general we can conclude that the use of n-dimensional point clouds does help in predicting grasping points in a random bin-picking problem. In the first test carried out, both models were assessed with scenes composed of known objects but in totally random configurations. It was shown that especially in the case of the suction tool, the precision obtained improves substantially when learning is performed on n-dimensional point clouds using GCNs. In the second test instead, we tried to compare the generalisation capability of the models with totally new items in randomly generated configurations. The obtained results demonstrated that the GCN-based suction model had strong generalisation capabilities to correctly predict affordances in similar but completely new parts. However, the precision scores obtained with the gripper indicated that the FCN generalised better to new parts than the GCN, suggesting that the input data was not significant enough for the GCN in the more complex gripper scenario.

Although the results obtained have contributed to answering our research question, the system developed had several limitations:

- Because each point in the point cloud is represented by a n-dimensional vector (depending on the features used to describe each point), the complexity of the problem grows proportionally when more features are included in the learning process. This reduces considerably the number of points that can be analysed in each scene, and although it was sufficient with the type of objects that have been used during the study, this can be problematic with very small objects.

- As it is known, 3D cameras often have problems when reconstructing the information of transparent elements, which means that only the colour information of these points can be used to infer the affordance. Although in this work we have not made a specific study with this type of objects, the fact that there is no spatial information of these objects could cause the proposed solution to fail.

- In the case of the suction tool, since the grasping point is composed of a single contact point, its normal vector is sufficient to define the orientation of the end effector. In the case of the two-finger gripper, however, the orientation of the gripper plays a very important role since each grasping point is composed

of two contact points. In our case, as it is always accessed vertically inside the bin, the scenes are rotated before they are introduced to the model to predict the vertical rotation of the gripper. Finally the rotation that gives the highest affordance is selected. The fact of having to make an inference for each possible vertical rotation makes the solution hardly scalable, especially when working on point clouds, where the computational cost is much higher.

As future work, the experience and knowledge acquired during the development of this work should be enriched by testing the models with the real robotic system in a real application to see whether the learned grasping representations are valid to pick real objects. Moreover, the manual annotation is a very time-consuming process. Although data augmentation techniques somehow alleviate it, the DL models still need too much annotated data to converge. Thus, we must look for ways to learn to predict object affordances, with less manually annotated samples. Lastly, the grasping strategy for the gripper must be extended to 6-DoF in a more flexible way to overcome the current generalisation and scalability limitations.

4.6 Publications

This line of research resulted in the following publication:

- Ander Iriondo Azpiri et al. “Affordance-Based Grasping Point Detection Using Graph Convolutional Networks for Industrial Bin-Picking Applications”. In: *Sensors* 21.3 (2021), p. 816

This article is attached to PART II of the work.

5 Dynamic mosaic planning for a robotic packing system

Along with object picking, placing is one of the core problems in robotic manipulation. In contrast to simple pick-and-drop methods, the aim of object placing is to find the best dropping pose taking into account criteria such as stability or compactness. Although there are several ways to drop an object, packing objects into boxes, better known as bin-packing, is one of the methods that has historically attracted most attention, especially in industrial environments such as logistics centres or large warehouses, due to the cost of storing and transporting oversized containers. In these industrial environments currently the packing processes are largely carried out manually. The great variety of objects to be handled (different material, weight, shape, texture, etc.) makes it very difficult to automate the bin-packing process.

Bin-packing solutions differ in the amount of prior information available to the application. Traditionally, bin-packing algorithms have focused on calculating offline the optimal drop-off configuration for a set of regular objects with known properties. On the one hand, these offline methods generally assume that objects can be packed in any orientation, and thus omit the presence of an automated picking/packing system such as a robot. On the other hand, they ignore the dynamics of a real packing environment and assume that the theoretical calculation of the mosaic (packing configuration) accurately reflects reality. However, the reality is that generally none of the above mentioned assumptions are fulfilled in industrial environments and therefore more flexible systems are needed.

In this context, this work focuses on developing a flexible online packing solution agnostic to the references being packed. The fact that the system is online allows us to take into account the dynamics of the environment each time an object is packed, and thus consider the actual packing situation in the calculation of the drop position of the next object in each iteration. Furthermore, because our system is reference-agnostic, it allows the system to be able to pack a wide variety of objects without having to be adjusted for each reference.

5.1 State of the art

Currently there are exact mosaic computation methods that solve the 3D bin packing offline and in an optimal way using cubic objects. One of the most popular was presented by Martello and Vigo in [122] and [123], where they use a branch-and-bound algorithm to optimally solve the problem. This method was later improved by Den Boef et al. in [124]. The bin-packing problem belongs to the NP-hard class and therefore cannot be solved in polynomial time. Therefore, a lot of work has been done on approximate solutions that search for near-optimal solutions in a much more agile way. The best known algorithms are *bottom-left* developed by Jakobs in [125], and *best-fit-decreasing* developed and analysed by Johnson et al. in [126]. However, these approximate algorithms were originally designed for 1D and 2D bin-packing.

Subsequently, meta-heuristic algorithms proved to be able to generate better solutions for the 3D bin-packing problem, comparing to previous approximate solutions. Algorithms such as *tabu-search* [127], *guided local search* [128], *genetic algorithms* [129] and even learning-based algorithms such as DRL have been successfully applied to generate valid packaging solutions efficiently [130]. These types of methods have two main limitations for a real packaging environment. On the one hand, these are designed to work only with cubic objects and assume that those can be packed in any orientation. However, if the packaging is to be carried out by a robot, the way in which the object is picked up considerably limits the possible drop poses. On the other hand, such algorithms assume that the objects to be packed are known and compute the entire mosaic offline, thus ignoring the dynamics of a real packing environment.

The reality in industrial intralogistics environments and department stores is that objects with more and more variability in material, shape and texture need to be handled. Especially due to the growth of e-commerce, unfortunately the objects to be packed are not cubic, and thus, the possible packing configurations are infinite. Therefore, the packing problem with irregular objects can only be solved by approximate solutions. In addition, when packing irregular objects, the probability that the generated mosaics are not stable increases considerably. This means that every time a new object is packed, it may move to a different position than the expected one. Consequently, offline planning of the mosaics is not practical and the monitoring of the real state of the mosaic is required.

In that vein, Wang and Hauser [131] proposed an heuristic based method that considered the offline packing of geometrically complex 3D objects with stability constraints. In their work, the picked parts were represented as 2D heightmaps that were generated using ray-casting, and the availability of the outbound box was monitored using a second 2D heightmap that was updated at each iteration of the offline planning. Despite authors considered an automatic packing system that restricts the packing orientations, uncertainties in the outbound box were ignored since the entire planning was carried out offline. Besides, the system was only assessed in a simulated environment. In a recent approach, Zhao et al. used a meta-heuristic algorithm to estimate a near-optimal packing mosaic for irregular objects [132]. There, the items were represented as meshes and the empty space as the bounding box of the bin. However, the approach was time demanding and could not be used in real time. Moreover, an automatic packing system was omitted, and it was assumed that the theoretical mosaic was a faithful representation of the real one, without considering the dynamics of a real packing system.

Offline methods have generally been successful in calculating tiles efficiently and achieve mosaics with a high degree of space utilisation. However, offline computation methods are hardly applicable in a real multi-reference object packing application, as it is difficult to fulfil the assumption that a robot is able to leave each and every object in the expected pose. The varying shape, material and texture of the objects themselves can cause them to move or fall once placed in the mosaic.

On the other hand, online methods incrementally calculate the mosaic as new objects are placed. The need for online mosaic generation is motivated by two main reasons: (1) the objects to be packed are not known or arrive in a non-deterministic way, and (2) The monitoring of the real mosaic is needed to get an accurate representation of its state. For example, Ha et al. developed a heuristic algorithm with the goal of packing orthogonal elements as they arrive and without having any prior information related to those objects [133]. However, their approach was only tested in a simulated environment, without taking into account the presence of an automated packing system and ignoring the uncertainties that may occur in a real environment. The framework developed by Wang and Hauser [134] dealt with the packing of a known set of objects arriving in a non-deterministic order. The proposed solution was composed of an offline planner and a verification algorithm that checked the feasibility of all permutations of arrival orders. Although the system handled both regular and irregular objects, an automatic packaging system was not used, and therefore, they did not monitor the status of the destination box, omitting

uncertainties such as drops in it. Additionally, authors themselves stated that their work was practical only with a small subset of objects.

Hong et al. went one step further and solved the irregular 3D object bin-packing in an online manner [135]. Regarding the monitoring of the picked part, a depth camera was used to estimate the area (highest contour) of the object, which was located below the picked part and aligned with the end effector of the robot. Concerning the destination box monitoring, the status of the bin was represented as a 2D depth-map and an analysis of the available contact-free 2D positions was performed, using the contour information of the picked part. Additionally, some other criteria such as placing the objects in the lowest possible position or bringing things together were applied. Like in the above mentioned references, a real picking and packing system was not used (only simulated some manipulation error) which could have a large effect in the structure of the final mosaic. Learning based algorithms such as DRL were also successfully applied in works such as [136] and [137]. The former lacked an automatic picking system and was only tested in a simulated environment. The latter, on the other hand, assumed that the system was able to perfectly place the item and omitted the uncertainties that could happen in the destination bin. Both of them were designed to work only with orthogonal objects.

Summing up, most of the works in the literature use orthogonal items and assume that the packing process is ideal . Therefore, they do not provide online monitoring of the status of both the picked part and the outbound box. However, this is crucial particularly in real unstructured scenarios with variable packing conditions and irregular multi-reference parts.

5.2 Goal

In the context of the PICKPLACE project, and in relation to the technical objective TO4: "Human and robot aware dynamic package planning", the objective was to develop a flexible packing system capable of packing a wide variety of objects in an orderly manner in a mosaic and without having to configure the system specifically for each reference. Precisely, three pick and package use cases were considered in the project: Mono-reference to multi-reference, multi-reference to mono-reference and multi-reference to multi-reference.

On the one hand, given that the variety of objects to be manipulated could be very large, it was not possible to assume that the mosaic to be built would be stable. On the other hand, the fact of using a robot to pack the objects introduces uncertainty in the packing process, and therefore it is necessary to take into account the dynamics of the packing process. For these two reasons it was necessary to implement an online packaging system, which would monitor the current state of the packaging at any given moment and plan accordingly.

Having said that, the research question that we asked ourselves is the following:

- **RQ4:** Are we able to develop a flexible bin-packing robotic system that handles the uncertainties introduced by an automatic packing system and packs arbitrary objects?

5.3 Developed approach

The developed robotic online packing system, capable of packing any object that the robot has been able to pick up, is composed of the following three main phases: monitoring of the picked object, monitoring of the target box and dynamic mosaic planning.

Thanks to the developed flexible grasping point detection system (see Chapter 4) the robot is able to pick up a wide variety of objects with both the suction tool and the two-finger gripper. Because this system is agnostic to the identity of the objects and only focuses on their appearance, the grasping points may vary if the sensory information varies. Therefore, a picked part monitoring algorithm has been developed, on the one hand to know how the robot has grasped it, and on the other hand, to estimate its dimensions and orientation.

Both the fact of using objects with different morphology and the use of a robot to pack the objects automatically led to uncertainties in the mosaic generation process (i.e. objects can move once the robot has left them). This leads to uncertainties in the mosaic generation process and therefore a heuristic algorithm has been developed that monitors its state each time a new object is placed, and calculates the free space in the form of cubes.

Finally, the developed dynamic mosaic scheduler, dubbed Dynamic IK-PAL, is based on how the current object has been picked and what is the state of the target box to heuristically estimate the best drop pose for the picked object. All the modules developed were integrated and validated on a real robotic system to demonstrate the system's ability to generate mosaics in a flexible way, with a wide variety of objects and without the need to be configured for each reference.

5.3.1 Theoretical background

5.3.1.1. Offline IK-PAL

Offline IK-PAL [1] is the offline mosaic planning algorithm previously developed by UHS which plans tiles with cubic objects. This algorithm assumes that all objects to be palletised are known and that they are always picked in the same way using manually defined grasping points. Since the mosaic planning is done offline, it is assumed that the robot will always be able to place the objects in the previously planned poses. In fact, every time an unexpected movement of the objects occurs, human intervention is necessary. The offline mosaic planning process consists of 3 main tasks: Data pre-treatment, layer calculation and sequencing.

Data pre-treatment The UHS' Warehouse Management System (WMS) is responsible for managing both order preparation and returns. Each time an order is generated, the WMS generates an input INP file which describes the properties of the pallet on which the mosaic is to be built, along with the information of each object that make up the order. The data related to both the pallet and objects in the INP file are shown in Table 5.1 and 5.2 respectively.

Keyword	"pallet" keyword
Length	Pallet X axis length
Width	Pallet Y axis length
Height	Pallet Z axis length
Max. height	Maximum pallet height
Max. weight	Maximum pallet weight

Tab. 5.1.: Features of the pallet in the INP file.

[1] https://www.ulmahandling.com/en/automated-intralogistics/system-robot-ik-pal

Ref	Product's reference number
Quantity	Number of units to include in the mosaic
X_dim	Product's length
Y_dim	Product's width
Z_dim	Product's height
X	X coordinate in the pallet
Y	Y coordinate in the pallet
Z	Z coordinate in the pallet
Weight	Product's weight
I	Robot entering direction
Product's side in the pallet	Left or right
K	Palletising sequence (nº of packing)
Morphology	Group according to product's morphology

Tab. 5.2.: Features of the objects in the INP file.

The offline IK-PAL algorithm uses this information to calculate how many pallets are needed to prepare the order, and determines which objects will be placed on each pallet. To that end criteria such as products' volume, degree of filling and maximum weight/height of each pallet is used.

Layer calculation All the objects are represented by their bounding box, which is then used to estimate their configuration inside the pallet. The products inside a pallet are arranged in layers or groups (see Figure 5.1). These layers are imaginary divisions inside the pallet that are established by the final customer according to their own criteria. These layers are composed of individual boxes or group of boxes with similar morphology that can be joined to form bigger box.

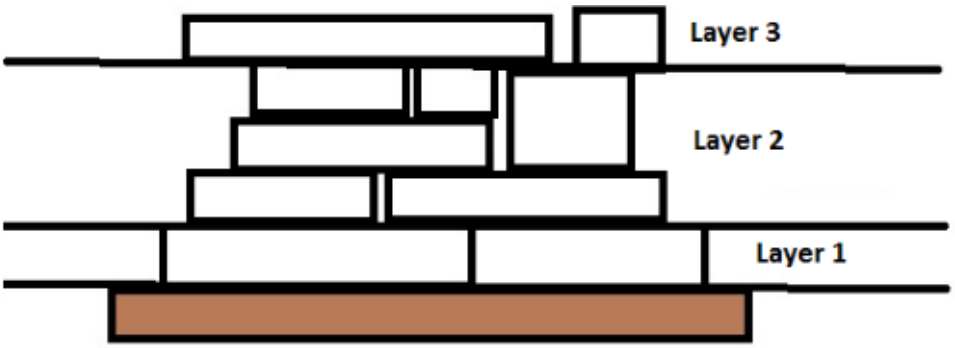


Fig. 5.1.: Products organised in layers.

The generated layers can be of full-size, half-size or quarter-size (see Figure 5.2).

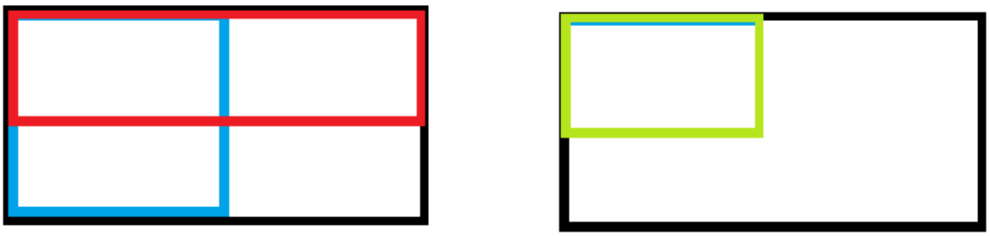


Fig. 5.2.: Generated layers depending on their size. In black a full-size layer. In red and blue, length and width half-size layers respectively. In green a quarter-size layer.

The packing problem is tackled as a 2D bin-packing problem (2BP). This problem consists of packing a set of rectangles into the minimum number of identical large rectangles (bins) without overlapping and the edges of the objects being parallel to each other. Since this is used to pack objects in 3D, the following steps are carried out:

1. Classify all the items with similar height.

2. Arrange these products to complete a full, half or quarter layer using an heuristic algorithm. To that end a fulfilment parameter is used related to the degree of filling.

3. Stability considerations. The first layer of the mosaic is always stable. However, for the rest of the layers the stability decreases. To calculate which products will be placed in each layer, the stacking and morphology features are considered. For instance, the most regular and less fragile objects will likely be placed in the first layer.

If it is not possible to fulfil a layer, the algorithm tries to commit half-size layers, both horizontally or vertically. If again this is not possible, quarter-size layers are calculated. The theoretical representation of a mosaic is shown in Figure 5.3.

Sequencing Once the mosaic has been calculated, this step provides the sequence in which each product will be manipulated. The result of the mosaic planning is an OUT file which describes both the sequence of products and how will they arrive to the order preparation area. The information that an OUT file contains can be seen in Table 5.3.

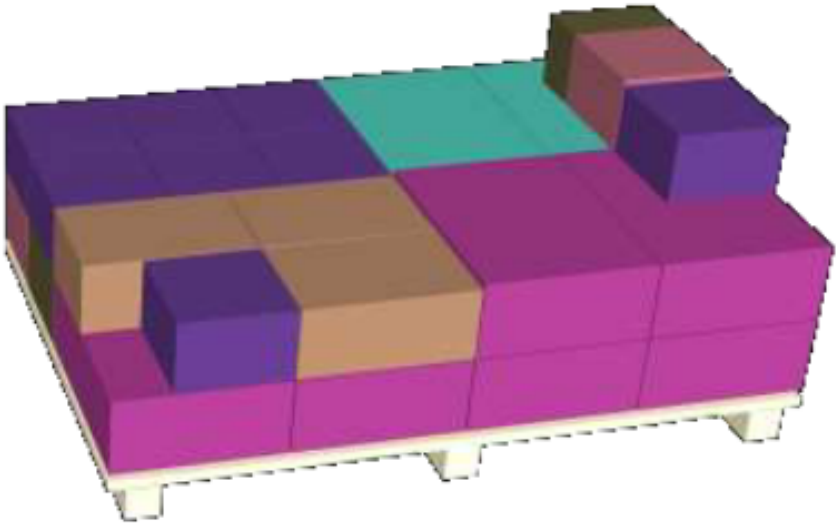


Fig. 5.3.: Theoretical representation of a mosaic.

Ref	Product's reference number
Quantity	Number of units to include in the mosaic
X	Product's length
Y	Product's width
Z	Product's height
Weight	Product's weight
Valid positions	Whether product can be rotated $90°$
Groups	Product's family
Stacking	Normal or fragile product
Morphology	Group according to product's morphology

Tab. 5.3.: Features of the objects in the OUT file.

Finally, the WMS sends this information to the PLC of the robot that is in charge of the picking/packing process.

5.3.1.2. Empty maximal spaces

To indicate the feasible positions where an object can be packed inside a container, several concepts such as corner points [138] or extreme points [139, 140] have been used.

In this work the concept of empty maximal spaces (EMSs) is used to represent the empty space inside the container. As it can be seen in Figure 5.4, after packing an

object the idea is to estimate the largest possible orthogonal empty volumes. As can

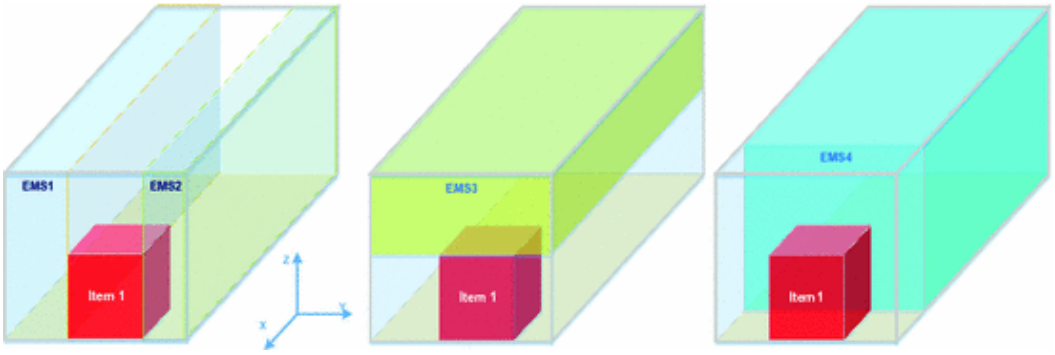


Fig. 5.4.: Empty maximal spaces after packing an item inside the container [133].

be seen, when the objects to be packed are orthogonal, the possible EMS are finite and it is possible to compute them optimally. However, when the packed objects are irregular, the possible EMS are infinite and can only be approximated.

5.3.2 Workspace

Although as previously mentioned the end users of the PICKPLACE project were both UHS and TOFAŞ, in order to simplify the validation of the system, it was carried out on the TOFAŞ prototype located at Tekniker. The TOFAŞ pilot is composed of 3 main areas (see Figure 5.5): (1) The inbound area, (2) the part monitoring area and (3) the outbound area.

The input area contains a single input container and the output area contains at most two containers. We use the 7-DoF UR10 collaborative robot with a suction end effector. The manipulator is attached to a moving track that allows the robotic arm to move from one zone to another. Above the inbound and outbound areas two cameras are placed, which are used to estimate the grasping points and to control the exit container respectively. In the first case, the camera is in a fixed position, as there is only one container in the inbound zone. In the second case, however, the camera is attached to a linear rail that allows up to two output containers to be monitored with a single camera. As for the picked parts monitoring area, three extrinsically calibrated *RealSense D435* cameras are used, all with a fixed location in the layout. This allows the system to have a full view of the part, enabling a better estimation of both dimensions and orientation. The robotic system is covered with a housing to reduce the impact of external light on the cameras. All software runs on

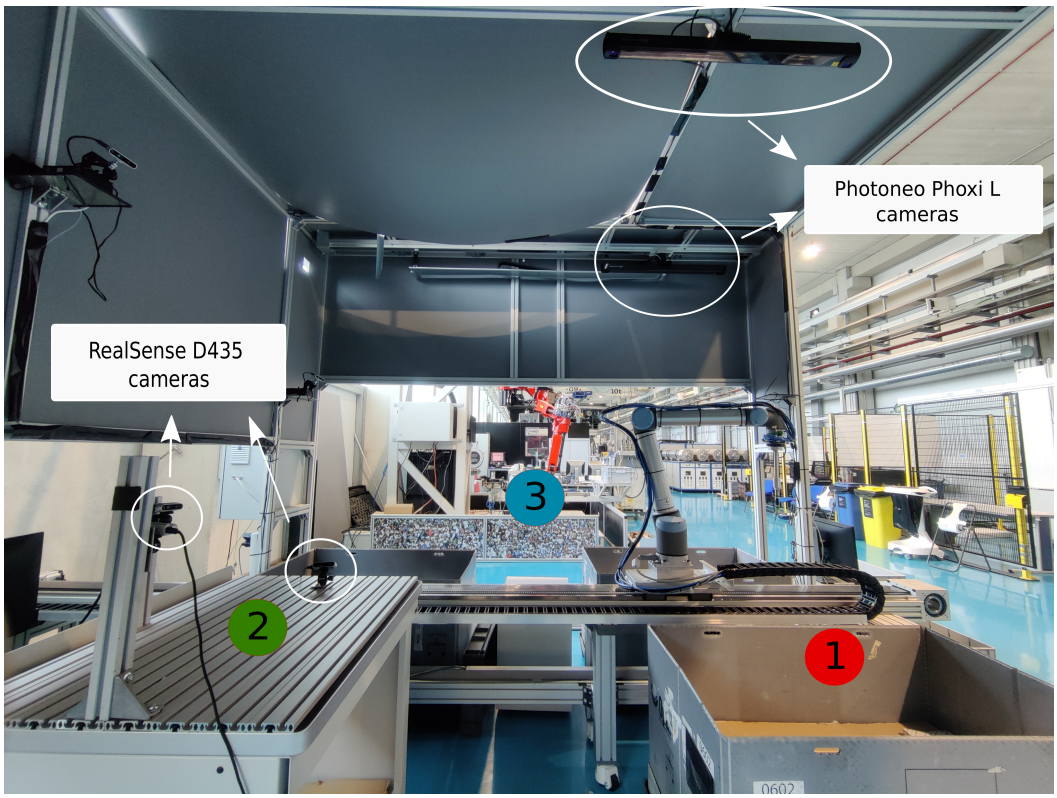


Fig. 5.5.: Main components of the TOFAŞ pilot. (1) Inbound area, (2) part monitoring area, (3) outbound area.

an Intel CPU i7-8700@3.2GHz x 12, with 32GB of RAM and an Nvidia GeForce RTX 2080 Ti GPU with 11GB of memory.

5.3.3 Use cases

In the context of the PICKPLACE project, the following pick and place use cases were considered:

1. Mono-reference to multi-reference: Typical case of order preparation. After the system receives an order from the WMS, it determines the arrival order of the mono-reference bins to the picking area. The WMS generates the sequence for a new order preparation or for a order return process, and is used in both prototypes. Once the robot picks the required object, it packs the item in the bin of the corresponding order, creating a multi-reference mosaic.

2. Multi-reference to mono-reference: Typical case of order return. The returned multi-reference bin goes directly to the picking area. The items are picked following the order that is determined by the flexible picking system, until no more items are left inside the bin. Once each object has been picked, its dimensions are estimated and, at the same time, the product is identified. Finally, each product is returned to the corresponding bin of the product's reference, creating a mono-reference mosaic.

3. Multi-reference to multi-reference: Special case of order preparation. Once each product is picked from the picking box, it is identified and placed in the corresponding multi-reference mosaic.

The first two use cases were demonstrated on the UHS prototype and the last one, instead, on the TOFAŞ prototype. Although the solution developed in the context of the project was designed with all use cases in mind, the validation of the packing system only focused on the last use case, as it is the most complex one.

5.3.4 Dynamic bin-packing system

As mentioned before, the dynamic packing system developed in the PICKPLACE project consists of the following main components:

Flexible grasping: A flexible grasping system based on detecting the objects' affordance is used to handle as many objects as possible without the need to manually configure the system for each reference. Consequently the grasping points are identified on-line and are not predefined. The implementation details are explained in Chapter 4.

Picked object monitoring: Because the grasping system is agnostic to objects' reference and only looks at their appearance, after grasping an object it is necessary to monitor it to know how it has been grasped and what its dimensions are.

Destination bin monitoring : The developed pick and drop system is able to handle objects with different morphology and consequently it is very likely that the generated packing mosaics are not stable. To handle those uncertainties it is necessary to monitor the drop box after placing a new object as objects may move.

Dynamic mosaic planning: Mainly due to the fact that the tiles cannot be assumed to be stable, the whole mosaic cannot be planned off-line and needs to be dynamic. Therefore, a planner has been developed which, receiving as input the monitorisation of both the picked object and the destination box, decides the optimal drop pose for the picked object in each iteration. Thus, the system is able to take into account the real state of the mosaic and make the calculations accordingly.

5.3.4.1. Picked part monitoring

A flexible grasping point detection system such as the one used in the context of the project does not guarantee that every object will always be grasped in the same way, since objects are not identified the grasping points are solely predicted by looking at aspects such as shape, colour, texture etc. This implies that the grasped object has to be monitored to know its dimensions and orientation with respect to the gripper. The following processing steps are performed before estimating the bounding box:

1. Combine the point clouds of the 3 *RealSense* cameras using the extrinsic calibrations.

2. Downsample the combined point cloud with a voxel-downsampling algorithm with a voxel-size of $V_b^3\ mm^3$.

3. Define a monitoring volume M_d with respect to a predefined pose in the space. The robot's end effector moves to a fixed position inside the volume to show the picked part to the cameras.

4. Remove the end effector from the point cloud. Since its dimensions and its position in the monitoring volume are known, a simple point cloud crop is done.

5. Apply a radius filter to remove the noise from the point cloud, with a filtering radius F_r and considering F_n points.

In Figure 5.6 it can be seen a graphical representation of the estimation of the bounding box. The red cube represents the monitoring volume, the green cube belongs to the estimated and optimised bounding box and the blue sphere represents the origin of the bounding box with respect to the end effector.

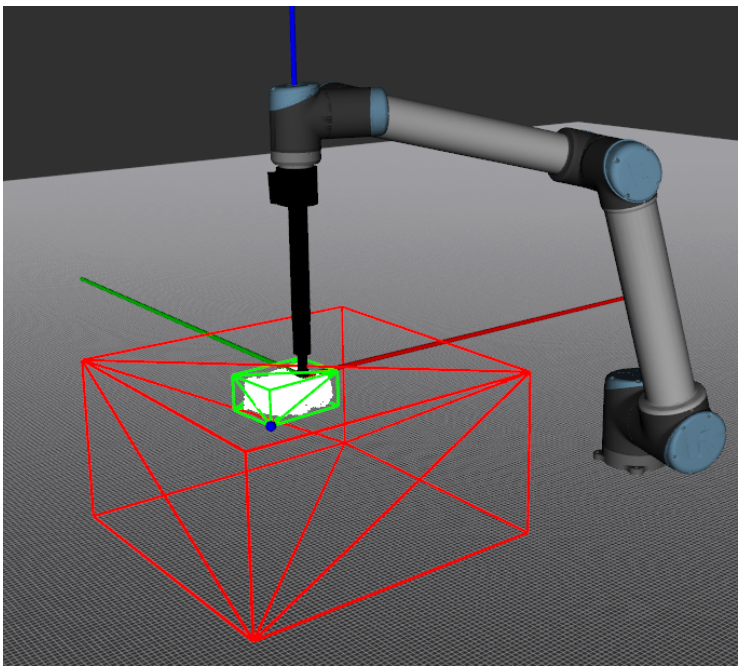


Fig. 5.6.: Bounding box estimation of the picked part. The monitoring volume and the bounding box are represented as red and green cubes respectively. The blue sphere indicates the origin of the bounding box.

After executing the processing steps, it is assumed that all the remaining points inside the monitoring volume belong to the object, and then its bounding box is estimated. Finally, the bounding box is optimised in the vertical z axis, as it is the most critical orientation during the packing process. The bounding box is not optimised in the x and y axes, as it is assumed that the error in these axes disappears due to the gravity when the item is packed. To do this, the origin of the point cloud is moved to its centroid and the cloud is rotated n discrete times with respect to the z axis, checking in which rotation the bounding box is smaller. The bounding box is always estimated along the axes of the cloud origin. Finally, the bounding box estimated with the smallest volume is transformed back from the translated coordinate frame to the end-effector coordinate frame.

5.3.4.2. Destination bin monitoring

The heuristic algorithm developed is based on the concept of Empty Maximal Spaces (EMS) previously used in several studies [129, 141, 142, 133]. As explained before in Section 5.3.1.2, EMS estimates largest free cubic volumes inside a box. In the context of 3D bin-packing applications, this information is later used to estimate

the optimal drop-off position for the next object. Thus, the idea is to iteratively estimate the EMSs after placing each object in the drop box. However, previous works only use orthogonal objects, the mosaic calculation is carried out offline using their theoretical poses, and thus, bypassing the dynamics of a real packing environment. In these cases where orthogonal objects are used and the estimation of the drop-off poses is performed using the theoretical representation of the mosaic, the number of EMSs that can be generated is finite and can therefore be calculated optimally. However, when these assumptions are not met and the mosaic calculation is based on the observed mosaic state obtained from sensor information (e.g. point cloud data), the number of possible EMSs is infinite and, thus, only approximate solutions can be obtained. Our approach does take into account the uncertainties and dynamics of a real packing system and is able to online estimate the optimal packing position for the next object.

We propose an heuristic algorithm that receives as input a point cloud of the scene and the location of the container in the scene and calculates the usable free volume inside the box in the form of free cubic volumes. A graphical representation of the EMSs generated in an example scene can be seen in Figure 5.7.

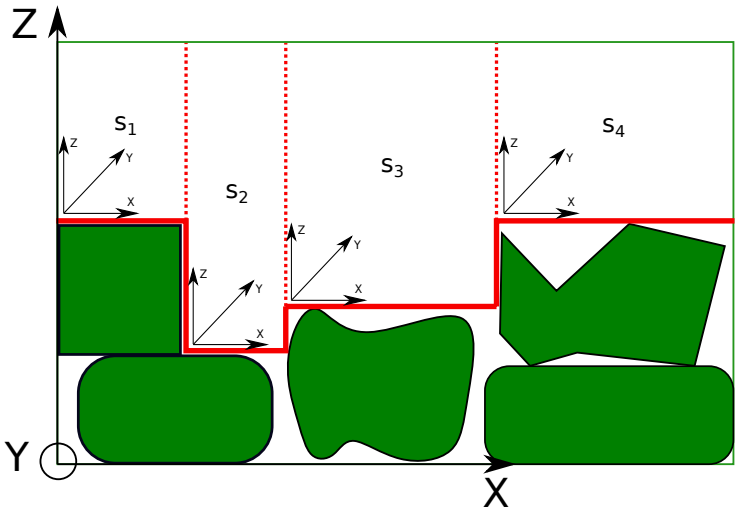


Fig. 5.7.: Side view of the generated empty cubes in 2D in an example scene. Green items represent already packed objects and s_1, s_2, s_3, s_4 are the generated EMSs for that scene.

The algorithm developed to estimate EMSs consists of four main processing phases and they are as follows:

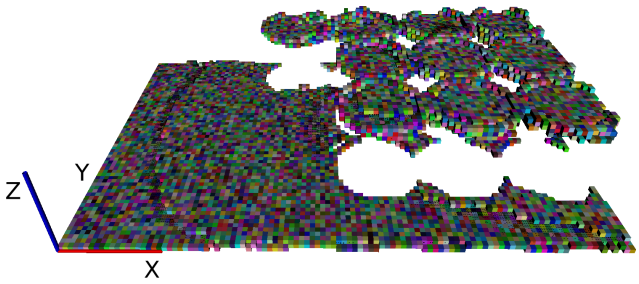
1. **Preprocessing:** First, knowing the location of the drop box in the scene and its dimensions, the point cloud is filtered to keep only the information that

belongs to its interior. Then, two noise reduction algorithms are applied, one based on clusters to remove groups of points that are not attached to the main point cloud, and the other to statistically remove the noise in the main point cloud. Finally, the occupied space is voxelised with voxel size V_s and represented as an occupancy grid for easy processing. An occupancy grid of a sample scene can be seen in Figure 5.8a.

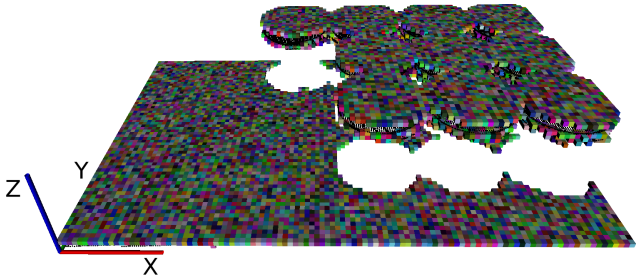
2. **Stabilisation:** The main objective of this second processing step is to uniformise the occupancy grid in order to generate planes to facilitate the calculation of the EMSs. To perform the stabilisation, a height threshold H is used and the aim is to generate uniform planes by modifying existing occupied voxels. An example of the occupancy grid after executing the stabilisation processing phase is depicted in Figure 5.8b.

3. **Square generation:** Two are the goals of this process. On the one hand, we seek to fill unoccupied holes, and on the other hand, we give the planes a square shape. The reason for these steps is that the EMSs are cubic and therefore they have to be built on a square or rectangular shaped occupied plane. For this purpose a second threshold Q is used and holes with an area equal to or smaller than Q^2 voxels are filled. An example of the occupancy grid after executing the stabilisation processing phase is depicted in Figure 5.8c.

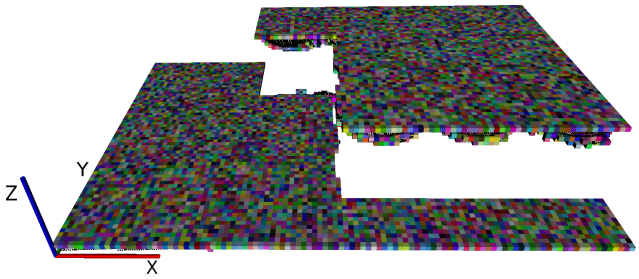
4. **Solution generation:** In this last phase the objective is to generate the EMSs using the previously processed occupancy grid. The cubic solutions are generated on occupied planes. The dimensions of the base of each cube are determined by the dimensions of the plane on which the cubic solutions are built. The height of each cube is defined by the difference of the height of the cube and the height of each plane on which the EMS is generated. At this point, the paradigm is changed and instead of processing at the occupancy grid level, it is done at the EMS level. Because the proposed solution is approximate, the generated EMSs are not optimal and therefore in a last step an attempt is made to join neighbouring solutions. The parameter J indicates whether the joining step is performed or not. A projection of the generated EMSs can be seen in Figure 5.9.



(a) Initial occupancy grid with $V_s = 5\ mm$.



(b) Occupancy grid after the stabilisation phase with $H = 5$.



(c) Occupancy grid after square generation phase with $Q = 20$.

Fig. 5.8.: Occupancy grid obtained after each processing phase.

Fig. 5.9.: Projections of the cubic solutions that represent the empty space inside the bin.

5.3.4.3. Dynamic IK-PAL

IK-PAL is the proprietary mosaic planning algorithm developed by UHS. The first version of this algorithm was an offline mosaic planner that could plan complete

mosaics with regular parts (see Section 5.3.1.1). This offline planner assumed that the items were always picked and packed in the same way, using the same gripping points, and that the set of items to be packed was known in advance. Furthermore, this method was designed to be used only with regular pieces, assuming that after placing each object in the target box, the mosaic would remain static. In fact, manual intervention by the operator was necessary every time an unexpected movement occurred or an object fell. However, this assumption of a static mosaic does not hold true when irregular items are introduced into the real robotic system. Therefore, the mosaic must be created online taking into account the state of the packaging after each packaging operation.

Therefore, in the context of the PICKPLACE project, a new dynamic version of the algorithm was designed and implemented. The new version of the mosaic planner receives as input the list of estimated EMSs at the destination bin and the bounding box of the picked item. As a result, it returns the 3D coordinate within the target container where the origin of the bounding box should be located, plus the required rotation of the picked item on the z-axis. Since the picked parts are represented as boxes first to calculate their dimensions and then to calculate their drop-off position, the rotation on the z-axis is defined by a binary variable, where a rotation of 90° is applied when enabled.

The Dynamic IK-PAL is a heuristic-based mosaic planner that, given only the current packing state, estimates the next packing pose with the following criteria:

1. First, the EMSs where the bounding box fits in are extracted. In each selected empty volume, there are two possible poses for the bounding box: As it is or with the rotation enabled.

2. A tree data structure is generated with all the possible solutions.

3. Criteria such as the degree of filling, compactness or the size of the remaining empty volumes are considered to rate them accordingly.

4. The solution with the highest rating is selected.

5.3.4.4. Flow of the packaging application

Algorithm 3 shows the flow of the dynamic bin-packing application.

Algorithm 3 Simplified dynamic bin packing application

```
1: exit ← false
2: grasping_points ← estimateGraspingPoints()
3: while not exit & grasping_points do
4:     // Step 1: Pick an object and estimate EMSs in target box
5:     move_robot("inbound_area") &
   EMS ← destinationBinMonitoring()
6:     errors ← 0
7:     picked? ← false
8:     while errors < N or not picked? do
9:         picked? ← pickItem(grasping_points[errors])
10:        if not picked? then
11:            errors ← errors + 1
12:        end if
13:    end while
14:    if errors = N then
15:        exit ← true
16:    else
17:        // Step 2: Estimate grasping points for the next cycle
18:        move_robot("part_monitoring_area") &
   grasping_points ← estimateGraspingPoints()
19:        // Step 3: Monitor the picked part
20:        orig, dims ← pickedPartMonitoring()
21:        // Step 4: Calculate optimal packing pose
22:        move_robot("outbound_area") &
   packing_pose ← DynamicIKPAL(EMS, dims)
23:        // Step 5: Pack the object
24:        packItem(packing_pose, orig, dims)
25:    end if
26: end while
```

The process runs as follows:

1. The robot is commanded to go to the inbound area and while it is moving, the drop-off container is monitored and EMSs are estimated. Thus, once the robot arrives at the destination, the EMSs are already calculated and those will later be used to calculate the drop-off pose of the picked object. Once there, the robot has N attempts to grab an object. At each attempt, it tries to pick up the

object corresponding to the selected grasping point. In case of failure, that grasping point is discarded and the next one is tried.

2. Once the object has been picked up, the robot is commanded to go to the picked part monitoring area and in parallel it proceeds to estimate the grasping points for the next cycle. The idea is to estimate the grasping points for the next cycle once the robotic arm leaves the input container, assuming that once the object is removed, the objects will not move any more.

3. Once the robot is in the picked part monitoring zone, it proceeds to monitor the picked object. The objective of this step is to associate the picked object to a cube which is optimal in the z axis. Thus, both the origin of the bounding box of the object with respect to the robot tool and its dimensions are estimated. This monitoring is then used, together with the EMSs, to estimate the drop pose.

4. After monitoring the picked object, the robot is commanded to move to the outbound area. In the meantime, using the estimated EMS and the bounding box information of the picked object, the Dynamic IK-PAL is called to estimate the final drop pose.

5. By the time the robot reaches the outbound area, the drop pose is already calculated and therefore the robot places the origin of the bounding box of the picked object in the estimated pose and then releases the object.

5.3.5 Experimentation

Before validating the entire system in a real packing environment, separate validation of the algorithms to monitor both the picked object and the target box was carried out.

5.3.5.1. Picked part monitoring

The main objective of the experimentation with the algorithm for monitoring the grabbed piece was (1) to select the appropriate parameters for the calculation of the dimensions of the collected object and (2) to estimate the error made in the

estimation. In this case, to calculate the accuracy of the dimensions estimated by the algorithm, a comparison was made with the measured dimensions of the object.

As expected, the quality of the 3D information provided by the *RealSense-D435* cameras was not comparable to industrial cameras such as the *Photoneo Phoxi L* which were used both to detect grasping points and to monitor the destination bin. Indeed, they were very sensitive to ambient light conditions. Therefore, the selection of the parameters was done experimentally and taking into account the lighting conditions of the real environment, and these can be seen in the Table 5.4.

Picked part monitoring	
V_b	$2\ mm$
F_r	$10\ mm$
F_n	5
M_d	$600 \times 600 \times 300\ mm$
N	64

Tab. 5.4.: Tuning parameters for the picked part monitoring algorithm.

To measure the error committed by the algorithm, 10 representative objects with different materials and shapes were selected. Then, after the robot picked up an object and it was at the monitoring station, the dimensions of the objects were measured. Furthermore, keeping the robot static in the monitoring volume, the algorithm was run 10 times (as the point cloud of the scene slightly changed in each acquisition), and the average dimensions estimated by the algorithm were calculated.

Table 5.5 shows the average errors made for each axis comparing the real dimensions measured by hand and those estimated by the algorithm. In our case, the objects were placed approximately 50 cm away from each of the cameras. According to the specifications of the camera model used, the error in depth estimation could be up to 2% in relation to the distance to the objects. Although there were other potential sources of error such as camera calibration, reflections caused by shiny materials, transparent objects etc. we concluded that the measurements made were accurate enough and in accordance with the specification of the cameras.

5.3.5.2. Destination bin monitoring

The main objective of the experimentation with the destination bin monitoring algorithm was to find the optimal parameter combination which offered robust

Item	Measured (cm)			Estimated (cm)			Error (cm)		
	x	y	z	$\overline{x}$	$\overline{y}$	$\overline{z}$	x	y	z
1	10	12	6	10.5	13.8	6	0.5	1.8	0
2	14	12.5	12	16.3	14.6	13.5	2.3	2.1	1.5
3	20.5	20.5	16.5	22.2	21.8	18.1	1.7	1.3	1.6
4	39	18.5	6	40.3	18.7	7.6	1.3	0.2	1.6
5	14.6	23	14	14.1	22	12	0.5	1	2
6	32	20	24.5	31.5	19.7	23.2	0.5	0.3	1.3
7	16.5	21.5	10	17.3	22.1	9.4	0.8	0.6	0.6
8	10	6.5	5	11.6	7.8	5.7	1.6	1.3	0.7
9	17	18	13	17	17.2	13.9	0	0.8	0.9
10	35.5	11	13	37.5	12	13.5	2	1	0.5
$\overline{x}$							1.12	1.04	1.07

Tab. 5.5.: Errors in the bounding box estimation during the picked part monitorisation.

behaviour in environments with high variability. As the algorithm relies on 3D information from a bin packing scene to calculate the EMSs, in which objects of all shapes and sizes can exist, the number of possible solutions is infinite and cannot be calculated in an optimal way. In our case, we use 3 criteria with the same level of importance which we seek to optimise when estimating the EMSs, and were the following:

- Maximise the total usable volume (the sum of the volumes of all the estimated EMSs that represent usable free space). To do so, we measured the ratio of estimated usable space to total space, which told us what percentage of the actual free space was estimated as usable space.

- Minimise the number of EMSs that represent the usable space inside the box.

- Minimise the algorithm's execution time (i.e. given a point cloud of the scene the time needed to estimate the EMSs), which was vital to guarantee the application's cycle time.

To that end, the algorithm was evaluated with a set of 168 scenes taken from the dataset published in [11], composed of scenes with random number, type and placement of objects. The localisation of the bin in the scene was also available. To carry out the search for the optimal parameters, we opted for an grid search

approach, and the possible values for each variable were experimentally selected. Those can be found in Equation 5.1.

$$\begin{aligned} V_s &= [1, 5, 10]\ mm \qquad & H &= [False, \frac{10}{V_s}, \frac{30}{V_s}, \frac{50}{V_s}] \\ Q &= [False, \frac{10}{V_s}, \frac{50}{V_s}, \frac{100}{V_s}] & J &= [False, True] \end{aligned} \tag{5.1}$$

In all the tests both the clustering-based and the statistical outlier filters were enabled to remove the noise from the point cloud scenes. Thus, per each scene 96 combinations were tested, which gave a total of $96 \cdot 168 = 16128$ trials.

For each of the 96 parameter combinations, the across-scene average values of the previously defined metrics were estimated. Since we wanted to optimise the parameters according to three different criteria, we tried to find a balance between them. Figure 5.10 represents the average values of all the combinations available in the grid search process. In red, the non-dominated combinations belonging to the Pareto front are represented, which are considered equally good solutions. In contrast, the combinations shown in blue are dominated solutions that do not belong to the Pareto front. In our case, we selected the combination of parameters represented with a green star in the Figure 5.10 among the non-dominated candidate solutions. The selected parameter combination and the estimated mean scores for each criterion are shown in Table 5.6.

Selected parameters	
V_s	$5\ mm$
H	2
Q	20
J	$True$
Estimated mean scores	
Usable volume ratio	83.67%
Execution time	$0.074\ s$
EMSs	26.9

Tab. 5.6.: Tuning parameters and estimated mean scores for the destination bin monitoring heuristic.

Finally, in order to have a clear view of the behaviour of the algorithm with the selected set of parameters, we evaluated it on scenes with different degrees of complexity. Specifically, we divided the set of scenes into three groups: Scenes with

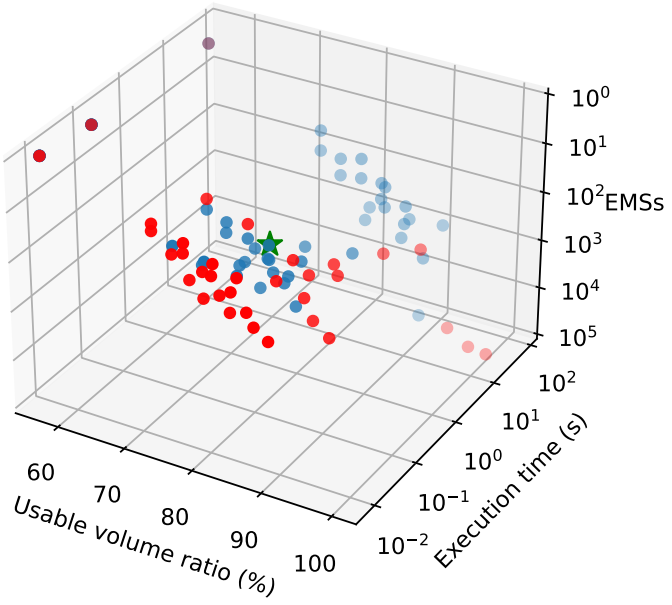
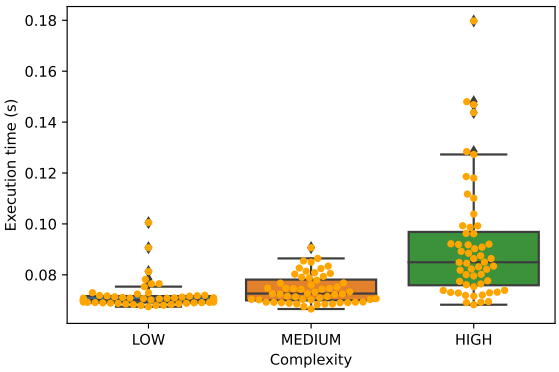


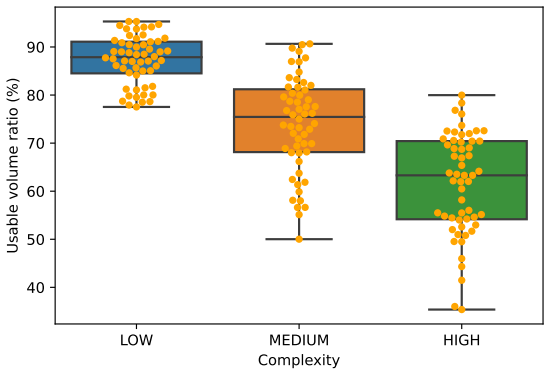
Fig. 5.10.: The scores of the 96 parameter combinations considering the average usable volume ratio, average execution time and the average number of generated EMSs. In red the non-dominated parameter combinations that belong to the Pareto front. The green star represents the scores of the selected parameter combination.

less than 10 objects were categorised as of "low complexity". Scenes with 10 to 20 objects were classified as of "medium complexity" and, finally, scenes with more than 20 objects were categorised as of "high complexity". Figure 5.11 depicts the aforementioned metrics for each scene complexity group.

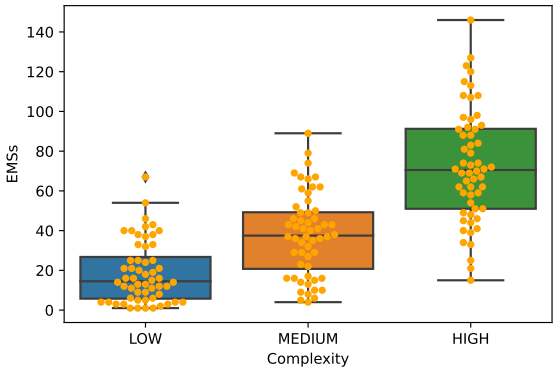
As for the execution times depicted in Figure 5.11a, although we can see that on average there was a slight increase as the complexity of the scene soared, it can be seen that between the best case of "low complexity" and the worst case of "high complexity" there was an increase of approximately three times. The same occurred within the scenes of "high complexity" that although there was not much variation between the number of objects, the execution time did vary. This occurred because the computational cost is not directly related to the number of objects in the scene but to the regularity of the scene. Although the scene itself has no effect on the processing steps performed at occupancy grid level, the more irregular the scene, the higher is the number of EMSs generated, as it can be seen in Figure 5.11c. An



(a) Execution times.



(b) Usable volume ratios.



(c) Generated EMSs.

Fig. 5.11.: Execution times, usable volume ratios and generated EMSs with respect to scenes with low, medium and high complexity.

increase in the number of EMSs generated has an effect on the solution joining phase where adjacent solutions are exhaustively searched for and joined together, consequently causing an increase in execution time. As can be guessed in Figure

5.11b, scenes with more objects tend to be more irregular, which also affects the proportion of usable volume. This happens because there are many unusable gaps in the irregular scenes, which are removed in the stabilisation and square generation phases.

5.3.6 System validation in the real robotic environment

The validation of the system in a real robotic environment consisted of 7 experiments of increasing difficulty, and the goal was to demonstrate the usefulness and flexibility offered by the packing system. The experiments were divided into three groups, from easy to difficult, in which, for simplicity, a single input and output box was used.

The main objective of these tests was to demonstrate the usability and flexibility of the system to generate mosaics with various types of objects with a large variability and without having to adjust it for each case. The tests carried out were the following:

- **Easy:**

  1. *Test 1:* Mono-reference boxes were placed in the inbound box, all of them with the same orientation.

  2. *Test 2:* Multi-reference boxes were manually thrown inside the inbound box, ordered from big to small considering their volume.

- **Medium:**

  1. *Test 3:* Multi-reference boxes were mixed in the inbound box, all of them in the most stable orientation. Once the algorithm was not able to pack more objects in that orientation, the orientation of the remaining objects was manually changed.

  2. *Test 4:* Multi-reference cans and bottles were placed in the inbound box, ordered from big to small considering their volume.

- **Complex:**

1. *Test 5:* Multi-reference boxes were placed in the inbound box, with random order and orientation.

2. *Test 6:* A reference of boxes and another reference of bottles were mixed in the inbound box.

3. *Test 7:* Random objects were randomly thrown to the inbound box.

The tests were conducted in the following way:

1. The number of objects for each test was selected based on the requirements of the test and the availability of the type of objects required.

2. All the items were put in the inbound box at the beginning of the test except in Test 2 and Test 4, in which the items were manually positioned (ordered from big to small).

3. At each iteration, the grasping point detection module decided the next object to pick.

4. The objects were packed in the same order that had been picked. We did not perform any object identification as a single destination bin was used and all the picked items were packed in the same bin.

5. The test ended when there were not objects left in the input bin, when the system was not able to pack the picked item or when the robot could not to pick more objects.

We used the optimal tuning parameters obtained during the experimentation detailed in Section 5.3.5. Since there was not any similar online bin-packing system publicly available, we recorded the performed tests to demonstrate the usefulness of the system. The recorded videos are publicly available in the following link: `https://bit.ly/3aWc0ke`.

In the first test we can see that the system is able to successfully pack a series of mono reference boxes that come with the same orientation. Although some objects slightly move/rotate when placed, we can see that these movements are taken into account in each iteration thanks to the monitoring of the target container. Based on that monitoring the system decides where to place the next object. Although the

dynamics of the automatic packing system are successfully taken into account, it introduces uncertainty in the packing process which causes the separation between objects to be large.

In the second and third tests it can be seen how the system behaves with multi-reference boxes. In both cases the packing order has a big impact on the final result. In addition, it can be seen that when packing certain objects they sometimes touch nearby objects. This happens due to the error committed in the dimension estimates of the object picked up, since those sometimes are smaller than the real dimensions. Although this causes the movement of nearby objects inside the box, the system reacts accordingly when packing the next object.

Although the system generates dynamically stable mosaics using multi-reference boxes placed in their most stable pose, this is not the case when the orientation of the boxes is random (test 5). In this case, the generated mosaic is very unstable, which causes some objects to fall when they are placed on top of other unstable objects. Although the system is able to consider such movements in the target box, the quality of the final mosaic decreases considerably.

Similarly to test 2, in test 4 we try to see the behaviour of the system also when the objects are ordered from bigger to smaller, but this time using cans and bottles. As can be seen the system works correctly. Although some small friction occur when placing some object that modify the final pose of the object to be left, the system takes it into account correctly in the following iteration. As can be guessed, it is key that they are ordered from larger to smaller so that the mosaic is stable, also with this type of objects.

In test 6, a box and a bottle reference are mixed in the inbound box, and the packing order is determined by the order in which the objects are picked. As can be seen, when objects of different sizes are packed, the generated mosaic is very irregular, which causes it to be unstable. This leads to some dropping after placing an object, which makes it even more difficult to generate a stable mosaic. Although the separation between objects and in general the use of space could be improved, it is clear that the strong point of the system is the ability to adapt to unforeseen situations and to plan online taking into account these uncertainties.

Finally, test 7 demonstrates the ability of the system to pack a wide variety of objects without the need to configure the system for each reference. It can be seen that the system shows great flexibility and is able to dynamically pack objects taken in a

random order. Of course, with such a variety of objects which are totally unknown to the system until the moment they are picked, it is practically impossible to build a stable and space-efficient mosaic.

5.4 Innovation

The packing of objects is a topic that has generated a lot of interest and has been in constant evolution. Due to the growing e-commerce, orders are becoming more and more customised to the customer and therefore logistics centres and department stores are becoming less and less specialised in specific objects and need to be able to handle a large variety of references in a flexible way. As shown in the state-of-the-art analysis in Section 5.1, there is currently work that has attempted to develop flexible packing systems capable of packing both regular and irregular objects. However, the major limitations of these works are the following:

- They assume that the packing process is ideal and ignore the uncertainties introduced by the use of irregular objects. This could lead to collisions with other items in the bin.

- They ignore the existence of an automatic packing system which restricts the possible drop poses of the piece, and may introduce an error in the drop.

On the one hand, the packing system developed in this work uses a robot as an automatic packing system, which is able to pick up a wide variety of objects without having to be configured for specific references. On the other hand, due to the variety of objects that the system is able to handle, the probability that after packing an object it will move or fall down is quite high. Consequently, the developed system first monitors the picked object to calculate its pose and dimensions, and then monitors the destination container to take into account the actual state of the packaging and calculate the drop-off pose accordingly. Finally, a dynamic planner has been developed which, receiving the output of both monitoring, estimates the best packing pose for the picked object taking into account criteria such as space utilisation or compactness.

The developed dynamic mosaic planner was finally integrated with an automatic picking system and validated in a real robotic scenario. To the best of our knowledge,

this was the first time that a complete online picking and packing robotic system was developed that allowed creating mosaics with arbitrary objects and considering the dynamics of a real robotic packing system.

5.5 Conclusions and future work

The results obtained in the real environment show that the developed system allows creating mosaics with arbitrary objects and considering the dynamics of a real robotic packing system. In fact, to our knowledge, this is the first time that a complete bin-picking and packing system is deployed in a real robotic system that is able to dynamically handle the uncertainties introduced by an automatic picking/packing system such as a robot.

In the recorded videos it can be seen that the system takes into account the uncertainties occurring inside the drop-off box, and makes estimations based on 3D data of the actual status of the box. Indeed, due to the fact that the scene is monitored whenever a new object is packed, the system successfully handles the dynamics of a real packing process, and thus, avoids collisions. Therefore we can answer the research question RQ4 in the affirmative. However, several limitations of the system were identified which can be improved in the future, and they are the following:

- Picking order and product orientation: The flexible picking system abstracts from product identification and aims to detect at least one grasping point in each iteration, with the ultimate goal of emptying the box. This means that the picking order is determined by the picking system, and it can vary in each iteration if the sensory information on which it bases its predictions varies. As expected, the order in which objects were picked had a major impact on the packing, especially with multi-reference objects. The orientation of the picked product also had a big impact on the quality of the final mosaic. In random multi-reference bin picking applications, the objects are usually cluttered inside the box, and this limits the way the robot can pick them. Therefore, the objects may not be in their most stable position and therefore, unless re-grasping is performed, the way the robot picks the objects considerably limits the possible drop-off poses.

- Inaccurate bounding box estimation: A set of 3 *RealSense D435* extrinsically calibrated cameras are used to monitor the gripped object. Compared to the industrial *Photoneo Phoxi L* cameras which are used both for detecting grasping points and for monitoring the target box, the quality of the point cloud is considerably less precise. This made the estimation of the dimensions of the objects generally larger, which consequently soared the inaccuracy when placing the objects in the mosaic, also causing an increase in the spacing between the objects.

- Occlusions: Based on the current configuration of the target box, the dynamic mosaic scheduler decides at each iteration which is the optimal drop pose for the picked object. Tests showed that in certain cases it has a tendency to generate columns of objects and this contributed to increase occlusions.

- External lighting: Changing lighting conditions have a big effect on 3D depth cameras. Although the validation pilot was covered with a housing and some noise filtering algorithms were used, this caused an increase in sensor noise. This led to inaccurate depth estimates, particularly with bright objects, which could cause collisions in the target box.

One of the main improvements needed relates to space utilisation. There were many factors that influenced it, and most of them could be improved in the future. However, there were other issues, such as the picking order of the products, which directly affected both mosaic quality and space utilisation, but which are difficult to improve in a multi-reference random bin-picking configuration. The gripping system has the ability to decide to some extent the order of picking, but this is limited to visible and graspable objects in the input bin.

In this work only a suction tool was considered for picking and placing the items, however it would be interesting to consider also other tools such as grippers. The gripper, compared to the suction, needs at least two contact points to handle the objects and this would introduce more complexity into the system. In addition, these contact points are usually located on the perimeter of the parts and this complicates the packing of objects close to each other.

The online mosaic planner plays a fundamental role in the mosaic generation process, as it is in charge of deciding the final position of each collected object. This work was conditioned by the dynamic planner of the UHS end user. Although in most cases it provided consistent results, it was sometimes difficult to understand why some

decisions were made by the planner. Therefore, the use of an open source planner would improve the traceability of the system. As far as we know, there is no planner with the requirements of our system, which would require the implementation of a new planner from scratch.

5.6 Publications

This line of research resulted in the following publication:

- Ander Iriondo et al. “Dynamic mosaic planning for a robotic bin-packing system based on picked part and target box monitoring”. In: *The International Journal of Advanced Manufacturing Technology* (2023), pp. 1–21

This article is attached in the PART II of the work.

6 Conclusions

This section briefly summarises the main conclusions drawn in relation to the questions posed during the 4 years of research.

RQ1: Can DRL be used to model a mobile manipulation behaviour such as picking?

The results obtained in the training process carried out in simulation, detailed in Chapter 3, showed that it is possible to apply DRL to learn a mobile manipulation behaviour only being guided by a reward signal. The success rates obtained during validation were high enough to confirm the feasibility of the technology. Despite the good results, the main weakness of the system was the fragility shown by the algorithms with respect to the selected agents, hyperparameters and reward functions used. This makes the behaviours learned in specific environments not generalisable to other environments, which makes the solution not scalable. Therefore, each environment needs the algorithm to be tuned in the best way to solve the problem and that is why the learned policies are not reproducible.

RQ2: If so, is the behaviour learned in simulation transferable to the real robot?

Having demonstrated the feasibility of applying DRL to learn a mobile manipulation behaviour without the need for explicit programming, the next objective was to validate it in the real environment. On the one hand, the development of a much more realistic simulated environment made it possible to reduce the simulation-to-reality gap, which in turn facilitated deployment to the real robot. On the other hand, the improvements introduced in the agent, the definition of the state representation and the reward function helped to achieve a more stable and robust behaviour in simulation. Finally, the controller learned in simulation was validated in the real environment, and the results obtained, also detailed in Chapter 3, were good enough to demonstrate the validity of the developed system.

RQ3: Does the use of n-dimensional point clouds help to predict the affordance of objects in a random bin-picking problem for both suction and two-finger end effectors?

In Chapter 4 we proposed to use GCNs to learn directly about n-dimensional point clouds instead of using 2D methods such as FCNs. Our intuition was that using higher dimensional data such as n-dimensional point clouds instead of images would improve the affordance-based detection of grasping points for suction and gripper. Although methods based on RGB-D images have been shown to be robust in detecting affordances in objects with a high degree of variability, it is unclear to what extent they take advantage of 3D depth information of the scene.

In the case of the suction tool, the results obtained with both known and unknown objects indicate that our GCN-based method substantially improves the accuracy of predictions compared to the 2D FCN-based method. Thanks to data augmentation applied directly on point clouds, and the use of normal vectors as additional features, a robust model with high generalisability to new objects was learned.

In the case of the two-finger gripper, although the results were better using the GCN-based method when dealing with known objects, this was not the case for scenes composed of unseen objects. In this case the 2D model showed higher flexibility, since its ability to generalise to new objects proved to be higher than using the point cloud-based method. In contrast to the suction model, performing data augmentation was not trivial for the gripper as the rotation of the point cloud had implicit information of the rotation of the gripper.

RQ4: Are we able to develop a flexible bin-packing robotic system that handles the uncertainties introduced by an automatic packing system and packs arbitrary objects?

Chapter 5 details the developed dynamic packing system capable of taking into account the dynamics of industrial environments such as warehouses or logistics centres to pack a wide variety of objects. The system was developed in a fully modular way and keeping in mind that the system should handle any kind of object. On the one hand, the monitoring of the picked object allowed us to know the dimensions and orientation of the picked object, regardless of its reference. On the other hand, the module for monitoring the target box played a key role as it allowed us to analyse the actual state of the box at each iteration, in order to dynamically calculate the next object and take into account any movements or drops that might

have occurred. Finally, the Dynamic IK-PAL planner was in charge of deciding the final drop pose taking into account both the monitoring of the caught object and criteria such as the degree of filling or compactness.

In the experimentation carried out, the system showed great flexibility in handling objects with a high degree of variability. In addition, the developed system takes into account the variations that occur in the mosaic being constructed to decide the pose of the next object. However, despite the flexibility offered by the system, there are several factors that somewhat penalise its efficiency. Due to the use of a flexible grasping point detection system, the objects are not known until the robot picks them up, and therefore, the calculation of the drop position has to be done online. Furthermore, it is the picking system that determines the order in which the objects are picked, and as seen in the experimentation, this has a large effect on the resulting mosaic. On the other hand, both the error introduced in the estimation of the object dimensions and the dynamics of the automatic packing system itself cause the spacing between the objects to be sub-optimal, and therefore, also negatively affects the quality of the final mosaic.

6.1 Future works

This work has attempted to take a step forward in advanced flexible robotics using AI and focusing on three specific use cases related to pick and place. Despite the contributions made, several limitations have been identified in the developed technologies that must be tackled in the future in order to increase the systems' robustness before deploying them in the industry.

In the first use case, DRL was used to learn a positioning policy for a mobile manipulator in order to successfully carry out a picking operation. On the one hand, the first future step is to add sensory information (e.g. laser readings, images etc.) to the learning process in order to learn a more intelligent controller with a better perception of the environment. On the other hand, although the learned behaviours are promising, they are not robust enough to be translated to an industrial application. The gap between simulation and reality means that simulation-trained controllers do not perform in the real environment as robustly as in the virtual world. Therefore, the future work will be focused on increasing the degree of realism of simulated environments (considering both visual and physical aspects), and on

applying advanced techniques to reduce the sim-to-real gap using techniques such as domain randomisation.

In the second use case, the Deep GCN DL-based architecture was used to predict grasping points in a generic way on point clouds. Although the results obtained were satisfactory especially in the case of the suction model, several improvement points were identified on which future work will focus. The first line of work will seek to reduce the time cost of generating a labelled dataset of grasping zones. Because manual labelling of the data is very costly, two main ways will be explored to speed up this task:

- Use of DL-based models that do not need many labelled examples to be trained.
- Development of semi-automated techniques to speed up data collection and labelling.

The second point of improvement concerns the computational cost and scalability of the solution, especially of the two-finger gripper model. The computational cost of performing an inference for both suction and gripper is considerably higher than image-based 2D models. As for suction only a single inference is performed for the whole scene, this is not a problem to meet the cycle times. For the gripper, however, a series of inferences are made to estimate the best gripper orientation, and this increases the inference time considerably. Therefore, the next step is to develop a more scalable approach for the gripper that also allows working with 6-DoF, as opposed to the 3-DoF of the current solution.

In the third and final use case, an automatic dynamic packing system for arbitrary objects was developed using a robot equipped with a suction end effector. In the mosaics generated by the system, it could be seen that a major point of improvement was the space utilisation within the container where the mosaic was generated. This was caused by several factors, and although some cannot be improved in a multi-reference to multi-reference palletising application (e.g. the picking order of the products, occlusions or variable lighting), the following improvements can be introduced to the system in the future:

- Use more accurate sensors to improve the estimation of the dimensions and the orientation of the picked item.
- Develop a system to push objects against walls in order to eliminate spaces.

Another of the most important improvements is related to the UHS' dynamic planner, as it sometimes makes unexpected decisions which have a great effect on the final result of the mosaic. Therefore, it will be key to develop a new open source planner that, taking as input the monitoring of both the caught object and the drop box, will heuristically estimate the best drop pose for the picked object. This will help to improve the traceability of the system. Finally, consideration will be given to extending the system to grippers other than suction. Although this will introduce complexity to the system, it will increase the flexibility of the system as it will be able to package a wider variety of objects.

Generally speaking, although current AI-based technologies allow for increased flexibility of robotic applications, there are still several loose ends which make these solutions not directly applicable in industrial applications. Therefore, the work on flexible robotic manipulation in the coming years may be focused on tying up these loose ends:

- The first major limitation is the poor explainability of certain current AI technologies such as DL or DRL. Although it has proven to be the technology that offers the best generalisation ability on high-dimensional data such as images or point clouds, the learned models are black boxes and it is not at all obvious to understand the causes of the decisions made by such models. In robotics, these neural network-based models are generally used for both perception and control systems, and the decisions made by these models have a direct impact on the performance of the robotic systems. Indeed, an unexpected decision can compromise the safety of the system. Therefore, it is essential to improve the explainability of such models in order to understand their behaviour in different cases and thus guarantee the correct functioning of the models.

- The second limitation is the large amount of labelled data needed to train robust deep neural networks. In DL-based perception applications, both data collection and labelling are usually carried out offline. In this case the most expensive process is labelling, which is usually carried out manually due to the difficulty of automating such a process where human judgement is vital. This costly process is impractical for the industry and should be somehow alleviated. That said, there are two avenues on which much work is currently being done but there is still much room for improvement: developing models that require a smaller number of labelled images, or developing semi-automated systems to speed up the labelling process. In DRL-based control applications, on

the other hand, there is no labelling process as a reward function is used to evaluate the quality of the controller at each instant. However, the training experience is usually collected online, which is extremely time-consuming and materially expensive to do it with real robots. Therefore, training processes are carried out in simulated environments. Although it is now possible to create simulated environments with a high degree of realism, there is still a gap between simulation and reality, which makes it difficult to learn robust behaviours. It is therefore interesting to explore ways to generate even more realistic simulations, as well as other methods to reduce the sim-to-real gap such as domain randomisation.

Bibliography

[1] Heiner Lasi, Peter Fettke, Hans-Georg Kemper, Thomas Feld, and Michael Hoffmann. "Industry 4.0". In: *Business & information systems engineering* 6.4 (2014), pp. 239–242 (cit. on p. 3).

[2] Saeid Nahavandi. "Industry 5.0—A human-centric solution". In: *Sustainability* 11.16 (2019), p. 4371 (cit. on p. 3).

[3] Iñaki Maurtua Ormaetxea. "Estrategias y tecnologías para la colaboración segura entre personas y robots en entornos industriales". PhD thesis. 2017 (cit. on p. 4).

[4] Domenico Prattichizzo and Jeffrey C Trinkle. "Grasping". In: *Springer handbook of robotics*. Springer, 2016, pp. 955–988 (cit. on p. 4).

[5] Jens Kober and Jan Peters. "Imitation and reinforcement learning". In: *Robotics & Automation Magazine* 17.2 (2010), pp. 55–62 (cit. on p. 5).

[6] Daron Acemoglu and Pascual Restrepo. "Robots and jobs: Evidence from US labor markets". In: *Journal of Political Economy* 128.6 (2020), pp. 2188–2244 (cit. on p. 5).

[7] Sara Bragança, Eric Costa, Ignacio Castellucci, and Pedro M Arezes. "A brief overview of the use of collaborative robots in industry 4.0: human role and safety". In: *Occupational and Environmental Safety and Health* (2019), pp. 641–650 (cit. on p. 5).

[8] Pat Langley and Herbert A Simon. "Applications of machine learning and rule induction". In: *Communications of the ACM* 38.11 (1995), pp. 54–64 (cit. on p. 17).

[9] Jurgen Kreuziger. "Application of machine learning to robotics-an analysis". In: *In Proceedings of the Second International Conference on Automation, Robotics, and Computer Vision (ICARCV)*. Citeseer. 1992 (cit. on p. 17).

[10] Ander Iriondo, Elena Lazkano, Loreto Susperregi, et al. "Pick and place operations in logistics using a mobile manipulator controlled with deep reinforcement learning". In: *Applied Sciences* 9.2 (2019), p. 348 (cit. on pp. 18, 65).

[11] Ander Iriondo Azpiri, Elena Lazkano Ortega, and Ander Ansuategi Cobo. "Affordance-Based Grasping Point Detection Using Graph Convolutional Networks for Industrial Bin-Picking Applications". In: *Sensors* 21.3 (2021), p. 816 (cit. on pp. 18, 96, 118).

[12] Ander Iriondo, Elena Lazkano, Ander Ansuategi, Ane Fernandez, and Iñaki Maurtua. "Dynamic mosaic planning for a robotic bin-packing system based on picked part and target box monitoring". In: *The International Journal of Advanced Manufacturing Technology* (2023), pp. 1–21 (cit. on pp. 18, 128).

[13]Ander Iriondo, Elena Lazkano, Ander Ansuategi, et al. “Learning positioning policies for mobile manipulation operations with deep reinforcement learning”. In: *International Journal of Machine Learning and Cybernetics* (2023) (cit. on pp. 19, 65).

[14]Loreto Susperregi, Ane Fernandez, Jorge Molina, et al. “RSAII: Flexible Robotized Unitary Picking in Collaborative Environments for Order Preparation in Distribution Centers”. In: *Bringing Innovative Robotic Technologies from Research Labs to Industrial End-users*. Springer, 2020, pp. 129–151 (cit. on pp. 19, 80).

[15]Thushara Sandakalum and Marcelo H Ang Jr. “Motion planning for mobile manipulators—a systematic review”. In: *Machines* 10.2 (2022), p. 97 (cit. on p. 23).

[16]Freek Stulp, Andreas Fedrizzi, Lorenz Mösenlechner, and Michael Beetz. “Learning and reasoning with action-related places for robust mobile manipulation”. In: *Journal of Artificial Intelligence Research* 43 (2012), pp. 1–42 (cit. on pp. 24, 25).

[17]Eitan Marder-Eppstein, Eric Berger, Tully Foote, Brian Gerkey, and Kurt Konolige. “The office marathon: Robust navigation in an indoor office environment”. In: *International conference on robotics and automation*. IEEE. 2010, pp. 300–307 (cit. on p. 24).

[18]David Coleman, Ioan Sucan, Sachin Chitta, and Nikolaus Correll. “Reducing the barrier to entry of complex robotic software: a moveit! case study”. In: *Journal of Software Engineering for Robotics* 5.1 (2014), pp. 3–16 (cit. on p. 24).

[19]Bruno Siciliano and Oussama Khatib. *Springer handbook of robotics*. Springer, 2016 (cit. on p. 24).

[20]Andreas Dömel, Simon Kriegel, Michael Kaßecker, et al. “Toward fully autonomous mobile manipulation for industrial environments”. In: *International Journal of Advanced Robotic Systems* 14.4 (2017), p. 1729881417718588 (cit. on pp. 24, 34).

[21]Jingren Xu, Kensuke Harada, Weiwei Wan, Toshio Ueshiba, and Yukiyasu Domae. “Planning an efficient and robust base sequence for a mobile manipulator performing multiple pick-and-place tasks”. In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 11018–11024 (cit. on p. 24).

[22]Vincent Padois, J-Y Fourquet, and Pascale Chiron. “From robotic arms to mobile manipulation: On coordinated motion schemes”. In: *Intelligent Production Machines and Systems*. Elsevier, 2006, pp. 572–577 (cit. on p. 24).

[23]Jindong Tan, Ning Xi, and Yuechao Wang. “Integrated task planning and control for mobile manipulators”. In: *The International Journal of Robotics Research* 22.5 (2003), pp. 337–354 (cit. on p. 24).

[24]Karl Berntorp, Karl-Erik Årzén, and Anders Robertsson. “Mobile manipulation with a kinematically redundant manipulator for a pick-and-place scenario”. In: *International Conference on Control Applications*. IEEE. 2012, pp. 1596–1602 (cit. on p. 24).

[25] Wim Meeussen, Melonee Wise, Stuart Glaser, et al. “Autonomous door opening and plugging in with a personal robot”. In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2010, pp. 729–736 (cit. on p. 25).

[26] Aitor Ibarguren and Paul Daelman. “Path Driven Dual Arm Mobile Co-Manipulation Architecture for Large Part Manipulation in Industrial Environments”. In: *Sensors* 21.19 (2021), p. 6620 (cit. on p. 25).

[27] Daniel Kappler, Peter Pastor, Mrinal Kalakrishnan, Manuel Wüthrich, and Stefan Schaal. “Data-Driven Online Decision Making for Autonomous Manipulation.” In: *Robotics: science and systems*. Vol. 11. 2015 (cit. on p. 25).

[28] Sheng Lin and Andrew A Goldenberg. “Neural-network control of mobile manipulators”. In: *IEEE Transactions on Neural Networks* 12.5 (2001), pp. 1121–1133 (cit. on p. 25).

[29] George Konidaris, Scott Kuindersma, Roderic Grupen, and Andrew Barto. “Autonomous skill acquisition on a mobile manipulator”. In: *Twenty-Fifth AAAI Conference on Artificial Intelligence*. 2011 (cit. on p. 25).

[30] Julian Ibarz, Jie Tan, Chelsea Finn, et al. “How to train your robot with deep reinforcement learning: lessons we have learned”. In: *The International Journal of Robotics Research* 40.4-5 (2021), pp. 698–721 (cit. on p. 26).

[31] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. “Deep reinforcement learning: A brief survey”. In: *IEEE Signal Processing Magazine* 34.6 (2017), pp. 26–38 (cit. on p. 26).

[32] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. “Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates”. In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 3389–3396 (cit. on p. 26).

[33] Marwan Qaid Mohammed, Kwek Lee Chung, and Chua Shing Chyi. “Review of Deep Reinforcement Learning-based Object Grasping: Techniques, Open Challenges and Recommendations”. In: *IEEE Access* (2020) (cit. on p. 26).

[34] Johanna Hansen, Francois Hogan, Dmitriy Rivkin, et al. “Visuotactile-RL: Learning Multimodal Manipulation Policies with Deep Reinforcement Learning”. In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 8298–8304 (cit. on p. 26).

[35] Kai Zhu and Tao Zhang. “Deep reinforcement learning based mobile robot navigation: A review”. In: *Tsinghua Science and Technology* 26.5 (2021), pp. 674–691 (cit. on p. 26).

[36] Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, et al. “Learning to walk via deep reinforcement learning”. In: *Robotics: Science and Systems* (2019) (cit. on pp. 26, 33).

[37] Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel Van De Panne. “Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning”. In: *Transactions on Graphics (TOG)* 36.4 (2017), pp. 1–13 (cit. on p. 26).

[38] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, et al. “Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation”. In: *Proceedings of The 2nd Conference on Robot Learning*. Ed. by Aude Billard, Anca Dragan, Jan Peters, and Jun Morimoto. Vol. 87. Proceedings of Machine Learning Research. PMLR, 29–31 Oct 2018, pp. 651–673 (cit. on pp. 26, 71).

[39] Rishabh Jangir, Guillem Alenyà, and Carme Torras. “Dynamic cloth manipulation with deep reinforcement learning”. In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 4630–4636 (cit. on p. 26).

[40] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, et al. “Continuous control with deep reinforcement learning”. In: *4th International Conference on Learning Representations (ICLR)*. 2016 (cit. on pp. 26, 33, 34).

[41] MyeongSeop Kim, Dong-Ki Han, Jae-Han Park, and Jung-Su Kim. “Motion planning of robot manipulators for a smoother path using a twin delayed deep deterministic policy gradient with hindsight experience replay”. In: *Applied Sciences* 10.2 (2020), p. 575 (cit. on p. 26).

[42] Scott Fujimoto, Herke Hoof, and David Meger. “Addressing function approximation error in actor-critic methods”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1587–1596 (cit. on pp. 26, 33).

[43] David Hsu, Jean-Claude Latombe, and Hanna Kurniawati. “On the probabilistic foundations of probabilistic roadmap planning”. In: *The International Journal of Robotics Research* 25.7 (2006), pp. 627–643 (cit. on p. 26).

[44] Lei Tai, Giuseppe Paolo, and Ming Liu. “Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation”. In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 31–36 (cit. on p. 26).

[45] Chao Wang, Jian Wang, Yuan Shen, and Xudong Zhang. “Autonomous navigation of UAVs in large-scale complex environments: A deep reinforcement learning approach”. In: *IEEE Transactions on Vehicular Technology* 68.3 (2019), pp. 2124–2136 (cit. on p. 26).

[46] Stephen Dankwa and Wenfeng Zheng. “Modeling a Continuous Locomotion Behavior of an Intelligent Agent Using Deep Reinforcement Technique.” In: *2nd International Conference on Computer and Communication Engineering Technology (CCET)*. 2019, pp. 172–175 (cit. on p. 26).

[47] PB Khoi, NT Giang, and HV Tan. “Control and Simulation of a 6-DOF Biped Robot based on Twin Delayed Deep Deterministic Policy Gradient Algorithm”. In: *Indian Journal of Science and Technology* 14.30 (2021), pp. 2460–2471 (cit. on p. 26).

[48] Julien Kindle, Fadri Furrer, Tonci Novkovic, et al. “Whole-Body Control of a Mobile Manipulator using End-to-End Reinforcement Learning”. In: *arXiv preprint arXiv:2003.02637* (2020) (cit. on p. 27).

[49] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. “Proximal Policy Optimization Algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017), pp. 1–12. arXiv: 1707.06347 (cit. on pp. 27, 33).

[50]Cong Wang, Qifeng Zhang, Qiyan Tian, et al. “Learning mobile manipulation through deep reinforcement learning”. In: *Sensors* 20.3 (2020), p. 939 (cit. on p. 27).

[51]Daniel Honerkamp, Tim Welschehold, and Abhinav Valada. “Learning kinematic feasibility for mobile manipulation through deep reinforcement learning”. In: *IEEE Robotics and Automation Letters* 6.4 (2021), pp. 6289–6296 (cit. on p. 27).

[52]Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. Vol. 1. 1. MIT press Cambridge, 1998 (cit. on p. 29).

[53]Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), p. 529 (cit. on pp. 31, 38).

[54]Marcin Andrychowicz, Filip Wolski, Alex Ray, et al. “Hindsight experience replay”. In: *Advances in neural information processing systems* 30 (2017) (cit. on p. 31).

[55]Will Dabney, Mark Rowland, Marc Bellemare, and Rémi Munos. “Distributional reinforcement learning with quantile regression”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018 (cit. on p. 31).

[56]Marc G Bellemare, Will Dabney, and Rémi Munos. “A distributional perspective on reinforcement learning”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 449–458 (cit. on p. 31).

[57]John Schulman, Sergey Levine, Philipp Moritz, Michael Jordan, and Pieter Abbeel. “Trust Region Policy Optimization”. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning*. Vol. 37. PMLR. 2015, pp. 1889–1897 (cit. on pp. 33, 37).

[58]Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, et al. “Asynchronous methods for deep reinforcement learning”. In: *International Conference on Machine Learning*. PMLR. 2016, pp. 1928–1937 (cit. on p. 33).

[59]Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al. “Playing atari with deep reinforcement learning”. In: *arXiv preprint arXiv:1312.5602* (2013) (cit. on p. 34).

[60]Josh Tobin, Rachel Fong, Alex Ray, et al. “Domain randomization for transferring deep neural networks from simulation to the real world”. In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 23–30 (cit. on p. 34).

[61]Morgan Quigley, Ken Conley, Brian Gerkey, et al. “ROS: an open-source Robot Operating System”. In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5 (cit. on p. 35).

[62]Nathan Koenig and Andrew Howard. “Design and use paradigms for gazebo, an open-source multi-robot simulator”. In: *International Conference on Intelligent Robots and Systems (IROS)*. Vol. 3. IEEE, pp. 2149–2154 (cit. on p. 35).

[63]Greg Brockman, Vicki Cheung, Ludwig Pettersson, et al. “Openai gym”. In: *arXiv preprint arXiv:1606.01540* (2016) (cit. on p. 37).

[64]Iker Zamora, Nestor Gonzalez Lopez, Víctor Mayoral Vilches, Alejandro Hernández Cordero, and Erle Robotics. “Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo”. In: *arXiv preprint arXiv:1608.05742* (2017) (cit. on p. 37).

[65]Antonin Raffin, Ashley Hill, Adam Gleave, et al. “Stable-Baselines3: Reliable Reinforcement Learning Implementations”. In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8 (cit. on p. 37).

[66]David Silver, Guy Lever, Nicolas Heess, et al. “Deterministic policy gradient algorithms”. In: *ICML*. 2014 (cit. on pp. 38, 48).

[67]Andrew Y Ng, Daishi Harada, and Stuart Russell. “Policy invariance under reward transformations: Theory and application to reward shaping”. In: *ICML*. Vol. 99. 1999, pp. 278–287 (cit. on p. 40).

[68]Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, et al. “Unity: A general platform for intelligent agents”. In: *arXiv preprint arXiv:1809.02627* (2018) (cit. on p. 44).

[69]Martin Bischof. *ROS-SHARP*. `https://github.com/siemens/ros-sharp`. [Accessed: 02-01-2023]. 2018 (cit. on p. 44).

[70]Richard S Sutton. “Learning to predict by the methods of temporal differences”. In: *Machine Learning* 3.1 (1988), pp. 9–44 (cit. on p. 48).

[71]Stephanie CY Chan, Sam Fishman, John Canny, Anoop Korattikara, and Sergio Guadarrama. “Measuring the Reliability of Reinforcement Learning Algorithms”. In: *International Conference on Learning Representations, Addis Ababa, Ethiopia*. 2020 (cit. on p. 55).

[72]Carlo Acerbi and Dirk Tasche. “Expected shortfall: a natural coherent alternative to value at risk”. In: *Economic notes* 31.2 (2002), pp. 379–388 (cit. on p. 55).

[73]Alexei Chekhlov, Stanislav Uryasev, and Michael Zabarankin. “Drawdown measure in portfolio optimization”. In: *International Journal of Theoretical and Applied Finance* 8.01 (2005), pp. 13–58 (cit. on p. 55).

[74]Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. “Monte carlo localization: Efficient position estimation for mobile robots”. In: *AAAI/IAAI* 343-349 (1999), pp. 2–2 (cit. on p. 57).

[75]Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. “Improved techniques for grid mapping with rao-blackwellized particle filters”. In: *IEEE Transactions on Robotics* 23.1 (2007), pp. 34–46 (cit. on p. 57).

[76]David G Lowe. “Distinctive image features from scale-invariant keypoints”. In: *International Journal of Computer Vision* 60.2 (2004), pp. 91–110 (cit. on p. 68).

[77]Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. “Surf: Speeded up robust features”. In: *European Conference on Computer Vision*. Springer. 2006, pp. 404–417 (cit. on p. 68).

[78] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. "ORB: An efficient alternative to SIFT or SURF". In: *International Conference on Computer Vision (ICCV)*. IEEE. 2011, pp. 2564–2571 (cit. on p. 68).

[79] Anis Sahbani, Sahar El-Khoury, and Philippe Bidaud. "An overview of 3D object grasp synthesis algorithms". In: *Robotics and Autonomous Systems* 60.3 (2012), pp. 326–336 (cit. on p. 68).

[80] Paul J Besl and Neil D McKay. "Method for registration of 3-D shapes". In: *Sensor fusion IV: control paradigms and data structures*. Vol. 1611. Spie. 1992, pp. 586–606 (cit. on p. 68).

[81] Kai-Tai Song, Cheng-Hei Wu, and Sin-Yi Jiang. "CAD-based pose estimation design for random bin picking using a RGB-D camera". In: *Journal of Intelligent & Robotic Systems* 87.3 (2017), pp. 455–470 (cit. on p. 68).

[82] Jeannette Bohg, Antonio Morales, Tamim Asfour, and Danica Kragic. "Data-driven grasp synthesis—a survey". In: *IEEE Transactions on Robotics* 30.2 (2013), pp. 289–309 (cit. on p. 68).

[83] Maryam M Najafabadi, Flavio Villanustre, Taghi M Khoshgoftaar, et al. "Deep learning applications and challenges in big data analytics". In: *Journal of Big Data* 2.1 (2015), p. 1 (cit. on p. 69).

[84] Jincheng Yu, Kaijian Weng, Guoyuan Liang, and Guanghan Xie. "A vision-based robotic grasping system using deep learning for 3D object recognition and pose estimation". In: *International Conference on Robotics and Biomimetics (ROBIO)*. IEEE. 2013, pp. 1175–1180 (cit. on p. 69).

[85] Ian Lenz, Honglak Lee, and Ashutosh Saxena. "Deep learning for detecting robotic grasps". In: *The International Journal of Robotics Research* 34.4-5 (2015), pp. 705–724 (cit. on p. 69).

[86] Sulabh Kumra and Christopher Kanan. "Robotic grasp detection using deep convolutional neural networks". In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 769–776 (cit. on p. 69).

[87] Yun Jiang, Stephen Moseson, and Ashutosh Saxena. "Efficient grasping from rgbd images: Learning using a new rectangle representation". In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2011, pp. 3304–3311 (cit. on p. 69).

[88] Xinwen Zhou, Xuguang Lan, Hanbo Zhang, et al. "Fully convolutional grasp detection network with oriented anchor box". In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 7223–7230 (cit. on p. 69).

[89] Renaud Detry, Dirk Kraft, Oliver Kroemer, et al. "Learning grasp affordance densities". In: *Paladyn, Journal of Behavioral Robotics* 2.1 (2011), pp. 1–17 (cit. on p. 70).

[90] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation". In: *Proceedings of the Conference on Computer Vision and Pattern Recognition*. IEEE. 2015, pp. 3431–3440 (cit. on p. 70).

[91] Andy Zeng, Shuran Song, Kuan-Ting Yu, et al. “Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching”. In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 1–8 (cit. on pp. 70, 74, 87, 93).

[92] Clemens Eppner, Arsalan Mousavian, and Dieter Fox. “A Billion Ways to Grasps - An Evaluation of Grasp Sampling Schemes on a Dense, Physics-based Grasp Data Set”. In: *Proceedings of the International Symposium on Robotics Research (ISRR)*. Hanoi, Vietnam, 2019 (cit. on pp. 70, 72).

[93] Andreas ten Pas, Marcus Gualtieri, Kate Saenko, and Robert Platt. “Grasp pose detection in point clouds”. In: *The International Journal of Robotics Research* 36.13-14 (2017), pp. 1455–1473 (cit. on p. 70).

[94] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, et al. “Dex-Net 2.0: Deep Learning to Plan Robust Grasps with Synthetic Point Clouds and Analytic Grasp Metrics”. In: *Robotics: Science and Systems (RSS)*. 2017 (cit. on pp. 70, 71).

[95] Jeffrey Mahler and Ken Goldberg. “Learning Deep Policies for Robot Bin Picking by Simulating Robust Grasping Sequences”. In: *Proceedings of the 1st Annual Conference on Robot Learning*. Vol. 78. Proceedings of Machine Learning Research. PMLR, 13–15 Nov 2017, pp. 515–524 (cit. on p. 71).

[96] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. “Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection”. In: *The International Journal of Robotics Research* 37.4-5 (2018), pp. 421–436 (cit. on p. 71).

[97] Stephen James, Andrew J. Davison, and Edward Johns. “Transferring End-to-End Visuomotor Control from Simulation to Real World for a Multi-Stage Task”. In: *Proceedings of the 1st Annual Conference on Robot Learning*. Vol. 78. PMLR, 13–15 Nov 2017, pp. 334–343 (cit. on p. 71).

[98] Jonathan Tremblay, Aayush Prakash, David Acuna, et al. “Training deep networks with synthetic data: Bridging the reality gap by domain randomization”. In: *Proceedings of the Conference on Computer Vision and Pattern Recognition Workshops*. IEEE. 2018, pp. 969–977 (cit. on p. 71).

[99] Zonghan Wu, Shirui Pan, Fengwen Chen, et al. “A comprehensive survey on graph neural networks”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2020), pp. 1–21 (cit. on pp. 71, 77).

[100] Abubakar Sulaiman Gezawa, Yan Zhang, Qicong Wang, and Lei Yunqi. “A Review on Deep Learning Approaches for 3D Data Representations in Retrieval and Classifications”. In: *IEEE Access* 8 (2020), pp. 57566–57593 (cit. on pp. 71, 74).

[101] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. “Pointnet: Deep learning on point sets for 3d classification and segmentation”. In: *Proceedings of the Conference on Computer Vision and Pattern Recognition*. IEEE. 2017, pp. 652–660 (cit. on p. 71).

[102] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. "Pointnet++: Deep hierarchical feature learning on point sets in a metric space". In: *Advances in neural information processing systems*. 2017, pp. 5099–5108 (cit. on p. 71).

[103] Hongzhuo Liang, Xiaojian Ma, Shuang Li, et al. "Pointnetgpd: Detecting grasp configurations from point sets". In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 3629–3635 (cit. on p. 72).

[104] Arsalan Mousavian, Clemens Eppner, and Dieter Fox. "6-dof graspnet: Variational grasp generation for object manipulation". In: *Proceedings of the International Conference on Computer Vision (ICCV)*. IEEE. 2019, pp. 2901–2910 (cit. on p. 72).

[105] Peiyuan Ni, Wenguang Zhang, Xiaoxiao Zhu, and Qixin Cao. "Pointnet++ grasping: learning an end-to-end spatial grasp generation algorithm from sparse point clouds". In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 3619–3625 (cit. on p. 72).

[106] Yuzhe Qin, Rui Chen, Hao Zhu, et al. "S4g: Amodal single-view single-shot se (3) grasp detection in cluttered scenes". In: *Conference on robot learning*. PMLR. 2020, pp. 53–65 (cit. on p. 72).

[107] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. "Deepgcns: Can gcns go as deep as cnns?" In: *Proceedings of the International Conference on Computer Vision (ICCV)*. IEEE. 2019, pp. 9267–9276 (cit. on pp. 74, 77, 84, 93).

[108] Lars Linsen. *Point cloud representation*. Fakultät für Informatik, Universität Karlsruhe. 2001 (cit. on p. 74).

[109] Yulan Guo, Hanyun Wang, Qingyong Hu, et al. "Deep learning for 3d point clouds: A survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.12 (2020), pp. 4338–4364 (cit. on pp. 74, 75).

[110] Charu C Aggarwal, Haixun Wang, et al. *Managing and mining graph data*. Vol. 40. Springer, 2010 (cit. on p. 76).

[111] Will Hamilton, Zhitao Ying, and Jure Leskovec. "Inductive representation learning on large graphs". In: *Advances in neural information processing systems*. 2017, pp. 1024–1034 (cit. on p. 76).

[112] Yue Wang, Yongbin Sun, Ziwei Liu, et al. "Dynamic graph CNN for learning on point clouds". In: *Transactions On Graphics (TOG)* 38.5 (2019), pp. 1–12 (cit. on p. 76).

[113] Jie Zhou, Ganqu Cui, Shengding Hu, et al. "Graph neural networks: A review of methods and applications". In: *AI Open* 1 (2020), pp. 57–81 (cit. on pp. 77, 94).

[114] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. "DropEdge: Towards Deep Graph Convolutional Networks on Node Classification". In: *International Conference on Learning Representations*. 2020 (cit. on p. 77).

[115] Wei-Lin Chiang, Xuanqing Liu, Si Si, et al. "Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2019, pp. 257–266 (cit. on p. 77).

[116]Iro Armeni, Ozan Sener, Amir R Zamir, et al. “3d semantic parsing of large-scale indoor spaces”. In: *Proceedings of the Conference on Computer Vision and Pattern Recognition*. IEEE. 2016, pp. 1534–1543 (cit. on p. 77).

[117]Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep residual learning for image recognition”. In: *Proceedings of the Conference on Computer Vision and Pattern Recognition*. IEEE. 2016, pp. 770–778 (cit. on p. 77).

[118]Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. “Densely connected convolutional networks”. In: *Proceedings of the conference on computer vision and pattern recognition*. IEEE. 2017, pp. 4700–4708 (cit. on p. 77).

[119]Jeffrey Mahler, Matthew Matl, Vishal Satish, et al. “Learning ambidextrous robot grasping policies”. In: *Science Robotics* 4.26 (2019) (cit. on p. 79).

[120]Michael Danielczuk, Jeffrey Mahler, Chris Correa, and Ken Goldberg. “Linear push policies to increase grasp access for robot bin picking”. In: *14th International Conference on Automation Science and Engineering (CASE)*. IEEE. 2018, pp. 1249–1256 (cit. on p. 79).

[121]Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. “Graph convolutional networks: a comprehensive review”. In: *Computational Social Networks* 6.1 (2019), p. 11 (cit. on p. 94).

[122]Silvano Martello and Daniele Vigo. “Exact solution of the two-dimensional finite bin packing problem”. In: *Management Science* 44.3 (1998), pp. 388–399 (cit. on p. 98).

[123]Silvano Martello, David Pisinger, and Daniele Vigo. “The three-dimensional bin packing problem”. In: *Operations Research* 48.2 (2000), pp. 256–267 (cit. on p. 98).

[124]Edgar Den Boef, Jan Korst, Silvano Martello, David Pisinger, and Daniele Vigo. “Erratum to “the three-dimensional bin packing problem”: robot-packable and orthogonal variants of packing problems”. In: *Operations Research* 53.4 (2005), pp. 735–736 (cit. on p. 98).

[125]Stefan Jakobs. “On genetic algorithms for the packing of polygons”. In: *European Journal of Operational Research* 88.1 (1996), pp. 165–181 (cit. on p. 98).

[126]David S. Johnson, Alan Demers, Jeffrey D. Ullman, Michael R Garey, and Ronald L. Graham. “Worst-case performance bounds for simple one-dimensional packing algorithms”. In: *Journal on Computing* 3.4 (1974), pp. 299–325 (cit. on p. 98).

[127]Teodor Gabriel Crainic, Guido Perboli, and Roberto Tadei. “TS2PACK: A two-level tabu search for the three-dimensional bin packing problem”. In: *European Journal of Operational Research* 195.3 (2009), pp. 744–760 (cit. on p. 98).

[128]Oluf Faroe, David Pisinger, and Martin Zachariasen. “Guided local search for the three-dimensional bin-packing problem”. In: *Journal on Computing* 15.3 (2003), pp. 267–283 (cit. on p. 98).

[129]José Fernando Gonçalves and Mauricio GC Resende. “A biased random key genetic algorithm for 2D and 3D bin packing problems”. In: *International Journal of Production Economics* 145.2 (2013), pp. 500–510 (cit. on pp. 98, 110).

[130] Haoyuan Hu, Xiaodong Zhang, Xiaowei Yan, Longfei Wang, and Yinghui Xu. "Solving a new 3d bin packing problem with deep reinforcement learning method". In: *ArXiv* abs/1708.05930 (2017) (cit. on p. 98).

[131] Fan Wang and Kris Hauser. "Stable bin packing of non-convex 3D objects with a robot manipulator". In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 8698–8704 (cit. on p. 99).

[132] Yinghui Zhao, Chris Rausch, and Carl Haas. "Optimizing 3D irregular object packing from 3D scans using metaheuristics". In: *Advanced Engineering Informatics* 47 (2021), p. 101234 (cit. on p. 99).

[133] Chi Trung Ha, Trung Thanh Nguyen, Lam Thu Bui, and Ran Wang. "An online packing heuristic for the three-dimensional container loading problem in dynamic environments and the Physical Internet". In: *European Conference on the Applications of Evolutionary Computation*. Springer. 2017, pp. 140–155 (cit. on pp. 99, 106, 110).

[134] Fan Wang and Kris Hauser. "Robot packing with known items and nondeterministic arrival order". In: *IEEE Transactions on Automation Science and Engineering* 18.4 (2020), pp. 1901–1915 (cit. on p. 99).

[135] Young-Dae Hong, Young-Joo Kim, and Ki-Baek Lee. "Smart pack: online autonomous object-packing system using RGB-D sensor data". In: *Sensors* 20.16 (2020), p. 4448 (cit. on p. 100).

[136] Lu Duan, Haoyuan Hu, Yu Qian, et al. "A Multi-task Selected Learning Approach for Solving 3D Flexible Bin Packing Problem". In: *Proceedings of the International Conference on Autonomous Systems and Multiagent Systems (AAMAS)*. 2019 (cit. on p. 100).

[137] Hang Zhao, Chenyang Zhu, Xin Xu, Hui Huang, and Kai Xu. "Learning practically feasible policies for online 3D bin packing". In: *Science China Information Sciences* 65.1 (2022), pp. 1–17 (cit. on p. 100).

[138] Andrea Lodi, Silvano Martello, and Daniele Vigo. "Heuristic algorithms for the three-dimensional bin packing problem". In: *European Journal of Operational Research* 141.2 (2002), pp. 410–420 (cit. on p. 105).

[139] Teodor Gabriel Crainic, Guido Perboli, and Roberto Tadei. "Extreme point-based heuristics for three-dimensional bin packing". In: *Informs Journal on Computing* 20.3 (2008), pp. 368–384 (cit. on p. 105).

[140] Yong Wu, Wenkai Li, Mark Goh, and Robert De Souza. "Three-dimensional bin packing problem with variable bin height". In: *European Journal of Operational Research* 202.2 (2010), pp. 347–355 (cit. on p. 105).

[141] Francisco Parreño, Ramón Alvarez-Valdés, José Fernando Oliveira, and José Manuel Tamarit. "A hybrid GRASP/VND algorithm for two-and three-dimensional bin packing". In: *Annals of Operations Research* 179.1 (2010), pp. 203–220 (cit. on p. 110).

[142] José Fernando Gonçalves and Mauricio GC Resende. “A parallel multi-population biased random-key genetic algorithm for a container loading problem”. In: *Computers & Operations Research* 39.2 (2012), pp. 179–190 (cit. on p. 110).

Part II

The research

7

Pick and Place Operations in Logistics Using a Mobile Manipulator Controlled with Deep Reinforcement Learning

Authors

Iriondo, Ander; Lazkano, Elena; Susperregi, Loreto; Urain, Julen; Fernandez, Ane; Molina, Jorge;

Publisher

Multidisciplinary Digital Publishing Institute (MDPI)

Journal

Applied Sciences

Year

2019

Quartile (Web of Science / Scimago)

Q2/Q1

DOI

https://doi.org/10.3390/app9020348

applied sciences

MDPI

Article

Pick and Place Operations in Logistics Using a Mobile Manipulator Controlled with Deep Reinforcement Learning

Ander Iriondo [1,*], Elena Lazkano [2], Loreto Susperregi [1], Julen Urain [1], Ane Fernandez [1] and Jorge Molina [1]

[1] Department of Autonomous and Intelligent Systems, Fundación Tekniker, Iñaki Goenaga, 5-20600 Eibar, Spain; loreto.susperregi@tekniker.es (L.S.); julen.urain@tekniker.es (J.U.); ane.fernandez@tekniker.es (A.F.); jorge.molina@tekniker.es (J.M.)

[2] Faculty of Computer Science, Pº Manuel Lardizabal, 1-20018 Donostia-San Sebastián, Spain; e.lazkano@ehu.es

* Correspondence: ander.iriondo@tekniker.es

Received: 30 December 2018; Accepted: 17 January 2019; Published: 21 January 2019

Abstract: Programming robots to perform complex tasks is a very expensive job. Traditional path planning and control are able to generate point to point collision free trajectories, but when the tasks to be performed are complex, traditional planning and control become complex tasks. This study focused on robotic operations in logistics, specifically, on picking objects in unstructured areas using a mobile manipulator configuration. The mobile manipulator has to be able to place its base in a correct place so the arm is able to plan a trajectory up to an object in a table. A deep reinforcement learning (DRL) approach was selected to solve this type of complex control tasks. Using the arm planner's feedback, a controller for the robot base is learned, which guides the platform to such a place where the arm is able to plan a trajectory up to the object. In addition the performance of two DRL algorithms ((Deep Deterministic Policy Gradient (DDPG)) and (Proximal Policy Optimisation (PPO)) is compared within the context of a concrete robotic task.

Keywords: deep reinforcement learning; mobile manipulation; robot learning

1. Introduction

Logistics applications demand the development of flexible, safe and dependable robotic solutions for part-handling including efficient pick-and-place solutions.

Pick and place are basic operations in most robotic applications, whether in industrial setups (e.g., machine tending, assembling or bin picking) or in service robotics domains (e.g., agriculture or home). In some structured scenarios, picking and placing is a mature process. However, that is not the case when it comes to manipulating parts with high variability or in less structured environments. In this case, picking systems only exist at laboratory level, and have not reached the market due to factors such as lack of efficiency, robustness and flexibility of currently available manipulation and perception technologies. In fact, the manipulation of goods is still a potential bottleneck to achieve efficiency in the industry and the logistic market.

At the same time, the market demands more flexible systems that allow for a reduction of costs in the supply chain, increasing the competitiveness for manufacturers and bringing a cost reduction for consumers. The introduction of robotic solutions for picking in unstructured environments requires the development of flexible robotic configurations, robust environment perception, methods for trajectory planning, flexible grasping strategies and human–robot collaboration.

This study focused specifically on the development of adaptive trajectory planning strategies for pick and place operations using mobile manipulators. A mobile manipulator is a mobile platform carrying one or more manipulators. The typical configuration of mobile manipulators in industry is an anthropomorphic manipulator mounted on a mobile platform complemented with sensors (laser, vision, and ultrasonic) and tools to perform operations. The mobility of the base extends the work-space of the manipulator, which increases the operational capacity.

Currently, mobile manipulators implementation in industry is limited. One of the main challenges is to establish coordinated movements between the base and the arm in a unstructured environment depending on the application.

Although navigation and manipulation are fields where much work has been done, mobile manipulation is a less known area because it suffers from the difficulties and uncertainties of both previously mentioned fields. We propose to solve the path planning problem using a reinforcement learning (RL) strategy. The aim is to avoid explicit programming of hard-to-engineer behaviours, or at least to reduce it, taking into account the difficulty of foreseeing situations in unstructured environments. RL-based algorithms have been successfully applied to robotic applications [1] and enable learning complex behaviours only driven by a reward signal.

Specifically, this study focused on learning to navigate to such a place that the mobile manipulator's arm is able to pick an object from a table. Due to the limited scope of the arm, not all positions near the table are feasible to pick the object and to calculate those positions analytically is not trivial. The goal of our work was to evaluate the performance of deep reinforcement learning (DRL) [2] to acquire complex control policies such as mobile manipulator positioning by learning only through the interaction with the environment. Specifically, we compared the performance of two model-free DRL algorithms, namely Deep Deterministic Policy Gradient (DDPG) and Proximal Policy Optimisation (PPO) algorithms, in two simulation tests.

The simulated robot we used is the mobile manipulator miiwa, depicted in Figure 1, which has been used in industrial mobile manipulation tasks (e.g., [3]).

Figure 1. Kuka miiwa.

2. Literature Review

There are different approaches to establishing coordinated movements between the base and the arm depending on the application. Nassal et al. [4] specified three types of cooperation, which differ in the degree of cooperation, the associated complexity and the potential for manipulation capabilities: (1) Loose cooperation is when the mobile base and the manipulator are considered two separate systems, and the base serves as transport. There are several very well-known methods for navigation and trajectory planning [5]. (2) Full cooperation is when the two systems are seen as one (with nine degrees of freedom), and both the base and the arm move simultaneously to position the tool center. There are various approaches to solve the path planning problem [6,7], however, in these cases, the computational cost is very high. (3) Transparent cooperation is the combination of the previous two: the manipulator control compensates for the base and the base moves accordingly to maximise a cost function related to the positioning of the manipulator to perform the task. In [4], an approach to this type of control is proposed.

In general, current robotic solutions follow the loose cooperation approach. The coordination of the two subsystems for mobile manipulation depends on the task that needs to be solved. Berntorp et al. [8] addressed the issue of pick and place where the robotic system must take a can of a known position and place it in another combining the movement of the arm and the base. In [9], the problem of opening doors is considered where the movement of the arm and base are coupled by sensing the forces generated between the arm and the door, and coordinating the forward movement of the base.

One of the principal goals of artificial intelligence (AI) is to enable agents to learn from their experience to become fully autonomous. This experience is obtained by interacting with the environment. Thus, the agent should continue improving its behaviour through trial and error until it behaves optimally. Deep learning enables reinforcement learning to face decision-making problems that were previously infeasible. The union of both of them, namely DRL, makes it possible to learn complex policies guided by a reward signal and has been applied to learn multiple complex decision making tasks in a wide variety of areas. For instance, in [10], the authors learn a chess player agent by self playing that is able to defeat a world champion. In [11], DRL is used to learn an intelligent dialogue generator agent. It also has been successfully applied to computer vision, for example to train a 2D object tracker agent that outperforms state-of-the-art real-time trackers [12].

The field of DRL applied to robotics is becoming more and more popular, and the number of published papers related to that topic is increasing quickly. Applications extend from manipulation [13–16] to autonomous navigation [17,18] and locomotion [19,20].

DRL has been successfully applied in some robotic path planning and control problems. For example, Gu et al. [16] used DRL to learn a low-level controller that is able to control a robotic arm to perform door opening. On the one hand, 25 variables are used to represent the environment's state. More specifically, those variables correspond to the joint angles and their time derivatives, the goal position and the door angle together with the handle angle. On the other hand, the action is represented by the torque for each joint. Robot navigation is another area where DRL also has been applied successfully; for example, in [18], DRL is applied to learn a controller to perform mapless navigation with a mobile robot. In this work, a low dimensionality state representation is also used, specifically 10-dimensional laser range findings, the previous action and the relative target position. The action representation includes linear and angular velocities of the differential robot.

Alternatively, several methods use 2D or 3D images instead of low-dimensional state representations. In [15], for example, 2D images are used with the goal of learning grasping actions. The authors tried to map images directly with low-level control actions and, in addition, the developed system is able to generalise and pick objects that the model has not been trained with. Moreover, other applications (e.g., [21]) use 3D depth images as state representation, but instead of learning directly from pixels, they encode the images to a lower dimensional space before training the model. This encoding enables to accelerate the training process.

Although there are multiple works about learning to control either robotic arms and mobile bases, to the best of our knowledge, references about DRL applied to mobile manipulation are few. The goal of this paper is to show that it is possible to use the arm path planner's feedback to learn a low-level controller for the base. The learned controller is able to guide the robot to a feasible pose in the environment to do the picking trial. To do that, a low-dimensional state representation is used.

In [22], the authors proposed to learn, through interaction with the environment, which place is the best to locate the base of a mobile manipulator, in such a way that the arm is able to pick an object from a table. In that way, the robot takes into account its physical conditions and also the environment's conditions to learn which is the optimal place to perform the grasping action. In this case, they acquire experience off-line, and, after applying some classifiers, such as support vector machines, they are able to learn a model that maps the state of the robot with some feasible places in the environment to do the picking trial. Then, the trained model is used on-line.

In comparison with the previous approach, the goal of the RL-based algorithms is to learn on-line, while the agent interacts with the environment, improving its policy until it reaches to the optimal behaviour. In addition, the goal of our work is to learn a controller, which gives a low-level control signal in each state to drive the mobile robot, while, in [22], instead of learning a controller to drive the robot up to the optimal pose, they tried to find directly which is the optimal pose.

3. Methodological Approach

According to Sutton and Barto [23], a reinforcement learning solution to a control problem is defined as a finite horizon Markov Decision Process (MDP). At each discrete time-step t, the agent observes the current state of the environment $s_t \in S$, takes an action $a_t \in A(s_t)$, receives a reward $r{:}S \times A \rightarrow \mathbb{R}$ and observes the new state of the environment s_{t+1}. At each episode of T time-steps, the environment and the agent are reset to their initial poses. The goal of the agent is to find a policy, deterministic $\pi_\theta(s)$ or stochastic $\pi_\theta(a|s)$, parameterised by θ under which the expected reward is maximised.

Traditional reinforcement learning algorithms are able to deal with discrete state and action spaces but, in robotic tasks, where both state and action spaces are continuous, discretising those spaces does not work well. Nevertheless, most state-of-the-art algorithms use deep neural networks to map observations with low-level control actions to be able to deal with continuous spaces. In our approach, a DRL algorithm is used to learn where to place the mobile manipulator to make a correct picking trial through the interaction with the environment. Those DRL algorithms need a huge amount of experience to be able to learn complex robotic behaviours and, thus, it is infeasible to train them acquiring experience in real world. In addition, the actions taken by the robot in the initial learning iterations are nearly random and both the robot and the environment might end up damaged as a result. Therefore, DRL algorithms are usually trained in simulation. Learning in simulation enables a faster experience acquisition and avoids material costs. Our study used a simulation-based approach, and we based it on open source robotic tools such as Gazebo simulator, Robot Operating System (ROS) middleware [24], OpenAI Baselines DRL library [25] and Gym/Gym-gazebo toolboxes [26,27].

DRL algorithms follow value-based, policy search or actor–critic architectures [28]. Value based-algorithms estimate the expected reward of an state or state–action pair and are able to deal with discrete action spaces, typically using greedy policies. To be able to deal with continuous action spaces, policy search methods use a parameterised policy and do not need a value function. Usually, those methods have the difficulty of not easily being able to evaluate the policy and they have high variance. Actor–critic architecture is the most used one in the state-of-the-art algorithms and combines the benefits of the two previous algorithm types. On the one hand, actor–critic-based methods use a parameterised policy (actor), and, on the other hand, use a value or action–value function that evaluates the quality of the policy (critic).

The proposed approach follows the actor–critic architecture. On the one hand, a parameterised policy π_θ is used to be able to deal with both continuous state and action spaces in stochastic

environments, encoded by the parameter vector θ. On the other hand, a parameterised value function is used to estimate the expected reward at each state or state–action pair, where w is the parameter vector that encodes the critic. The state value function $V_w(s)$ estimates the expected average reward of all actions in the state s. The action–value function $Q_w(s, a)$, instead, estimates the expected reward of executing action a in state s. Then, the critic's information is used to update both actor and critic. In DRL algorithms, both actor and critic are parameterised by deep neural networks, and the goal is to optimise those parameters to get the agent's optimal behaviour.

In addition, DRL algorithms are divided into two groups, on-policy and off-policy, depending on how they are able to acquire experience. On-policy algorithms expect that the experience used to optimise their behaviour policy is generated by the same policy. Off-policy methods, instead, can use experience generated by another policy to optimise its behaviour policy. Those methods are said to be able to better explore the environment than on-policy methods because they use a more exploratory policy to get experience. In this study, we compared an on-policy algorithm and an off-policy algorithm, to see which type of methods adjusts better to our mobile manipulation behaviour learning. Specifically, PPO and DDPG were used, being those on-policy and off-policy, respectively. The first one learns stochastic policies, which maps states with actions represented by Gaussian probability distributions. DDPG, instead, is able to deterministically map states with actions. Both algorithms follow actor–critic architecture.

3.1. Algorithms

In this section, we describe the theoretical basis of PPO and DDPG.

3.1.1. PPO

PPO [29] follows the actor–critic architecture and it is based on the trust-region policy optimisation (TRPO) [30] algorithm. This algorithm aims to learn a stochastic policy $\pi_\theta(a_t|s_t)$ that maps states with Gaussian distributions over actions. In addition, the critic is a value function $V_w(s_t)$ that outputs the mean expected reward in state s_t. This algorithm has the benefits of TRPO and in general of trust region based methods but it is much simpler to implement it. The intuition behind trust-region based algorithms is that, at each parameter update of the policy, the output distribution cannot diverge too much from the original distribution.

To update the actor's parameters, a clipped surrogate objective is used. Although another loss function is also proposed, using a Kullback–Leibler (KL) divergence [31] penalty on the loss function instead of the clipped surrogate objective, the experimental results obtained are not as good as with the clipped one. Let $r_t(\theta)$ denote the probability ratio defined in Equation (1), so that $r_t(\theta_{old}) = 1$.

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{1}$$

θ_{old} is the actor's parameter vector before the update. The objective of TRPO is to maximise the objective function $L(\theta)$ defined in Equation (2). Here, $\mathbb{E}[...]$ indicates the average over a finite batch of samples. The usage of the advantage $\hat{A}_t$ in policy gradient algorithms was popularised by Schulman et al. [32] and indicates how good the performed action is with respect to the average actions performed in each state. To compute the advantages, the algorithm executes a trajectory of T actions and computes them as defined in Equation (3). Here, t denotes the time index $[0, T]$ in the trajectory of length T and γ is the discount factor.

$$L(\theta) = \mathbb{E}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t\right] \tag{2}$$

$$\hat{A}_t = -V_w(s_t) + r_t + \gamma r_{t+1} + ... + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V_w(s_T) \tag{3}$$

At each policy update, if the advantage has a positive value, the policy gradient is pushed in that direction because it means that the action performed is better than the average. Otherwise, the gradient is pushed in the opposite direction.

Without any constraint, the maximisation of the loss function $L(\theta)$ would lead to big changes in the policy at each training step. PPO modifies the objective function so that penalises big changes in the policy that move $r_t(\theta)$ away from 1. Maintaining $r_t(\theta)$ near to 1 ensures that, at each policy update, the new distribution does not diverge to much from the old one. The objective function is defined in Equation (4).

$$L^{CLIP}(\theta) = \mathbb{E}[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon))\hat{A}_t] \tag{4}$$

Here, ϵ is a hyper-parameter that changes the clip range.

To update the value function $V_w(s)$ (the critic), the squared-error loss function is used (Equation (5)) between the current state value and a target value. The target value is defined in Equation (6).

$$J(w) = (V_w(s_t) - V_t^{targ})^2 \tag{5}$$

$$V_t^{targ} = \hat{A}_t + V_w(s_t) \tag{6}$$

The PPO algorithm is detailed in Algorithm 1. Although this algorithm is designed to be able to have multiple parallel actors getting experience, only one actor is being used.

Algorithm 1 Proximal Policy Optimisation (PPO).

1: **for** $e \in episodes$ **do**
2: **for** $a \in actors$ **do**
3: Run policy $\pi_{\theta_{old}}$ in environment for T time-steps
4: Compute advantage estimates $\hat{A}_1 ... \hat{A}_T$
5: **end for**
6: Optimise actor's loss function L^{CLIP} with regard to θ, with K epochs and minibatch size N $\leq T \cdot actors$
7: $\theta_{old} \leftarrow \theta$
8: **end for**

3.1.2. DDPG

DDPG [33] combines elements of value function based and policy gradient based algorithms, following the actor–critic architecture. This algorithm aims to learn a deterministic policy $\pi_\theta(s) = a$ and it is derived from the deterministic policy gradient theorem [34].

Following the actor–critic architecture, DDPG uses an action–value function $Q_w(s, a)$ as critic to guide the learning process of the policy and it is based on the deep Q-network (DQN) [35]. Prior to DQN, it was believed that learning value functions with large and nonlinear function approximators was difficult. DQN is able to learn robust value functions due to two innovations: First, the network is trained off-policy getting experience samples from a replay buffer to eliminate temporal correlations. In addition, target networks are used, which are updated more slowly, and this gives consistent targets during temporal difference learning.

The critic is updated according to the gradient of the objective defined in Equation (7).

$$L(w) = \mathbb{E}[(Q_w(s_t, a_t) - y_t)^2] \tag{7}$$

where

$$y_t = r(s_t, a_t) + \gamma Q_w(s_{t+1}, a_{t+1})|_{a_{t+1}=\pi_\theta(s_{t+1})} \tag{8}$$

The actor is updated following the deterministic policy gradient theorem, defined in Equation (9). The intuition is to update the policy in the direction that improves $Q_w(s,a)$ most .

$$\nabla J(\theta) = \mathbb{E}[\nabla_\theta \pi_\theta(s_t) \nabla_a Q_w(s_t, a_t)|_{a=\pi_\theta(s_t)}] \tag{9}$$

As mentioned before, the target value defined in Equation (8) is calculated using the target networks $\pi'_{\theta'}$ and $Q'_{w'}$, which are updated more slowly and this gives more consistent targets when learning the action–value function.

As DDPG is an off-policy algorithm, the policy used to get experience is different from the behaviour policy. Despite the behaviour policy being deterministic, typically, a stochastic policy is used to get experience, being able to better explore the environment. This exploratory policy is usually achieved adding noise to the behaviour policy. Although there are common noises such as normal noise or Ornstein–Uhlenbeck noise [36], which are added directly to the action generated by the policy, in [37], Plapperta et al. proposed adding noise to the neural network's parameter space to improve the exploration and to reduce the training time. The DDPG algorithm is described in Algorithm 2.

Algorithm 2 Deep Deterministic Policy Gradient (DDPG).

Initialise the actor $\pi_\theta(s)$ and the critic $Q_w(s,a)$ networks.
Initialise the target networks Q' y π' with the weights $\theta' \leftarrow \theta$, $w' \leftarrow w$
Initialise the replay buffer
for $e \in episodes$ **do**
    Initialise noise generation process
    Get the first observation
    for $t \in steps$ **do**
        Select the action $a_t = \pi_\theta(s_t) + N$
        Execute the action a_t, get the reward r_t and the observation s_{t+1}
        Store the transition $< s_t, a_t, r_t, s_{t+1} >$ in replay buffer
        Get M experience samples $< s_i, a_i, r_i, s_{i+1} >$ from the replay buffer
        $y_i = r_i + \gamma Q'_{w'}(s_{i+1}, \pi'_{\theta'}(s_{i+1}))$
        Update the critic minimising the loss :

$$L = \frac{1}{M} \sum_i (y_i - Q_w(s_i, a_i))^2$$

        Update the policy of the actor:

$$\nabla_\theta J \approx \frac{1}{M} \sum_i \nabla_a Q_w(s,a)|_{s=s_i, a=\pi_\theta(s_i)} \nabla_\theta \pi_\theta(s)|_{s=s_i}$$

        Update target networks:

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta'$$

$$w' \leftarrow \tau w + (1-\tau)w'$$

    end for
end for

3.2. Simulated Layout

To model our world, we used the Gazebo model based simulator. The elements that are placed in this simulated world are the robot miiwa, the table and an object, which is on top of the table, as depicted in Figure 2.

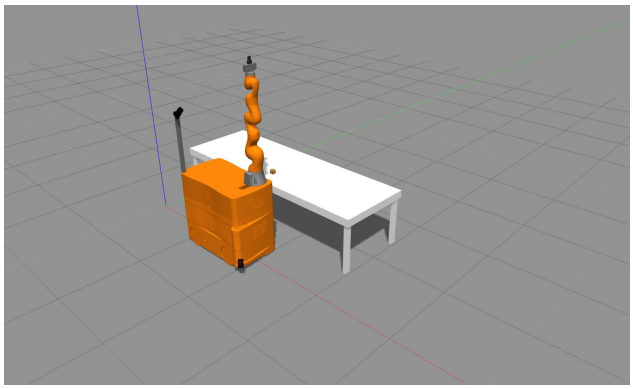


Figure 2. Simulated world in Gazebo.

The mobile manipulator miiwa is composed of a 7 DoF arm and a 3 DoF omnidirectional mobile base. To be able to control the mobile base, the *gazebo planar move* plugin was used.

To test if our environment was modelled correctly and to know if the algorithms could learn low-level control tasks in that environment, the learning process was divided into two simulation tests. In both, the algorithm must learn to control the mobile base with velocity commands, such that, at each discrete time-step, the algorithm gets the state of the environment and publishes a velocity command. The objective of those tests was to learn a low-level controller to drive the robot to a place where the arm can plan a trajectory up to the object on the table. To learn those controllers, the feedback of the arm's planner was used. The summary of the tests is listed in Table 1.

Table 1. Test setup summary.

Test	Robot Initial Pose	Box Initial Pose	Objective
Test 1	Variable	Variable	The arm to be able to plan a trajectory up to the box
Test 2	Variable	Constant	To plan a trajectory with an obstacle in the table

4. Implementation

The application was implemented in a modular way using ROS for several reasons. On the one hand, it enabled us to modularise the application and to parallelise processes. On the other hand, Gazebo is perfectly integrated with ROS and offers facilities to control the simulation using topics, services and actions.

OpenAI Gym is a toolkit to do research on DRL algorithms and to do benchmarks between DRL algorithms. This library includes some simulated environments that are used to test the quality of new algorithms and its use is widespread in the DRL community. Gym offers simple functions to interact with the environments, which are mostly modelled in the Mujoco [38] simulator. Due to the simplicity of the interface that Gym offers, it has become a standard way to interact with environments in DRL algorithms. Gym-gazebo is an extension of Gym that enables the user to create robotic environments in Gazebo and offers the same simple interface to the algorithm to be able to interact with the environment. All the environments we modelled were integrated with Gym-gazebo, which enabled us to straightforwardly use OpenAI Baselines DRL library, which is designed to work with Gym.

Gym-gazebo wraps the used DRL algorithms in ROS nodes and enables interaction with the environment. Nevertheless, another ROS node has been developed that is in charge with controlling all the elements of the simulation and works as bridge between the Gym-gazebo node and the simulator. To be able to control the simulation physic updates and to compute them as fast as possible, we developed a Gazebo plugin, which in turn is a ROS node. Thus, using ROS communication methods, we could control each simulated discrete time-step and we simulated those steps faster than real time.

Specifically, this node takes care of all time-steps being of the same length, executing a fixed number of physic updates at each step.

The *tf broadcaster* node uses the *tf* tool that ROS offers to keep track of all the transformations between frames. We used this node to publish some transformations such as the transformation between the *object* and *world* frames. Consequently, the robot is always aware of where the object is in order to be able to navigate up to it. The implemented architecture is shown in Figure 3.

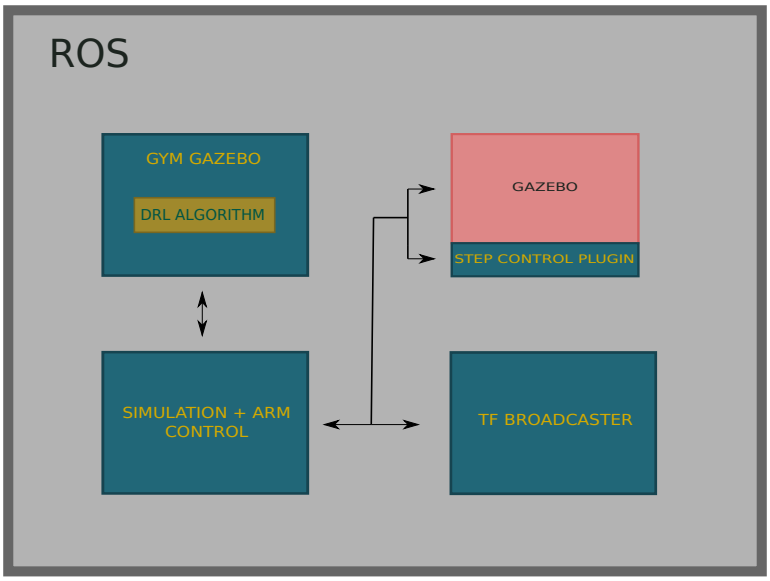


Figure 3. Implemented architecture. ROS: Robot Operating System.

4.1. Simulation

The learning process was carried out during a fixed number of time-steps, which were divided into episodes of 512 time-steps. Gazebo's *max step size* T_m is an important parameter that indicates the time-step at which Gazebo computes successive states of the model. The default value is 1 ms, but, in this case, $T_m = 2$ ms was used, which gave enough stability and enabled a faster simulation. Thus, each iteration of the physic engine meant 2 ms of simulated time and those iterations could be computed as quickly as possible to be able to accelerate the simulation. Besides, the *real time update rate* U_r parameter indicates how many physic iterations will be tried in one second. The *real time factor* is defined in Equation (10) and indicates how much faster the simulation goes in comparison with the real time. Thus, $rtf = 1$ indicates that the simulation is running on real time.

$$rtf = T_m \cdot U_r \tag{10}$$

In addition, a *frame-skip* N_{T_m} was defined to be able to get a reasonable control rate. In this application, $N_{T_m} = 4$ was used so the discrete time-step size is defined in Equation (11) and doing that we achieve a control rate of 125 Hz.

$$T_s = T_m \cdot N_{T_m} = 8 \text{ ms} \tag{11}$$

Thus, the length of each discrete time-step is 8 ms and those steps are computed faster than real time. Gazebo does not allow the control of the physics engine iterations, so a Gazebo plugin was developed to be able to execute the simulation for some iterations and to be able to compute those iterations as fast as possible. When the plugin was told to run a time-step, it ran N_{T_m} physics iterations and the *real time update rate* was set to 0 to compute those iterations as quickly as possible. Doing this, the *real time factor* increased and enabled us to run the simulation about 5–10 times faster than in real time.

At each step, the learning algorithm generates an action, specifically a velocity command, which is sent to the mobile base. After executing it and at the end of the step, a reward signal is given evaluating the quality of the action performed by the base. The action is the same for all the experiments and is defined in Equation (12).

$$a = [v_x \;\; v_y \;\; \omega_z] \tag{12}$$

To be able to control the mobile base, the *gazebo planar move* plugin was used, which enabled the sending of velocity commands to the omnidirectional base. In addition, it published the odometry information, which was used to know where the robot is respect to a parent coordinate frame. The path planning and control of the arm was made using MoveIt! [39] and enabled us to plan collision free trajectories. Besides that, MoveIt! uses an internal planning scene where objects can be added to be taken into account when the trajectory is planned.

4.2. Network Architectures

The library of DRL algorithms used is OpenAI Baselines. This library offers high quality implementations of many state-of-the-art DRL algorithms implemented in Tensorflow [40]. Although the implementation offers complex networks such as convolutional neural networks (CNN) or recurrent neural networks (RNN), we used fully connected multi-layer perceptrons (MLP) to parameterise both policies and value functions.

4.2.1. PPO

The network architecture used in this algorithm is the one proposed by Schulman et al. [29] in the original paper. To parameterise both the policy $\pi_\theta(a_t|s_t)$ and the value function $V_w(s_t)$, a four layered MLP was used. In both actor and critic, the first layer's size depends on the size of the state encoding, which was different in each test. The actor's input layer is followed by two hidden layers of 64 neurons each. The output layer's size depends on the action space's size. In this case, the action was a vector of three elements representing the velocity command, which has to be sent to the mobile base. Specifically, the velocity command is composed of the linear velocities in x and y axes, and the angular velocity in z axis. As mentioned before, this algorithm uses a stochastic policy and, thus, each action is defined by a Gaussian distribution. Each distribution is characterised by a mean and a standard deviation. Hence, the neurons of the last hidden layer are fully connected with the mean of each action. In addition, three variables are used to store and update the standard deviation.

The activation function applied to the output of each neuron of the hidden layers is *Tanh*. Instead, no activation function is applied to the output of the neurons of the last layer.

Regarding the value function $V_w(s_t)$, the first three layers have the same architecture as the actor's first three layers. The output layer, instead, is composed of a single neuron, which is fully connected with each neuron of the last hidden layer and outputs the expected average reward of a state. The activation function used in the hidden layers is *Tanh* as well.

4.2.2. DDPG

The network architecture used is the default implementation that the OpenAI Baselines library offers, which is very similar to the architecture proposed by Lillicrap et al. [33]. As in PPO, to parameterise both the policy $\pi_\theta(s_t)$ and the action–value function $Q(s_t, a_t)$, a four layered MLP was used. This algorithm aims to learn a deterministic policy so each state will be mapped with a concrete action. The actor's input layer also depends on the state encoding and it is followed by two hidden layers, composed of 64 neurons each. Concerning the output layer, it is composed of three neurons, one per action, and each neuron is fully connected with each neuron of the last hidden layer.

With respect to the activation functions, the *Rectified Linear Unit* (ReLU) function is used in each neuron of the hidden layers. In the output layer, instead, *Tanh* is used to bound the actions between -1 and 1. In addition, a process called *layer normalisation* [41] is applied to the output of each hidden

layer to simplify the learning process and to reduce the training time. DDPG is an off-policy algorithm, hence it uses a more exploratory policy to get experience. As explained in the algorithm's description (Section 3.1.2), there are multiple types of noise but, in this application, the noise is added in the neural network's parameter space.

The action–value function $Q_w(s, a)$ receives as input the state and the action and outputs the expected future reward of executing action a_t in state s_t. The input layer depends on the size of the state codification. The input layer is followed by two hidden layers of 64 neurons each and actions are not included until the second hidden layer. The output layer only has one neuron and outputs the q-value. The activation function used in the critic network is ReLU for each neuron of the hidden layers. The last layer's neurons do not have any activation function, because the action–value does not have to be bounded.

To minimise the complexity of the critic and to avoid over-fitting, a $L2$ regularisation term is added to the critic's loss function [42]. This penalises the model for having parameters with high values.

4.3. Test Setup

Here, we describe the simulation setup in addition to the state codification and the reward function we used in each test. For both PPO and DDPG, the setup and reward functions were the same in all tests.

The robot's work-space is limited and a new episode starts when it goes out of limits. Those limits are defined in Equation (13).

$$x_r^w = [-1.0, 3.0] \quad y_r^w = [-1.5, 0.0] \tag{13}$$

In addition, although the robot was penalised by navigating with high speed, the environment bounds the velocities applied to the mobile base. The maximum linear velocity in x and y axes was 1 m/s and the maximum angular velocity in z axis was 1 rad/s. Thus, if the algorithm predicted velocities higher than those, the limit velocities were applied and the agent was penalised proportionally. Besides, high accelerations were also penalised proportionally, but in this case the environment did not bound them.

The learning process was divided into episodes where the robot had T time-steps to complete the task. The episode length was $T = 512$ time-steps, which is about 4 s, and a discrete time-step t was terminal if:

- The robot collides with the table.
- Robot poses out of limits.
- $t = T$.

The tuning hyper-parameters used in each algorithm are described in Table 2 and were not changed across tests.

Table 2. Hyper-parameters.

PPO Setup Hyper-Parameters	
Actor/Critic learning rate	$f(step) = step \cdot 3 \times 10^{-4}$
Clip-range	0.2
Discount factor γ	0.99
Batch size	512
Mini-batch size	64
Updates	$\frac{training\ time\_steps}{batch\_size}$
Training epochs per update	10
DDPG Setup Hyper-Parameters	
Actor's learning rate	1×10^{-4}
Critic's learning rate	1×10^{-3}
Batch size	128
Discount factor γ	0.99
Critic l2 regularisation	1×10^{-2}
Running epochs	500
Cycles per epoch	10
Rollouts per cycle	512
Updates	$epochs \cdot cycles$
Training iterations per update	100

4.3.1. Test 1

Here, the objective was to learn a controller that guides the robot to a place near the table so that the arm could plan a trajectory up to the object. The robot and object initial poses were variable, which are defined in Equations (14) and (15). The robot's initial y_r^w coordinate was constant so that the robot always began at the same distance from the table. The state codification is defined in Equation (16) and it is composed of the following 10 variables:

Robot's position in world coordinate system: x_r^w, x_r^w
Robot's rotation on z axis in world coordinate system: yaw_r^w
Robot's linear velocities on x and y axes: v_x, v_y
Robot's angular velocity in z axis: ω_z
Object's position in world coordinate system: x_{obj}^w, y_{obj}^w
Distance between the robot and the object: $d(p_r, p_{obj})$
Remaining time steps to end the episode: t

$$x_r^w = [0.0, 3.0] \quad y_r^w = -1.5 \quad yaw_r^w = [-\pi, \pi] \tag{14}$$

$$x_{obj}^w = [0.5, 2.5] \quad y_{obj}^w = [0.5, 0.75] \quad z_{obj}^w = 1.2 \tag{15}$$

$$s = [x_r^w \ \ y_r^w \ \ yaw_r^w \ \ v_x \ \ v_y \ \ \omega_z \ \ x_{obj}^w \ \ y_{obj}^w \ \ d(p_r, p_{obj}) \ \ t] \in \mathbb{R}^{10} \tag{16}$$

The linear and the angular velocities included in the state were the velocities sent in the previous time-step to the robot (previous action). The reward function used is defined in Equation (17). A nonlinear function was used to give higher rewards when the robot was close to the object and high velocities and accelerations were penalised to encourage smooth driving. Here, Δv means the velocity difference (acceleration) between current and previous action. Instead, v_high penalised each linear or angular velocity being higher than a threshold. As mentioned before, the maximum linear and angular velocities were 1 m/s and 1 rad/s, respectively. In addition, in the last time-step of the episode, an additional reward was given if the arm could plan a trajectory up to the object. The number of remaining time-steps was included in the state for robot to be aware when this last step was coming. The *collision* variable was equal to 1 when a collision occurred, and 0 otherwise.

$$r(s_t, a_t) = \frac{1}{d(p_r, p_{obj})} \cdot (1 - collision) - 0.5 \cdot \Delta v - 0.5 \cdot v\_high(a_t) + 100 \cdot success \tag{17}$$

$$v\_high(a_t) = \sum_{i=1}^{3} a_{t_i}, \text{ if } a_{t_i} > k \tag{18}$$

4.3.2. Test 2

In this test, as in the previous one, the robot had to navigate to a place near the table such that the arm could plan a trajectory up to the object in the table. In this case, a wall was placed near the object to see if the algorithm could discard poses that were behind the wall. The robot's initial pose was variable (Equation (19)) and the object's initial pose was constant (Equation (20)). As in the first test, the robot's initial y_r^w coordinate was constant so that the robot always began at the same distance from the table. The state codification defined in Equation (21) is composed of the following 8 variables:

Robot's position in object's coordinate system: x_r^{obj}, x_r^{obj}
Robot's rotation on z axis in object's coordinate system: yaw_r^{obj}
Robot's linear velocities on x and y axes: v_x, v_y
Robot's angular velocity in z axis: ω_z
Distance between the robot and object's coordinate system origin: $d(p_r, \mathbb{O})$
Remaining time steps to end the episode: t

$$x_r^w = [0.0, 3.0] \quad y_r^w = -1.5 \quad yaw_r^w = [-\pi, \pi] \tag{19}$$

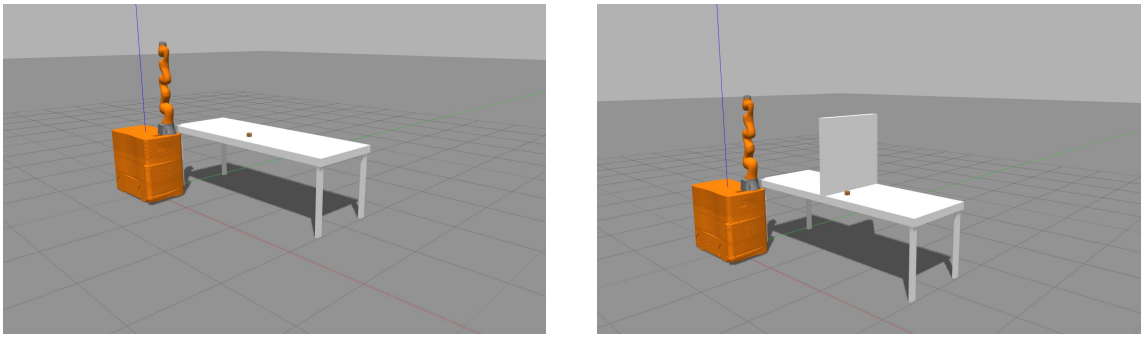
$$x_{obj}^w = 1.7 \quad y_{obj}^w = 0.75 \quad z_{obj}^w = 1.005 \tag{20}$$

$$s = [x_r^{obj} \;\; y_r^{obj} \;\; yaw_r^{obj} \;\; v_x \;\; v_y \;\; \omega_z \;\; d(p_r, \mathbb{O}) \;\; t] \in \mathbb{R}^8 \tag{21}$$

The used reward function is defined in Equation (22). In this case, the distance was computed between the robot's position p_r and object's coordinate system origin $\mathbb{O}$.

$$r(s_t, a_t) = \frac{1}{d(p_r, \mathbb{O})} \cdot (1 - collision) - 0.5 \cdot \Delta v - 0.5 \cdot v\_high(a_t) + 100 \cdot success \tag{22}$$

The environments used in the first and second tests are depicted in Figure 4a,b, respectively.



(**a**) Environment 1. (**b**) Environment 2.

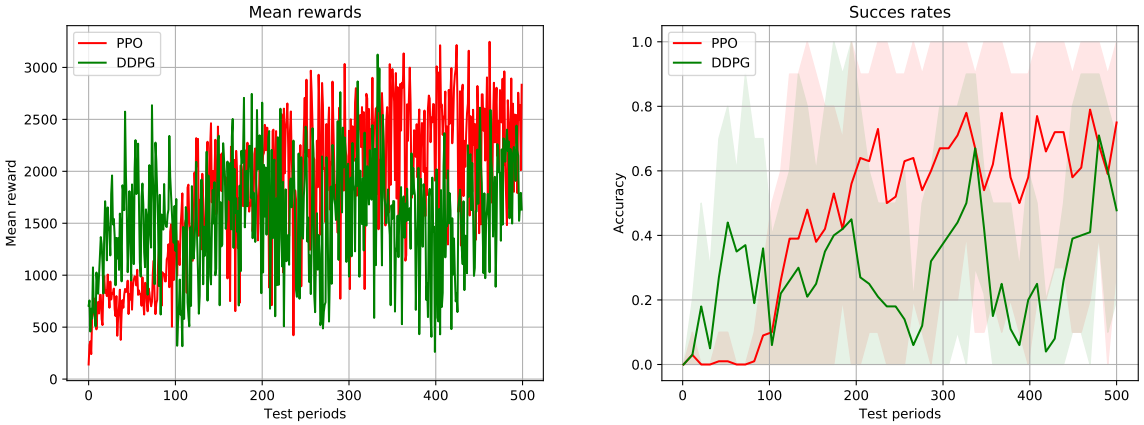
Figure 4. Simulated environments.

5. Results

The learning process was carried out during a fixed number of time-steps and, to be able to evaluate the performance of the algorithms, several evaluation periods of 10 episodes each were made. Specifically, 500 evaluation periods were made uniformly distributed over the learning process and the metrics used to evaluate the performance were the mean accumulated reward and the success rate, which are the most used ones in the DRL community.

5.1. Test 1

The goal of this test was to learn a low-level controller that could drive the mobile manipulator's base close to the table, so that the arm could plan a trajectory up to the object. The learning process was carried out during 5M time-steps approximately, and an evaluation period was performed every 20 episodes. The results obtained with both PPO and DDPG algorithms are depicted in Figure 5. Figure 5a shows the mean accumulated rewards obtained in each of the 500 evaluation periods. The obtained success rates are depicted in Figure 5b. Due to the unstable nature of DRL algorithms, the obtained success rates vary considerably. Thus, to better understand the results, the mean value and the maximum/minimum values are shown per 10 test periods.



(**a**) Accumulated mean rewards. (**b**) Success rates.

Figure 5. Test 1 results.

Concerning the accumulated mean rewards, DDPG converged faster than PPO but, when the learning process moved along, PPO obtained higher rewards. The fact that DDPG obtained higher initial rewards is explained by two main reasons: (1) The policy bounds the actions between -1 and 1 so that it is not penalised for high velocities. (2) From the initial learning steps, this algorithm predicts smooth velocity changes across consecutive time-steps and, thus, is not penalised for high accelerations. Instead, PPO does not bound the actions and that is why it is penalised for high velocities and accelerations in the initial learning steps. Nevertheless, this algorithm is able to understand that lower velocities/accelerations are not penalised and, once learned, it obtains higher rewards than DDPG.

Regarding the success rates, it can be seen clearly that DDPG approximated the goal faster than PPO. Although, in some evaluation periods, it succeeded 100% of the time, it presented an unstable behaviour. As explained before, PPO takes more time to approximate the goal but, when it learns to drive smoothly, it shows a more stable behaviour. Besides, it could get 100% success rate considerably more times than DDPG. Off-policy algorithms are said to be able to better explore the environment than on-policy algorithms. In our environment, due to the initial random poses and the limited work-space of the robot, the exploration problem decreased considerably. PPO is an on-policy algorithm and takes more time to explore the environment than DDPG and that is another reason DDPG approaches the goal faster than PPO. Nevertheless, PPO relatively quickly improves enough for its policy to be able to succeed.

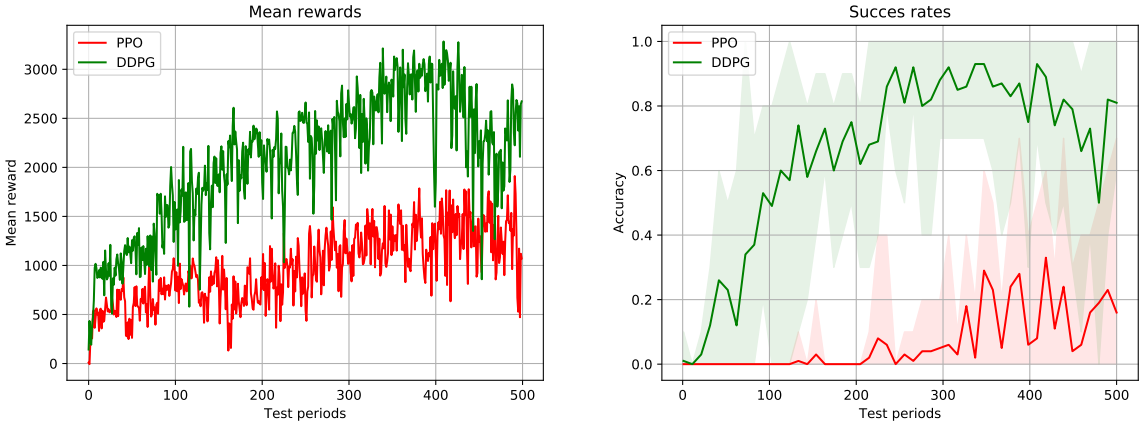
DRL has been applied to goal reaching applications either in manipulation or in navigation. Typically, those goals are not surrounded by obstacles and this makes the learning process easier. In this case, the goal (the object) was on top of the table and the robot had to learn to approximate to it without colliding with the obstacle. The reward function defined in this test encouraged the robot approximating to the object as much as possible and the robot had to use the contact information to learn where the table was to not collide with it. Although an additional reward was given when the

arm's planning succeeded, it first tried to get as close as possible to the table, sometimes colliding with it and that is one of the reasons that caused the unstable behaviour of both algorithms.

Due to the variable pose of the object, in some episodes, if the object was near to table's edge, it had the possibility to get closer, scoring higher rewards. In addition, the initial poses of both the robot and the object in every test period were totally random and thus the accumulated rewards and/or success rates varied considerably.

5.2. Test 2

The goal of this test was to learn a low-level controller that could drive the mobile manipulator's base close enough to the table for the arm to be able to plan a trajectory up to the object. In this case, a wall was placed near the object with the aim of making the decision making problem more difficult. The learning process was carried out during 2.5M time-steps and an evaluation period aws performed every 10 episodes. In Figure 6, the results obtained with both PPO and DDPG algorithms are depicted.



(**a**) Accumulated mean rewards. (**b**) Success rates.

Figure 6. Test 2 results.

Concerning the accumulated mean rewards (Figure 6a), DDPG's performance was better than PPO's for the overall learning process. Due to the bounded actions and the smooth output of DDPG, it could get higher rewards in the initial learning steps. Thanks to the fact that this algorithm is off-policy, it is able to better explore the environment and, thus, it learns to locate the robot's base much closer to the object than PPO. In addition, the constant pose of the box simplified the exploration problem, since the base always had to navigate to the same area of the environment and, consequently, the mean rewards were not as irregular as in the first test.

The success rates depicted in Figure 6b indicate that DDPG's performance was much better than PPO's, being the former able to score 100% success rate multiple times. DDPG could learn relatively quickly where the grasping zone is, discarding the poses over the wall. After navigating to the grasping zone, the learned policy stoped the robot sending near to zero velocities to the base. In addition, the deterministic behaviour of the algorithm enabled the robot to learn a more stable behaviour near the table, avoiding collisions. Even though PPO could learn to drive the robot near the correct grasping zone, it could not learn to stop it and continued sending velocities high enough to get the robot out of the grasping zone. Besides, due to the stochastic policy, the robot's performance near the table was not as robust as DDPG's, sometimes colliding with it.

6. Discussion and Future Work

In this work, we successfully implemented several DRL algorithms for learning picking operations using a mobile manipulator. It is shown that it is possible to use the arm's feedback to learn a low-level controller that drives the base to such a place that the arm is able to plan a trajectory up to the object.

Two state-of-the-art DRL algorithms were applied and compared to learn a mobile manipulation task. Specifically, the arm planner's feedback was used to learn to locate the mobile manipulator's base. To the best of our knowledge, this is the first approach that uses the arm's feedback to acquire a controller for the base.

Although DRL enables the learning of complex policies driven only by a reward signal, the unstable nature of those algorithms makes it difficult to obtain a robust behaviour. In addition, the sensitivity of DRL algorithms to hyper-parameters hinders finding the best parameter combination to get a robust and stable behaviour. Even though an optimisation over those hyper-parameters could be made, the large training times makes this process very expensive and commonly the default values proposed in the literature are used. Concerning the algorithms tested in this work, the results obtained show that the behaviour of the algorithms is dependant on several properties of the environment such as the state/action codification and the reward function definition. In addition, the network architecture used to encode either policies and the value-function has a large effect in the learning process. Even though the same network architecture and hyper-parameters were used in both tests, the results obtained are very different. Therefore, each environment needs the algorithm to be tuned in the best way to solve the problem, which is why the learned policies are not reproducible.

The reward function definition is another key point of the learning process, since it is entirely guided by this signal. A logical approach could be to give a reward to the robot in the last step only if the arm plans a trajectory, but, unfortunately, using only sparse rewards does not work well. In several goal reaching applications, either with arms or mobile bases, a distance dependant reward is proposed. In our application, this encourages the robot to navigate close to the object so the arm can plan a trajectory. After some tests, we saw that a nonlinear distance function accelerates the learning process.

Although DRL algorithms are typically trained in simulation, the experience acquisition is still the bottleneck of DRL based applications applied to robotics. Even though we accelerated the simulation, the entire learning process took several hours. Nevertheless, to be able to transfer the learned policy to a real robot and reduce the reality gap, the robot must be simulated accurately. Thus, a balance between simulation accuracy and training time acceleration should be found.

Moreover, it is complex to tune the algorithms to get a robust performance. We intend to increase the perception capabilities of the robot to be able to navigate in a more secure way and to be aware of the dynamical obstacles placed in the environment, using 2D/3D vision for example. Most of the applications in the literature map observations directly with low-level control actions and this black-box approach is not scalable. To be able to learn multiple behaviours and to combine them, hierarchical DRL proposes to learn a hierarchy of behaviours in different levels. In that vein, our goal is to learn a hierarchy of behaviours and, after training them in simulation, test those behaviours in a real robot.

Author Contributions: Conceptualisation, A.I. and L.S.; methodology, A.I., L.S., and E.L.; software, A.I.; formal analysis, A.I., J.U., A.F., and J.M.; data curation, A.I.; writing—original draft preparation, A.I.; writing—review and editing, E.L. and L.S.; and supervision, E.L., L.S.

Funding: This Project received funding from the European Union's Horizon 2020 research and Innovation Programme under grant agreement No. 780488.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

RL	Reinforcement learning
DRL	Deep reinforcement learning
MDP	Markov Decision Process
ROS	Robot Operating System
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
MLP	Multi Layer Perceptron
DDPG	Deep Deterministic Policy Gradient
PPO	Proximal Policy Optimisation

References

1. Kober, J.; Bagnell, J.A.; Peters, J. Reinforcement learning in robotics: A survey. *Int. J. Robot. Res.* **2013**, *32*, 1238–1274. [CrossRef]
2. Li, Y. Deep reinforcement learning: An overview. *arXiv* **2017**, arXiv:1701.07274.
3. Dömel, A.; Kriegel, S.; Kaßecker, M.; Brucker, M.; Bodenmüller, T.; Suppa, M. Toward fully autonomous mobile manipulation for industrial environments. *Int. J. Adv. Robot. Syst.* **2017**, *14*. [CrossRef]
4. Nassal, U.; Damm, M.; Lüth, T. A mobile platform supporting a manipulator system for an autonomous robot. In Proceedings of the 5th World Conference on Robotics Research, Cambridge, MA, USA, 27–29 Spetember 1994.
5. Siciliano, B.; Khatib, O. *Springer Handbook of Robotics*; Springer: Berlin, Germany, 2016.
6. Padois, V.; Fourquet, J.Y.; Chiron, P. From robotic arms to mobile manipulation: On coordinated motion schemes. In *Intelligent Production Machines and Systems*; Elsevier: Amsterdam, The Netherlands, 2006; pp. 572–577.
7. Tan, J.; Xi, N.; Wang, Y. Integrated task planning and control for mobile manipulators. *Int. J. Robot. Res.* **2003**, *22*, 337–354. [CrossRef]
8. Berntorp, K.; Arzén, K.E.; Robertsson, A. Mobile manipulation with a kinematically redundant manipulator for a pick-and-place scenario. In Proceedings of the 2012 IEEE International Conference on Control Applications (CCA), Dubrovnik, Croatia, 3–5 October 2012; pp. 1596–1602.
9. Meeussen, W.; Wise, M.; Glaser, S.; Chitta, S.; McGann, C.; Mihelich, P.; Marder-Eppstein, E.; Muja, M.; Eruhimov, V.; Foote, T.; et al. Autonomous door opening and plugging in with a personal robot. In Proceedings of the 2010 IEEE International Conference on Robotics and Automation (ICRA), Anchorage, AK, USA, 3–8 May 2010; pp. 729–736.
10. Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv* **2017**, arXiv:1712.01815.
11. Li, J.; Monroe, W.; Ritter, A.; Galley, M.; Gao, J.; Jurafsky, D. Deep reinforcement learning for dialogue generation. *arXiv* **2016**, arXiv:1606.01541.
12. Yoo, S.; Yun, K.; Choi, J.Y.; Yun, K.; Choi, J. Action-Decision Networks for Visual Tracking with Deep Reinforcement Learning. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017.
13. Levine, S.; Pastor, P.; Krizhevsky, A.; Ibarz, J.; Quillen, D. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *Int. J. Robot. Res.* **2018**, *37*, 421–436. [CrossRef]
14. Popov, I.; Heess, N.; Lillicrap, T.; Hafner, R.; Barth-Maron, G.; Vecerik, M.; Lampe, T.; Tassa, Y.; Erez, T.; Riedmiller, M. Data-efficient deep reinforcement learning for dexterous manipulation. *arXiv* **2017**, arXiv:1704.03073.
15. Quillen, D.; Jang, E.; Nachum, O.; Finn, C.; Ibarz, J.; Levine, S. Deep Reinforcement Learning for Vision-Based Robotic Grasping: A Simulated Comparative Evaluation of Off-Policy Methods. In Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, QLD, Australia, 21–25 May 2018.

16. Gu, S.; Holly, E.; Lillicrap, T.; Levine, S. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 29 May–3 June 2017; pp. 3389–3396, doi:10.1109/ICRA.2017.7989385. [CrossRef]
17. Chen, Y.F.; Everett, M.; Liu, M.; How, J.P. Socially aware motion planning with deep reinforcement learning. In Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 24–28 September 2017; pp. 1343–1350.
18. Tai, L.; Paolo, G.; Liu, M. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 24–28 September 2017; pp. 31–36.
19. Peng, X.B.; Berseth, G.; Yin, K.; Van De Panne, M. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Trans. Graph. (TOG)* **2017**, *36*, 41. [CrossRef]
20. Heess, N.; Sriram, S.; Lemmon, J.; Merel, J.; Wayne, G.; Tassa, Y.; Erez, T.; Wang, Z.; Eslami, A.; Riedmiller, M.; et al. Emergence of locomotion behaviours in rich environments. *arXiv* **2017**, arXiv:1707.02286.
21. Breyer, M.; Furrer, F.; Novkovic, T.; Siegwart, R.; Nieto, J. Flexible Robotic Grasping with Sim-to-Real Transfer Based Reinforcement Learning. *arXiv* **2018**, arXiv:1803.04996.
22. Stulp, F.; Fedrizzi, A.; Beetz, M.; Autonomous, I.; Group, S. Learning and Performing Place-Based Mobile Manipulation. In Proceedings of the 2009 IEEE 8th International Conference on Development and Learning, Shanghai, China, 5–7 June 2009; pp. 1–7.
23. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An introduction*; MIT press: Cambridge, UK, 1998; Volume 1.
24. Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; Ng, A.Y. ROS: An open-source Robot Operating System. In Proceedings of the ICRA Workshop on Open Source Software, Kobe, Japan, May 17 2009; VoLume 3, p. 5.
25. Dhariwal, P.; Hesse, C.; Klimov, O.; Nichol, A.; Plappert, M.; Radford, A.; Schulman, J.; Sidor, S.; Wu, Y. OpenAI Baselines. 2017. Available online: https://github.com/openai/baselines (accessed on 18 January 2019).
26. Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; Openai, W.Z. OpenAI Gym. *arXiv* **2016**, arXiv:1606.01540.
27. Zamora, I.; Gonzalez Lopez, N.; Vilches, V.M.; Hernández Cordero, A.; Robotics, E. Extending the OpenAI Gym for Robotics: A Toolkit for Reinforcement Learning Using ROS and Gazebo. *arXiv* **2017**, arXiv:1608.05742v2.
28. Arulkumaran, K.; Deisenroth, M.P.; Brundage, M.; Bharath, A.A. A brief survey of deep reinforcement learning. *arXiv* **2017**, arXiv:1708.05866.
29. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.
30. Schulman, J.; Levine, S.; Moritz, P.; Jordan, M.I.; Abbeel, P. Trust Region Policy Optimization. *arXiv* **2015**, arXiv:1502.05477.
31. Kullback, S.; Leibler, R.A. On information and sufficiency. *Ann. Math. Stat.* **1951**, *22*, 79–86. [CrossRef]
32. Schulman, J.; Moritz, P.; Levine, S.; Jordan, M.; Abbeel, P. High-dimensional continuous control using generalized advantage estimation. *arXiv* **2015**, arXiv:1506.02438.
33. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous Control With Deep Reinforcement Learning. *arXiv* **2016**, arXiv:1509.02971v5.
34. Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; Riedmiller, M. Deterministic Policy Gradient Algorithms. In Proceedings of the 31st International Conference on Machine Learning (ICML 2014), Beijing, China, 21–26 June 2014.
35. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef]
36. Uhlenbeck, G.E.; Ornstein, L.S. On the theory of the Brownian motion. *Phys. Rev.* **1930**, *36*, 823. [CrossRef]
37. Plappert, M.; Houthooft, R.; Dhariwal, P.; Sidor, S.; Chen, R.Y.; Chen, X.; Asfour, T.; Abbeel, P.; Openai, M.A. Parameter Space Noise for Exploration. *arXiv* **2018**, arXiv:1706.01905v2.

38. Todorov, E.; Erez, T.; Tassa, Y. Mujoco: A physics engine for model-based control. In Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vilamoura, Portugal, 7–12 October 2012; pp. 5026–5033.
39. Chitta, S.; Sucan, I.; Cousins, S. Moveit![ROS topics]. *IEEE Robot. Autom. Mag.* **2012**, *19*, 18–19. [CrossRef]
40. Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. TensorFlow: A System for Large-Scale Machine Learning. In Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI), Savannah, GA, USA, 2–4 November 2016; Volume 16, pp. 265–283.
41. Ba, J.L.; Kiros, J.R.; Hinton, G.E. Layer Normalization. *arXiv* **2016**, arXiv:1607.06450.
42. Ng, A.Y. Feature selection, L 1 vs. L 2 regularization, and rotational invariance. In Proceedings of the Twenty-First International Conference on Machine Learning, Banff, AB, Canada, 4–8 July 2004; ACM: New York, NY, USA, 2004; p. 78.

© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).

8

Affordance-Based Grasping Point Detection Using Graph Convolutional Networks for Industrial Bin-Picking Applications

Authors

Iriondo, Ander; Lazkano, Elena; Ansuategi, Ander;

Publisher

Multidisciplinary Digital Publishing Institute (MDPI)

Journal

Sensors

Year

2021

Quartile (Web of Science / Scimago)

Q2/Q1

DOI

https://doi.org/10.3390/s21030816

sensors

MDPI

Article

Affordance-Based Grasping Point Detection Using Graph Convolutional Networks for Industrial Bin-Picking Applications

Ander Iriondo [1,*], Elena Lazkano [2] and Ander Ansuategi [1]

[1] Department of Autonomous and Intelligent Systems, Fundación Tekniker, Iñaki Goenaga, 5-20600 Eibar, Spain; ander.ansuategi@tekniker.es
[2] Computer Science and Artificial Intelligence (UPV/EHU), Pº Manuel Lardizabal, 1-20018 Donostia-San Sebastián, Spain; e.lazkano@ehu.eus
* Correspondence: ander.iriondo@tekniker.es

Abstract: Grasping point detection has traditionally been a core robotic and computer vision problem. In recent years, deep learning based methods have been widely used to predict grasping points, and have shown strong generalization capabilities under uncertainty. Particularly, approaches that aim at predicting object affordances without relying on the object identity, have obtained promising results in random bin-picking applications. However, most of them rely on RGB/RGB-D images, and it is not clear up to what extent 3D spatial information is used. Graph Convolutional Networks (GCNs) have been successfully used for object classification and scene segmentation in point clouds, and also to predict grasping points in simple laboratory experimentation. In the present proposal, we adapted the Deep Graph Convolutional Network model with the intuition that learning from *n*-dimensional point clouds would lead to a performance boost to predict object affordances. To the best of our knowledge, this is the first time that GCNs are applied to predict affordances for suction and gripper end effectors in an industrial bin-picking environment. Additionally, we designed a bin-picking oriented data preprocessing pipeline which contributes to ease the learning process and to create a flexible solution for any bin-picking application. To train our models, we created a highly accurate RGB-D/3D dataset which is openly available on demand. Finally, we benchmarked our method against a 2D Fully Convolutional Network based method, improving the top-1 precision score by 1.8% and 1.7% for suction and gripper respectively.

Keywords: affordance grasping; grasping point detection; graph convolutional network; pick and place; deep learning

check for updates

Citation: Iriondo, A.; Lazkano, E.; Ansuategi, A. Affordance-Based Grasping Point Detection Using Graph Convolutional Networks for Industrial Bin-Picking Applications. *Sensors* **2021**, *21*, 816. https://doi.org/10.3390/s21030816

Academic Editor: Kourosh Khoshelham
Received: 26 November 2020
Accepted: 21 January 2021
Published: 26 January 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Pick and place are basic operations in most robotic applications, whether in industrial setups (e.g., machine tending, assembling or bin-picking) or in service robotics domains (e.g., agriculture or home). Picking and placing is a mature process in structured scenarios. Nevertheless, it is not the case in less structured industrial environments or when parts with higher degree of variability have to be manipulated. The market demands more flexible systems that will allow for a reduction of costs in the supply chain, increasing the competitiveness for manufacturers and bringing a cost reduction for consumers. The introduction of robotic solutions for picking in unstructured environments requires the development of flexible robotic configurations, robust environment perception, methods for trajectory planning, flexible grasping strategies and human-robot collaboration. Such robotics solutions have not reached the market yet and remain as laboratory prototypes due to the lack of efficiency, robustness and flexibility of currently available manipulation and perception technologies.

The grasping point detection is one of the robotic areas that most attention has attracted since early years of robotic manipulation. The problem of detecting an appropriate grasping

point consists of looking for the best object picking location (position and orientation) depending on the type of end effector available. The complexity of the grasping point detection depends on the arrangement of the objects to be handled, and the casuistry varies depending on the structure of the arrangement:

- Structured: the parts to be picked are organized inside the bin and follow an easily recognizable pattern.
- Semi-structured: some predictability exists in the way parts are organized and the items are relatively well separated.
- Random: the parts are randomly organized inside the bin and they do not follow any pattern.

Noticeably, the less the structure of the arrangement, the higher the complexity of the grasping point detection.

The grasping point detection has been traditionally tackled as a computer vision detection problem. In the past, hand-designed features were used to first recognize the object and then to perform model based grasp planning. However, although those methods are robust with small subsets of known objects, they lack of flexibility and are time-consuming. Unlike some other traditional computer vision problems, these detection methods are used on-line and to meet cycle-time restrictions, computational speed is required. The work presented here focuses on flexible, safe and dependable part-handling in industrial environments. In such dynamical areas as warehouses or distribution centres, a huge number of items need to be handled, usually in unstructured scenarios [1].

Traditional grasping point detection methods, even though being efficient in very specific tasks, have proven to be inefficient in tasks with uncertainty and more flexible solutions are demanded. As stated by Kober and Peters in [2], hard coded behaviours are restricted to the situations that the programmer considered and are impractical in dynamic scenarios.

Recently, Deep Learning (DL) based methods have gained popularity due to the affordable price and increasing computation capability of devices such as GPUs and TPUs. DL based methods have shown state-of-the-art performance in a wide range of computer vision, audio and natural language processing problems and it has also been successfully applied to detect grasping points in more complex scenarios [3]. First approaches applied DL to identify randomly placed objects in the scene, to later heuristically obtain grasping points, based on the 3D information of the identified object [4]. However, contemporary approaches do not rely on the identity of the objects and apply DL to predict object affordances. The affordance-based grasping point detection has gained attention due to its capability to learn from the shape, colour and texture of the objects, without relying on part identity. Thus, those algorithms are able to generalize to never seen objects, providing flexible grasping solutions [5].

Yet, most of the DL based grasping point detection algorithms predict grasping points in 2D data, typically images obtained from both RGB or RGB-D cameras, losing the 3D spatial information. Recent approaches use Graph Convolutional Networks (GCNs) to learn geometric features directly in point clouds, and have been successfully applied to predict grasping points [6,7]. Although these methods suffer from computational cost due to the spatial operations applied to the input data, they have shown improved learning capabilities in non-Euclidean data. To alleviate the computational cost, the input clouds are usually sub-sampled both to meet hardware restrictions (e.g., GPU and memory) and to get reasonable computation speeds. In addition to the spatial coordinates of the points in the cloud, these algorithms are able to learn from n-dimensional points and, therefore, spatial features such as surface normals can be included in the training data.

The main goal of the work presented here is to analyse whether the usage of n-dimensional point clouds and GCNs contributes to better learn object affordances for suction and gripper in a random bin-picking application. The contributions of the paper are as follows:

1. We designed a method based on GCNs to predict object affordances for suction and gripper end effectors in a bin-picking application. This method includes, an adaptation to a point cloud scene segmentation algorithm based on GCNs to predict affordance scores in n-dimensional point clouds, and a bin-picking oriented data preprocessing pipeline.
2. We created a highly accurate dataset of random bin-picking scenes using a Photoneo Phoxi M camera, which is openly available on demand.
3. We benchmarked our GCN based approach with the one presented in [5], which uses 2D Fully Convolutional Networks to predict object affordances.

The rest of the paper is structured as follows: Section 2 reviews the literature. Section 3 presents the details of the generated dataset and introduces the 3D affordance grasping approach. Additionally, the evaluation procedure and the used metrics are also detailed. The details of the implementation are explained in Section 4. Finally, the obtained results and the conclusions are presented in Sections 5 and 6 respectively.

2. Literature Review

The problem of finding a suitable grasping point among an infinite set of candidates is a challenging and yet unsolved problem. There are many approaches and a huge variety of methods that try to optimize approximate solutions. According to Sahbani et al. these methods can be categorized as analytic or data-driven [8].

Traditionally, analytical approaches have been used to develop robotic applications to solve specific tasks, mainly implemented with rules based on expert knowledge [9]. Those algorithms are based on kinematic or dynamic models, and the grasping point detection is formulated as a constrained optimization problem [10–12]. Even though the analytical approaches are effective, they tend to be very task specific and time-consuming.

Data-driven techniques have proven to overcome the mathematical complexity of the existing analytical methods that solve the grasping point detection problem [13]. These alternative techniques focus more on extracting object representations and are implemented as learning and classification methods. Therefore, grasp representations are learned from experience and used to retrieve correct grasps in new queries. Besides, the parameterization of the grasp is less specific and thus, data-driven algorithms are more robust to uncertainties in perception [14–16]. Nevertheless, data-driven methods usually need a big number of annotated data, which usually implies a time-consuming annotation process.

In [13], Bohg et al. split data-driven grasping approaches into three sub-categories, depending on the prior knowledge about the query object:

1. Known objects: The query objects have been previously used to generate grasping experience.
2. Familiar objects: The query objects are similar to the ones used to generate grasping experience. This approaches assume that new objects are grasped similar to old ones.
3. Unknown objects: Those approaches do not assume to have prior grasping experience related to the new object.

When the objects to be manipulated are known and the number of references to be handled is small, a database with 3D objects and predefined grasping candidates is usually used (e.g., created using GraspIt! [17]). First, a pose estimation algorithm is applied to locate the object in the scene, using visual and geometric similarity [18–21]. Then, grasping candidates are filtered due to collision and reachability issues and the best one is selected, after being ranked by some quality metrics. In spite of being very robust handling one or few object references, such ad-hoc solutions fail when the number of references increases (e.g., due to the unavailability of thousands of 3D models of parts or the computational cost to find such a high number of references in the scene).

However, in industrial setups with highly unstructured environments, changing conditions and highly variable or even unknown multi-reference parts, more flexible methods are needed. To tackle the shortcomings of model based algorithms, most recent

approaches use deep neural networks (DNNs) to map robotic sensor readings to labels annotated either by a human or by a robot [22].

In works such as [23–25], authors use Convolutional Neural Networks (CNNs) to identify and segment objects in 2D images of the scene. Then, per each identified object, predefined grasping points are used to pick them. Nevertheless, techniques based on the object identification fail to deal with high number of references and novel objects.

Instead of relying on the object identification, other approaches try to predict grasping points focusing on the shape, colour and texture of the parts. In [26], authors created a grasping dataset with both successful and unsuccessful grasps, to later map RGB-D images with graspable regions using CNNs. In [27], the aforementioned dataset was used to create real-time grasps using CNNs. Similar to the previous methods, also in [28–30], the grasp was represented as a rectangle over the 2D input images that indicated the point, orientation and the opening of the gripper. In these applications, the grasping pose detector CNN worked in an sliding window manner, performing an inference per detected object in the scene. Thus, the computational complexity depended on the number of objects in the scene. In spite of the fact that the aforementioned methods deal correctly with scattered objects and show good generalization capabilities with never seen objects, they fail to predict grasping points in cluttered scenes [22].

As an alternative to codifying the grasp as a rectangle, other approaches try to predict pixel-wise affordances. For instance, in one of the first attempts, Detry et al. were able to learn grasping affordance models implemented as continuous density functions [31]. In this case, the affordances were learned letting a robot pick and drop objects. However, the learned models were specific to a small set of known objects. Recent methods use Fully Convolutional Networks (FCNs) [32] to learn pixel-wise affordances. Nguyen et al. first applied DL to predict real-time affordances in RGB-D images, to deal with a small set of known objects [33]. In a more recent approach, Zeng et al. used DL to predict pixel-wise affordances for a multifunctional gripper. The predicted grasping scores were used to choose the best action of a predefined set of 4 actions: suction down, suction side, grasp down and flush grasp [5]. Same authors also developed a robotic system that was able to learn to grasp and push objects in a reinforcement learning (RL) setting [34]. In this case, FCNs were used to model affordance based policies. More recently Zeng et al. also made use of FCNs to encode affordance based policies to learn complex behaviours such as picking and throwing objects through the interaction with the environment [35]. In spite of the fact that these FCN based models have shown to be capable to learn object affordances, they can not cope with non-euclidean data such as point clouds. Therefore, the feature extraction is performed uniquely in the RGB/RGB-D images and do not take advantage of the 3D spatial information.

Rather than learning to predict optimal grasping points, in other works authors try to learn visuomotor control policies to directly predict robot actions to pick objects, avoiding the need for an additional robot controller. For instance, Mahler and Goldberg were able to learn a deep policy to pick objects in cluttered scenes in [36]. In [37] authors trained a large CNN to predict the probability of success of task space motions of the gripper, only using RGB images. The learning process was carried out with 14 real robotic manipulators gathering experience in two months of continuous execution. However, the cost of getting experience in reality makes the solution hardly transferable to the industry. Similar to the proceeding proposed by Levine et al., in [38] Kalashnikov et al. used RL to learn high accurate control policies to pick objects in cluttered scenes, also using RGB images. In this approach also the experience of multiple real robots was used to optimize the policy neural network. To avoid the cost of real setups with several robots and to decrease the time to acquire experience, in [39] James et al. proposed to learn the visuomotor control policies in simulation. To reduce the simulation to reality gap, the training process is usually carried out with domain randomization [40].

The aforementioned methods extract grasping features in RGB/RGB-D images, and therefore only a single point of view of the scene is usually used in the learning process.

Even if traditional CNNs are able to learn features in euclidean data (e.g., RGB-D images, projections or voxels), they fail to deal with non-euclidean unordered data types such as graphs, were connections between nodes can vary. Recently, Graph Convolutional Networks (GCNs) [41] have gained popularity due to their ability to learn representations over graph-structured data, specially in non-euclidean data such as meshes and point clouds. Similar to 2D CNNs, those models have been used for classification, scene segmentation and instance segmentation [42], particularly in 3D point clouds. As suggested in [43,44], traditional and most used GCNs usually are very shallow models, due to the over-smoothing and over-fitting problem of deep GCNs. Recent works as [45–47] try to mitigate these problems introducing several changes in the traditional GCN based model architectures.

GCNs have been successfully applied to solve the grasping point detection problem. Liang et al. used the PointNet [6] network to infer the quality of grasping candidates for a gripper end effector, after applying an antipodal grasp sampling algorithm [48]. Although this technique showed good generalization capability with novel objects, the grasp sampling process was time-consuming due to the need to infer the quality of each proposal. Besides, only local features around the grasp were used to infer the grasp quality, which did not take into account the global object distribution and occlusions that happen in cluttered scenes. Ni et al. proposed a single-shot grasp proposal network for a gripper end effector, based on PointNet++ [7], to avoid the time-consuming grasp candidate sampling [49]. Furthermore, a simulation-based automatic dataset generation proceeding was proposed, using Ferrari and Canny metrics [12]. In spite of the fact that authors were able to predict grasping points also in novel objects, complete 3D models of the objects were used to generate the training dataset in simulation, which are hard to obtain in setups where a big number of references have to be handled. In addition, the grasping dataset was created with single objects, without taking into account global factors in the scene such as occlusions and entanglements. PointNet++ was also used in [50] to implement a single-shot grasp proposal network for a gripper end effector. Although at inference time a single-view point cloud of the scene was used, to train the network, Qin et al. used a simulation based method, that also depends on the availability of 3D models of the parts. In this work also the grasping candidates were generated analytically in single objects and not in cluttered scenes. Although unfeasible grasp proposals are discarded after checking the collisions in cluttered scenes, the global arrangement of the objects is also not taken into account when the grasping dataset is generated.

All the reviewed GCN based methods focused their work only on the gripper end effector and did not consider the grasping point generation for suction. In addition, none of them took into account the global arrangement of the objects when the grasping database was generated. Most of them used simulation to generate the dataset, where it is not straightforward to take into account these global scene factors. Furthermore, none of the reviewed works proposed a bin-picking oriented method.

Generative approaches have also been updated to deal with 3D data, and applied to generate grasping points in point clouds. Mousavian et al. presented a variational auto-encoder which used PointNet++ to implement both the encoder and the decoder [51]. In addition, authors also implemented a PointNet++ based evaluator network to assess the generated grasp candidates. However, this technique dealt with objects with relatively few variability and not in heavy cluttered scenes.

In the presented work, we propose to learn a single-shot affordance-based grasping point detector using GCNs for suction and gripper end effectors, avoiding the need of complex grasp candidate sampling methods. To the best of our knowledge, this is the first time that GCNs are applied to predict affordances both for suction and gripper end effectors in a bin-picking application. Contrary to the general approach in the literature, we propose not to use 3D object models to generate the dataset in simulation, but to use real world scenes instead. Following that idea, we avoid the simulation to reality gap that commonly happens in models trained with synthetically generated data. To annotate the dataset, we take into account global scene factors such as occlusions (e.g., partially

occluded objects are graspable or not depending on the weight of the objects that covers them), object entanglements, etc. Those are difficult to take into account in simulation but crucial in real applications. With that purpose, we have created a highly accurate dataset in real random bin-picking scenes using the industrial 3D Photoneo Phoxi M camera [52]. Although such expensive cameras are not widespread for laboratory level applications, the high accuracy they offer is vital in industrial bin-picking setups. Furthermore, we show that GCNs are able to converge with a relatively small training dataset of n-dimensional point clouds using data augmentation.

3. Problem Specification and Setup

The random bin-picking problem can be divided into the following categories:

1. Mono-reference: the objects are randomly placed but belong to the same reference, which is usually known (e.g., Figure 1a).
2. Multi-reference: the objects are randomly placed and belong to multiple references. The number of references can be high and novel objects may appear in the bin (e.g., Figure 1b).

(**a**) Mono-reference bin-picking. (**b**) Multi-reference bin-picking.

Figure 1. Random bin-picking scenarios.

In mono-reference applications, where the parts to be handled are known and their models can be easily obtained, 3D model matching algorithms are typically used to estimate the 6-DoF pose of the parts in the scene [53]. However, the multi-reference random bin-picking is an unsolved problem yet essential for flexible/efficient solutions.

Our work is focused on the context of multi-reference bin-picking. DL based algorithms have been widely used in the literature to handle uncertainties in the scene and novel objects. Particularly, methods that predict object affordances have shown to be robust to uncertainties and have a good generalization capability. However, most of them are only able to handle 2D data, and do not take into account 3D spatial features of the parts/scene. Recently, Zeng et al. won the Amazon Robotics Challenge (ARC) with a bin-picking method that was based on affordances [5]. Authors faced the multi-reference random bin-picking as a 2D scene segmentation problem and were able to get pixel-wise affordance scores. Based on that work, our intuition was that learning directly in n-dimensional point clouds would lead to a performance boost in the grasping point detection.

With that aim, we selected the Deep GCN model proposed in [46] where some novelties were introduced that led to a very deep GCN based model. Although originally it was implemented for segmentation purposes, the adaptation we introduced allows us to obtain affordance scores in bin-picking scenes, following the idea of Zeng et al. In addition, we followed a bin-picking oriented pipeline, which made the learned GCNs applicable to other picking scenarios, as it is later on explained in Section 3.2.2.

For evaluation purposes, we have created a dataset with multiple annotated bin-picking scenes. The dataset was annotated once and we used it to train and test both methods (FCNs and GCNs). The details of the dataset and the annotations are explained in Section 3.2. Finally, in order to make a fair comparison, the metrics defined in [5] have been used to measure the deep models. These are further explained in detail in Section 3.4.

3.1. Multi-Functional Gripper

In the context of the Pick-Place European project [54], a multi-functional gripper has been developed by Mondragon Assembly (Figure 2). This multi-functional gripper is composed of the following retractable end effectors: A magnet, a suction cup and a two finger gripper. All of them are equipped with tactile sensors developed by Fraunhofer—IFF that are designed to softly handle any kind of product.

Figure 2. Multi-functional gripper attached to the UR robot.

Although in this work the gripper was not directly used to annotate the dataset, it was important to define its particularities since the annotation of the dataset highly depends on these specifications. In this way, the graspability of the objects was tested individually with each end effector. For instance, the quality of a grasping point can drastically change depending on the maximum width between the fingers of the gripper. In this work, both, suction and gripper end effectors were considered.

3.2. Dataset

In industrial bin-picking applications, the perception of the scene must be as accurate as possible, due to the precision needed to handle objects with variable shape, colour and texture. At the same time, industrial 3D cameras are becoming more and more robust against changing environmental conditions. In a first attempt, we chose to profit from the dataset provided by [5]. However, the bin localization was not available for the suction and, preliminary results showed that the depth data was not accurate enough due to the location of the camera. Thus, this option was discarded. Instead, we opted to use a Photoneo Phoxi M camera to generate our dataset, which has been widely used in many bin-picking applications and has demonstrated a great performance [55,56]. We positioned the camera overhead the bin to reduce occlusions. An example of a captured scene and its corresponding point cloud are depicted in Figure 3. To generate the dataset, we used a set of 37 rigid and semi-rigid opaque parts that were selected in the context of the Pick-Place EU project. In spite of the fact that we included some transparent and shiny objects, we opted to focus our analysis on opaque parts.

(**a**) RGB image of the scene. (**b**) Point cloud of the scene.

Figure 3. Dataset sample scene.

As a result, the dataset was composed of the following elements per each scene:

- Intrinsic parameters of the camera.
- RGB-D images of the scene. Using the intrinsic parameters, images can be easily transformed to a single-view point cloud.
- Bin localization with respect to the camera.
- Point cloud of the scene.

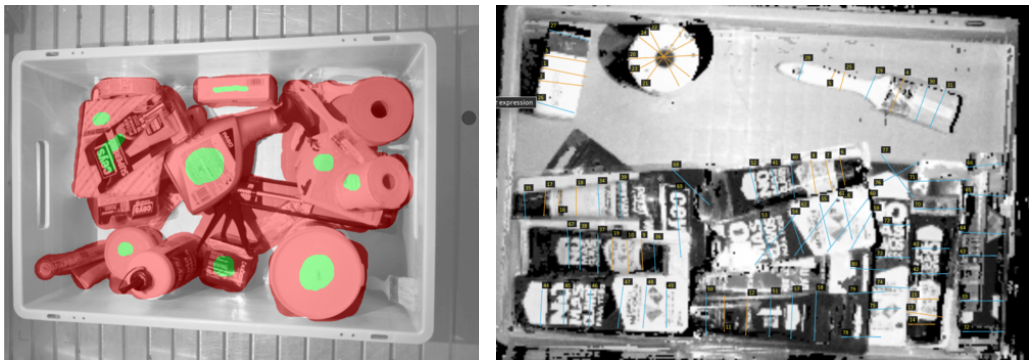
As RGB-D images can be easily converted to single-view point clouds, the annotation process was carried out only in RGB-D data.

3.2.1. Annotation

The performance of the learned models highly depends on the quality of the annotated data. Although simulation based dataset annotation methods are widely used, we believe that the experience and the criteria of the human is vital to analyze the particular casuistry of each scene. To overcome this burden, we created a small dataset composed of 540 scenes with randomly placed multi-reference parts. We followed the annotation style proposed in [5] and we labeled the acquired scenes twice, for suction and gripper end effectors respectively. The labeling process was made taking into account the restrictions of the multi-functional gripper presented in Section 3.1. We physically used the multi-functional gripper to extract the good and bad grasping areas in individual parts and applied this knowledge to manually annotate complex scenarios. The manual annotation lets us take into account weight distributions, object entanglements and complex situations in general that are difficult to consider in simulation. In the application presented by [5], authors tackled the grasping point detection as a scene segmentation problem, being able to predict pixel-wise affordances. For that purpose, the RGB-D dataset was annotated pixel-wise. The main particularities of each type of annotation are described below:

Suction Annotations

For the suction end effector, a grasping point is defined by a 3D contact point in the scene along with its normal vector. Here, the normal vector determines the orientation of the suction end effector. As depicted in Figure 4a, the pixels that belong to objects and are affordable for the suction tool, were annotated with green colour. The pixels that belong to objects but are not good grasping points, were annotated with red colour. Finally, the rest of the points that belonged to the scene were annotated as neutral. Suction annotations were stored as masks, and the value of each pixel of the mask indicated the class of the corresponding pixel in the original image. The annotation process was carried out using the Pixel Annotation Tool [57].



(**a**) Suction annotation example. (**b**) Gripper annotation example.

Figure 4. Annotation examples in RGB-D data.

Gripper Annotations

A gripper's grasping point is defined by:

1. A 3D point in the space that indicates the position where the center between the fingers of the gripper has to be placed.
2. The orientation of the gripper in the vertical axis.
3. The opening of the fingers.

The RGB-D scenes were transformed into orthographic heightmaps, and top views of the scenes were obtained, due to the fact that only vertical gripper actions were taken into account. For that purpose, first we located the bin with a 3D model matching algorithm, and we transformed the clouds into the bin coordinate frame. Then, the points that laid outside the bin were discarded and the resulting cloud was projected orthographically and stored again as RGB-D image.

As it can be seen in Figure 4b, grasps were annotated with straight lines. Good and bad grasps are represented with orange and blue colours respectively. Following that approach, only hard negative grasping points were annotated as bad. The center of the line indicates the 3D point in the space where the gripper should move to, and the orientation of the line with respect to the horizontal axis of the image indicates the grasping angle. The opening of the gripper fingers is computed on-line during execution, taking into account the local geometry of the grasping area.

Similar to the annotation proceeding proposed in [5], the annotations made with lines were converted to pixel-wise labels. We only took into account rotations in the z axis to only perform vertical grasps. For the sake of simplifying the problem, the z axis was divided into n fixed angles, in our case $n = 16$. To decide to which discrete angle each annotated line belonged to, the angle θ between each line and the horizontal axis of the heightmap was computed. Thus, for each annotated RGB-D scene for the gripper, 16 pixel-wise annotations were obtained (one mask per angle). The transformation from the line annotations into masks was done in the following way:

1. The θ angle between each line and the horizontal (y) axis indicates the discrete n orientation in z axis that the annotation belongs to.
2. The scene RGB-D image is rotated n times with an increment of $360/n$ with respect to the z axis of the bin.
3. The center pixel of each annotated line is computed and included in the corresponding annotation mask among the n possible orientations.

To annotate the dataset for the gripper, we modified the VGG Image Annotator to take into account the angle of each annotation with respect to the horizontal axis of the heightmap (y) [58].

3.2.2. RGB-D Annotations to 3D Point Clouds

In this section are detailed the steps carried out to transform RGB-D annotated data into 3D annotated point clouds for both, suction and gripper effectors.

Suction

Our goal was to follow traditional preprocessing pipelines proposed in bin-picking applications, with the next requirements: To ease the learning process of grasping points in 3D point clouds and to create a generic solution for any bin-picking application. For that purposes, we designed the pipeline showed in Figure 5 and afterwards we used it to transform the RGB-D dataset into a single-view point cloud dataset. Basically, we aimed to abstract from the camera pose in the scene by always working in the bin coordinate frame. This also gives the possibility to work with multiple extrinsically calibrated cameras.

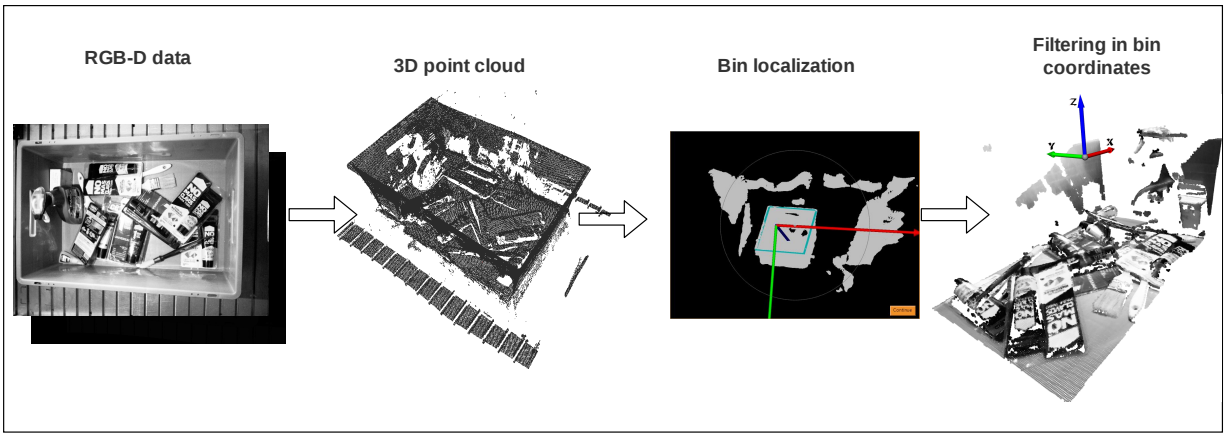


Figure 5. Data preprocessing pipeline.

Conforming to the pipeline, first, the RGB-D images were transformed into point clouds using the camera intrinsics matrix. Then, the pose of the bin was obtained, using a 3D model matching algorithm previously developed at Tekniker [1]. As the dimensions of the bin were known, the bin localization algorithm created the 3D model of the top edges of the bin and matched it to the obtained point cloud of the scene, to finally obtain its pose. For that purpose the Mvtec Halcon [59] library was used. Finally, the clouds were transformed into the new coordinate frame and the points that laid outside the bin were filtered using the bin size. In this case, all the tests were made with a bin size of $length = 600$ mm, $width = 400$ mm and $height = 230$ mm. Additionally, the standard voxel downsampling algorithm offered by the Open3D [60] library was used, with voxel size 0.002 m, followed by a statistical outlier removal from the same library.

Alongside the training data, also the annotations needed to be transformed into 3D. Figure 6 shows the original (Figure 6a) and the transformed (Figure 6b) annotations for the suction end effector. The output of the transformation was a semantically annotated 3D point cloud where: green, blue and pink pixels belong to good, bad and neutral grasping point classes, respectively. In this case, the annotation transformation into 3D was straightforward, and point classes were assigned depending on which class the original RGB-D pixel belonged to.

(**a**) Suction annotation in RGB-D data.

(**b**) Same scene converted to semantically annotated point cloud.

Figure 6. Annotation transformation for the suction.

Gripper

Contrary to the followed procedure for the suction, in this case, the annotated RGB-D orthographic heightmap was already transformed into the bin coordinate frame and, thus, the conversion of RGB-D images to 3D point clouds in the bin coordinate frame was straightforward. As well as from each annotated RGB-D image a labeled point cloud was obtained for suction, in the case of the gripper, instead, n point clouds were obtained. As explained before, we only took into account rotations in the z axis to only perform vertical grasps. The transformation from 2D annotations into 3D was made in the following way:

1. The θ angle between each line and the y axis (horizontal axis of the orthographic heightmap) indicates to which discrete n orientation in the z axis the annotation belongs to.
2. The scene point cloud was rotated n times with an increment of $360/n$ with respect to the z axis of the bin.
3. The 3D coordinate of the centre pixel of each annotated line was computed in the corresponding point cloud among the n rotated clouds.
4. The length of each line indicates the radius of the circumference around the centre pixel that was annotated.

An example of the gripper annotation conversion to 3D can be seen in Figures 7 and 8. On the one hand, the annotated RGB-D orthographic heightmap can be seen in Figure 7. On the other hand, two of the 16 generated annotated clouds are depicted in Figure 8. Specifically, $n = 6$ and $n = 12$ discrete orientations in the z axis have been selected for illustration purposes. In these examples the y axis (green) indicates the orientation of the gripper.

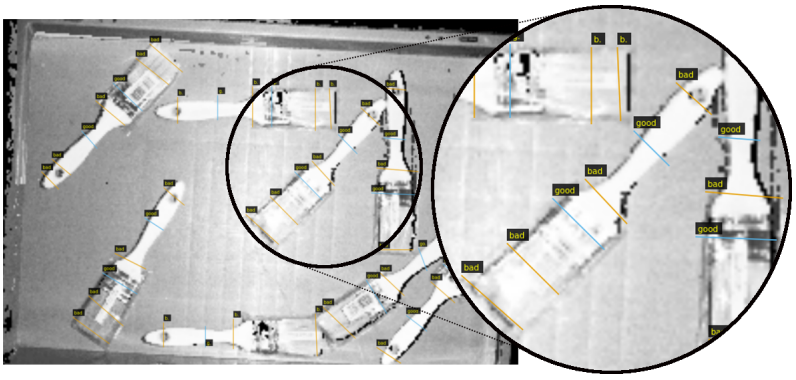
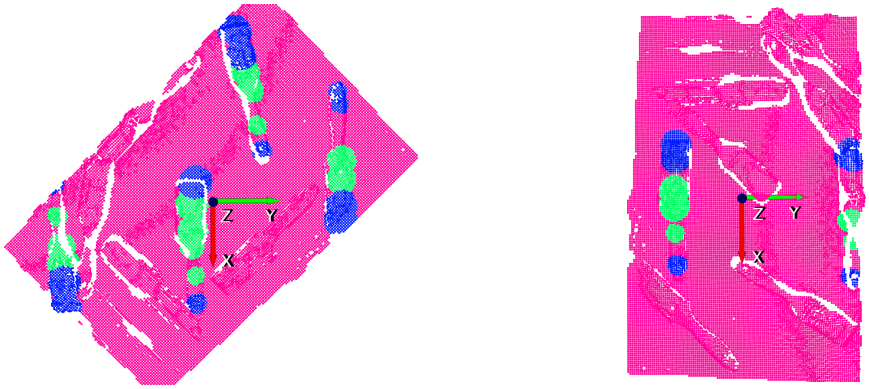


Figure 7. Gripper annotations in a RGB-D orthographic heightmap.



(**a**) $n = 6$ annotated 3D point cloud. (**b**) $n = 12$ annotated 3D point cloud.

Figure 8. Annotation transformation for gripper.

3.3. 3D Affordance Grasping with Deep GCNs

In spite of the fact that CNNs have shown strong performance with Euclidean data, this is not the case for many applications that deal with non-euclidean data. Graphs are popular data structures that are used in many applications such as social networks, natural language processing, biology or computer vision [61]. Although traditional CNNs are not able to systematically handle this kind of data, GCNs have shown to be able to overcome the shortcomings of CNNs.

A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined by an unordered set of vertices $\mathcal{V}$ and a set of edges $\mathcal{E}$ indicating the connections between vertices. Vertices are represented associating each vertex $v \in \mathcal{V}$ with a feature vector $h_v \in \mathbb{R}^D$, where D indicates the dimension of the feature vectors. So, the graph is represented concatenating the feature vectors of the set of unordered vertices $h_\mathcal{G} = [h_{v_1}, h_{v_2}, h_{v_3}, ..., h_{v_N}] \in \mathbb{R}^{DxN}$, where N indicates the number of vertices in the graph.

In GCNs, the most used operations at each layer are aggregation and update. These operations receive the input graph $\mathcal{G}_l = (\mathcal{V}_l, \mathcal{E}_l)$ and they output the graph $\mathcal{G}_{l+1} = (\mathcal{V}_{l+1}, \mathcal{E}_{l+1})$ at the l-th layer. On the one hand, the aggregation function is used to collect the information of the neighbour vertices (e.g., max-pooling aggregator). On the other hand, the update function applies a transformation to the collected data by the aggregate operation, in order to generate new vertex representations (e.g., using MLPs). At each layer, the aforementioned operations are applied to each vertex $v \in \mathcal{V}_l$ and new vertex representations are generated [62].

Most GCNs have fixed graph structures and only per-vertex features are updated at each iteration. However, more recent works demonstrate that changing graph structures contribute to better learn graph representations. For instance, Wang et al. proposed a neural network module dubbed EdgeConv [63] that finds k nearest neighbours in the feature space to reconstruct the graph at each EdgeConv layer. The authors claim that this architecture is suitable for classification and segmentation tasks in point clouds.

Furthermore, as reported in recent work, traditional GCNs cannot go as deep as CNNs due to the high complexity in back-propagation and they are no more than three layers deep [43,44]. State-of-the-art works suggest multiple changes in GCN architectures to overcome the aforementioned shortcomings [45–47]. The Deep GCNs model architecture developed by Li et al. in [46] proposes multiple changes that allow to effectively learn deep GCNs that are up to 56 layers deep. Moreover, this architecture has shown great performance in semantic segmentation tasks in the S3DIS indoor 3D point cloud dataset [64].

The adaptations that allow deep GCNs are twofold:

- Dilated aggregation: Dilated k-NN is used to find dilated neighbours after every GCN layer, getting as result a Dilated Graph. Having a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with a Dilated k-NN and d dilation rate, the Dilated k-NN returns the $k \times d$ nearest neighbours in

the feature space, ignoring every d neighbours. The l_2 distance is used in the feature space.

- Residual and Dense connections: Based on the success of models with residual connections between layers such as ResNet [65], or dense connections such as DenseNet [66], these concepts have been translated to GCNs, allowing much deeper models.

The model chosen for our affordance-based grasping algorithm includes the dilated aggregation so to increase the receptive field of the GCN. Additionally, we chose the residual connections between the layers to increase the depth of the network. The selection of this network configuration was motivated by the improved performance achieved against other architectures in [46]. The used architecture is illustrated in Figure 9.

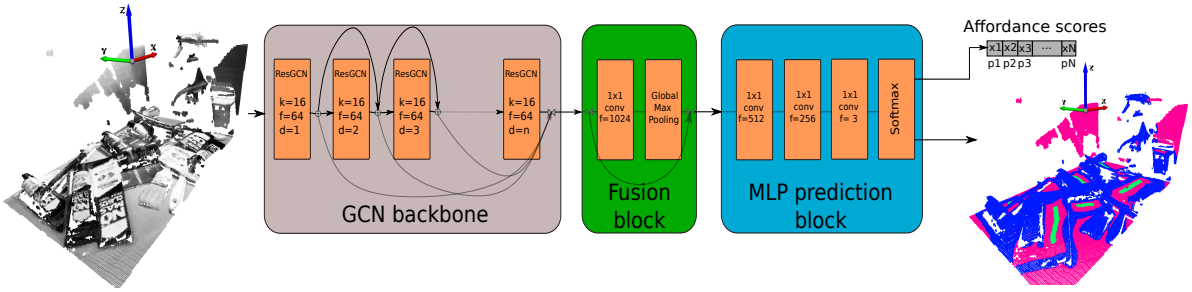


Figure 9. Used model architecture.

When the model is fed with the n-dimensional point cloud data, first the GCN backbone is in charge of extracting features from the input data. Then, the fusion block fuses the extracted features by the backbone and global features are extracted. Finally, the prediction block predicts point-wise labels. As it can be seen in Figure 9, we modified this latter block to obtain point-wise affordance scores. The output tensor of the network has the following shape: $(N_{batch}, N_{pts}, N_{classes})$, where N_{batch} indicates the number of batches fed to the network, N_{pts} is the number of points per batch, and $N_{classes}$ the number of classes. In our case $N_{classes} = 3$ (good, bad and neutral grasping points). Thus, to obtain the point-wise labels, the *argmax* operation is performed over the last axis of the output tensor. To obtain good grasping affordances, however, we only get the channel corresponding to good grasping points, as the *softmax* operation outputs the probability distribution over the three classes. In our case the affordances are predicted in the first channel of the last axis, as shown in Equation (1).

$$affordances = [N_{batch}, N_{pts}, 0] \tag{1}$$

Doing so, the points can be ordered depending on their affordance score. The higher the affordance score, the higher is the likelihood of this grasp being a success. We used the same general model architecture both for suction and gripper end effectors. Nevertheless, we selected specific hyper-parameters for each case, as it is later explained in Section 4.

3.4. Benchmark Definition and Metrics

Here we describe the tests defined to compare both FCN and GCN based methods and the metrics defined to measure the performance of each of them.

3.4.1. Benchmark Definition

To assess each model and to make a comparison between them, two test were carried out:

1. Test 1: The methods were trained with the 80% and assessed against the remaining 20% of the dataset, that was composed of scenes with known objects, but completely new object arrangements that were not used to train the model.
2. Test 2: We created a set of 100 new scenes containing randomly arranged similar but never seen objects, in order to assess the generalization capability of each model. To that end, more than 15 new parts were selected.

3.4.2. Metrics

As authors claim in [5], a method is robust if it is able to consistently find at least one suction or grasp proposal that works. Thus, the metric used to measure the robustness of the methods is the *precision* of the predictions against the manual annotations. The precision was computed with respect to multiple confidence levels:

- $Top-1$ precision: For each scene, the pixel (for the FCN) and the point (for the GCN) with highest affordance score was taken into account to measure the precision.
- $Top-1\%$ precision: In this case the pixels/points were sorted according to their affordance scores and those within the 99th percentile were selected to measure the precision.

In both cases, a grasping proposal is considered as a true positive if it has been manually annotated as good grasping area, and as a false positive if has been manually annotated as bad grasping area.

4. Implementation

In this section all the details related to the data preprocessing, network configurations and used training hyper-parameters are explained.

4.1. Data Preprocessing for the Suction

In order to fit the generated 3D dataset with the GCN model for the suction, some preprocessing steps have been performed. First, the number of points that GCNs could analyze was limited due to computational complexity issues and hardware restrictions. In spite of the fact that the general approach in the literature is to split the input point cloud in blocks later to sequentially feed the model (using a reduced number of points per block), we decided our model to be a single-shot detector. Therefore we increased the number of points to be processed in each batch and set it to 8192. As proposed in [46], we used a random sampling method to reduce the sampling time compared to other complex sampling methods. For suction, each input point was defined by a 9-dimensional feature vector containing, (see Equation (2))

1. The point coordinates with respect to the bin coordinate system.
2. RGB values normalized to [0–1].
3. Normal vector computed using the points within a radius of $0.05m$ before the random sampling.

$$[x_{bin}, y_{bin}, z_{bin}, r, g, b, n_x, n_y, n_z] \tag{2}$$

Here our intuition was that the usage of normal vector information (not available in 2D) would lead to a performance boost, as most suction grasping points are located in flat and regular surfaces. Furthermore, as the annotated data was limited, we applied some data augmentation to generate more training data. For suction, after randomly sampling 8192 points from the scene, the sampled point cloud was randomly rotated in the z axis. As good grasping points do not change when the input cloud is rotated with respect to the vertical axis, the augmentation was straightforward.

4.2. Data Preprocessing for the Gripper

Similar to the followed preprocessing steps for the suction, again 8192 points were randomly sampled for each scene. As explained in Section 3.2, for each annotated heightmap, as many rotated point clouds as discrete angles n were obtained. Thus, n inferences were performed to detect grasping points in a scene for the gripper. Each input point was defined by a 6-dimensional feature vector (Equation (3)):

1. The point coordinates with respect to the bin coordinate system.
2. RGB values normalized to [0–1].

$$[x_{bin}, y_{bin}, z_{bin}, r, g, b] \tag{3}$$

As gripper grasping points usually are in irregular surfaces, our intuition was that normal vector information would not contribute to better learn grasping point representations. Thus, we opted to exclude it. Aside from that, in this case it was not possible to augment the data rotating it vertically, as the rotation of the data had implicit information of the grasping orientation. Therefore, we considered the random sampling as data augmentation, as each sampled point set was totally different to the previously generated ones.

4.3. Training

The details of the tuning hyper-parameters used in our tests are shown in Figure 10. The parameters that are not mentioned, were left as default. As aforementioned in Section 3.2, both for suction and gripper, the dataset was annotated using three classes: Good, bad and neutral. However, as the main goal was to distinguish between good and bad points, and to equalize the instances belonging to each class, the background class was trained with zero loss. Therefore, the elements belonging to that class were not taken into account when the loss was computed. The random sampling and the data augmentation were applied on-line at each training step. We trained our models in the cloud using an AWS EC2 instance with a *16 GB Nvidia V100* GPU.

FCN Hyper-Parameters	
Learning rate type	Constant
Base learning rate	0.001
Momentum	0.99
Batch size	1
RGB backbone	ResNet-101
Depth backbone	ResNet-101
Training steps	5000
Optimizer	SGD
GCN Hyper-Parameters	
Learning rate type	With decay
Base learning rate	0.001
Momentum	0.9
Batch size	2
Backbone GCN	EdgeConv
Backbone layers	14
Decay step	20000
Decay rate	0.5
Num neighbours	16
Num filters	64
Points	8192
Optimizer	Adam

(**a**) Hyper-parameters for suction models.

FCN Hyper-Parameters	
Learning rate type	Constant
Base learning rate	0.001
Momentum	0.99
Batch size	2
RGB backbone	ResNet-101
Depth backbone	ResNet-101
Training steps	20000
Optimizer	SGD
GCN Hyper-Parameters	
Learning rate type	With decay
Base learning rate	0.001
Momentum	0.9
Batch size	6
Backbone GCN	EdgeConv
Backbone layers	7
Decay step	50000
Decay rate	0.5
Num neighbours	16
Num filters	64
Points	8192
Optimizer	Adam

(**b**) Hyper-parameters for gripper models.

Figure 10. Hyper-parameters for suction and gripper Fully Convolutional Network (FCN) and Graph Convolutional Network (GCN) models.

5. Results

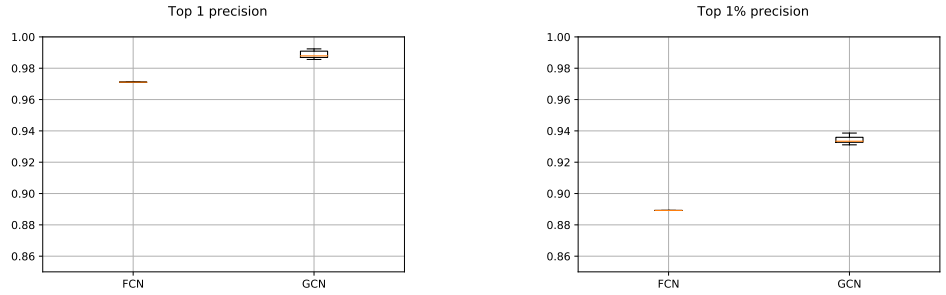
In this section are detailed the results obtained in the tests defined in Section 3.4. In each and every performed test, in addition to the obtained precision scores, a graphical example was given with the following colour/class correspondence: green for good grasping points, blue for bad grasping points and pink for neutral points.

5.1. Test 1

In this test the algorithms were trained with the 80% of the dataset and assessed against the remaining 20%. The metrics defined in Section 3.4 were used to compare both methods.

5.1.1. Suction

The results obtained with both suction FCN and GCN models are shown in Figure 11. In the case of the FCN, as the 2D input data were constant for testing, it was executed once. Nevertheless, as the n-dimensional input points were randomly sampled and with the aim of measuring the average performance, the GCN was executed five times. Therefore, the precision scores obtained with the GCN slightly varied.



(**a**) Top-1 scores for the suction in test 1.

(**b**) Top-1% scores for the suction in test 1.

Figure 11. Obtained precision scores per confidence percentiles for the suction models in Test 1.

The results showed that the deep GCN was able to outperform the precision scores obtained by the FCN. On the one hand, the usage of n-dimensional spatial data helped to better learn grasping point representations for the suction. As most of the suctionable areas were located in flat surfaces, the introduction of the normal vectors in the learning process led to a performance boost. On the other hand, the data augmentation played a key role, as the annotated data was somehow limited. First, the random sampling avoided over-fitting the input data, since the obtained point set after each sampling iteration was totally different to the previously used ones to train the model. In addition, the random rotation in z axis of the sampled point set, contributed even more to generate new grasping points that helped to learn more generic grasp representations.

Nevertheless, the data augmentation was not trivial for the 2D FCN. Although visual transformations could be applied to the RGB image (light, texture, colour, ...), it was not trivial to augment the depth data, as it was stored as a 2D image and had pixel-wise correspondence with the RGB image. In addition, it was not clear up to what extent 2D models as FCNs take advantage of the 3D spatial information, since traditional 2D convolution operations were designed to only extract 2D features. As our training dataset was not so big, the FCN over-fitted it in few training iterations and, thus, the model was not able to learn such good grasp representations as GCNs.

The designed bin-picking oriented pipeline also contributed to the improvement of the results. As all scenes were transformed into the bin coordinate frame and points outside the bin were discarded, the learning problem was simplified and the GCN showed an improved performance, independently of the location of the bin.

An example of an inference of the GCN with a scene obtained from the test split of the dataset is depicted in Figure 12. For illustration purposes, top-1% and top-5% precisions have been selected.

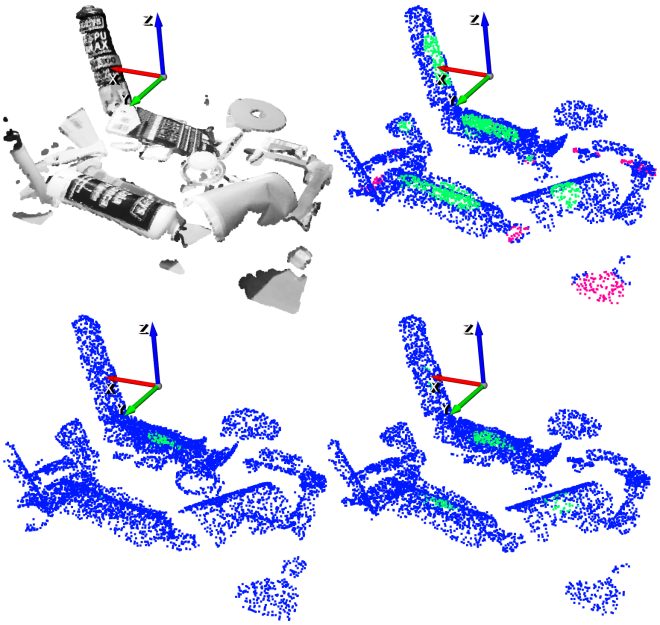
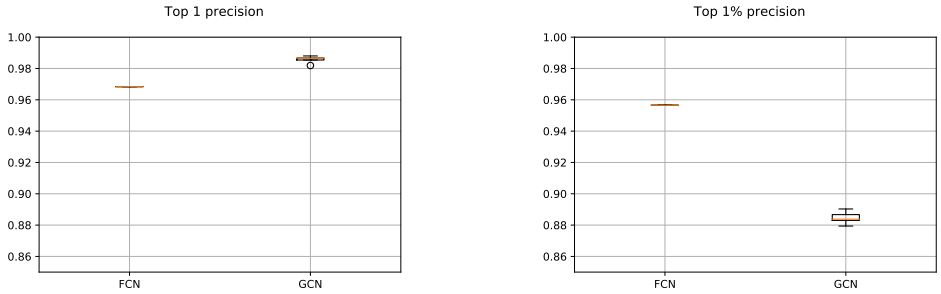


Figure 12. Suction result example. Top-left: Point cloud of the scene. Top-right: Ground-truth annotation. Bottom-left: top-1% predictions. Bottom-right: top-5% predictions.

5.1.2. Gripper

The results obtained for the gripper with both, FCN and GCN models, are depicted in Figure 13. We followed the same methodology as in the case of the suction, with one and five executions for the FCN and GCN respectively. Since in the GCN based approach points were sampled from the original scene, contrary to the FCN based method, the obtained precisions vary. As it can be seen in Figure 13a, the GCN based method outperformed the results obtained with the FCN, giving at least a valid grasping point per each scene with high precision. Nevertheless, Figure 13b shows that the FCN performed more precisely taking into account the highest 1% of the predicted affordances. This means that the GCN showed a more overconfident performance than the FCN, sometimes assigning high affordance scores to points that did not deserve it.



(**a**) Top-1 scores for the gripper in test 1.

(**b**) Top-1% scores for the gripper in test 1.

Figure 13. Obtained precision scores per confidence percentiles for the gripper models in Test 1.

On the one hand, the random sampling of the input data led the GCN model to learn generic features and prevented it from over-fitting the input data. On the other hand, the fact that the GCN learned from n-dimensional features allowed the model to be more confident predicting top-1 affordances. In spite of the fact that the GCN was more confident taking into account the maximum affordance value per each scene, this confidence dropped when the affordances in the 99th percentile were taken into account and the FCN showed a stronger performance.

The results obtained with two different vertical angles for a single scene are shown in Figures 14 and 15. For illustration purposes, the $n = 4$ and $n = 5$ discrete vertical angles have been selected, among the $n = 16$ possible rotations.

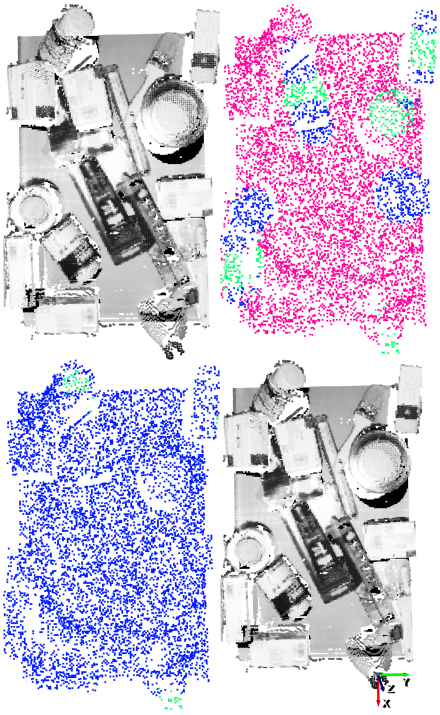


Figure 14. Gripper result example for $n = 4$. Top-left: point cloud of the scene. Top-right: ground-truth annotation. Bottom-left: Top-1% predictions. Bottom-right: Top-1 grasping point. The vertical orientation for the gripper is determined by the y (green) axis.

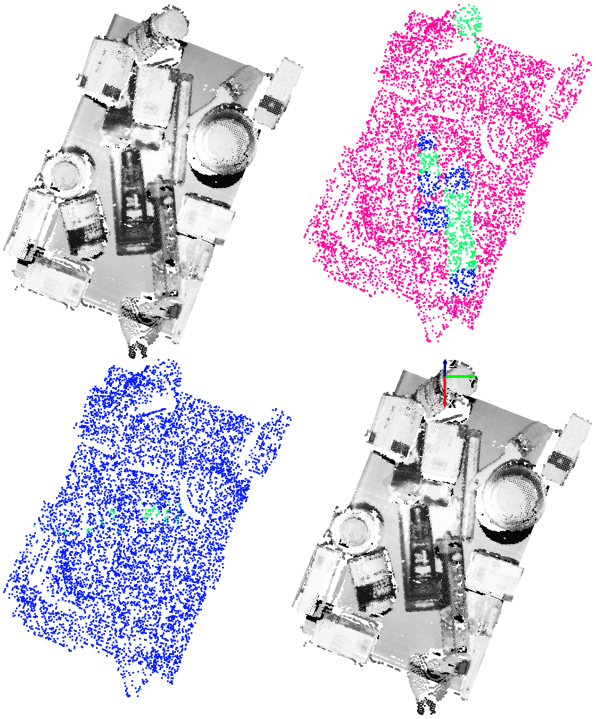


Figure 15. Gripper result example for $n = 5$. Top-left: point cloud of the scene. Top-right: ground-truth annotation. Bottom-left: Top-1% predictions. Bottom-right: Top-1 grasping point. The vertical orientation for the gripper is determined by the y (green) axis.

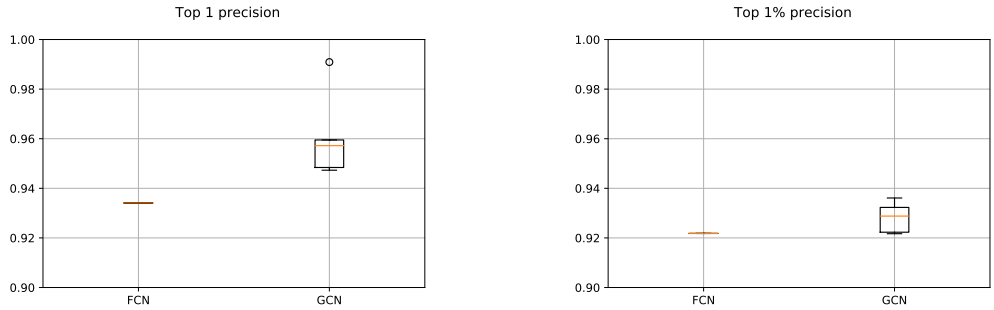
In these examples it can be appreciated that, taking into account the top-1% affordance scores, the GCN model was able to also correctly predict grasping points also in objects that were not annotated as good or bad. In addition, sometimes the model was overconfident and predicted relatively high affordance scores to points that were not good grasping points. This could happen due to the fact that only hard negative samples are annotated as bad. Nonetheless, in both examples the predicted top-1 grasping points are correct and the top-1 precision of the GCN supports its capability to learn grasping points for the gripper.

5.2. Test 2

The main goal of this test was to assess the generalization capability of the trained models for suction and gripper. For that purpose, the previously trained models were assessed against a set of 100 new scenes, composed of randomly arranged completely new parts.

5.2.1. Suction

The achieved top-1 and top-1% precision scores for the FCN and the GCN are depicted in Figure 16. Similar to the previous tests, also in this test the same number of executions were performed per each model. In spite of the fact that both models behaved precisely when they were assessed with scenes full of never seen objects, the GCN showed a stronger generalization capability, obtaining higher top-1 and top-1% precision scores.



(**a**) Top-1 scores for the suction in Test 2.

(**b**) Top-1% scores for the suction in Test 2.

Figure 16. Obtained precision scores per confidence percentiles for the suction models in Test 2.

Regarding the generalization capability of the FCN, although no data augmentation methods were applied, in few iterations it learned generic enough features to correctly predict affordances in new objects with random arrangements. Nevertheless, the data augmentation and the random input data sampling had a lot to do with the results obtained with the GCN. Due to the fact that all point clouds were transformed into the bin coordinate frame, the application of these preprocessing steps was straightforward. Consequently, this preprocessing prevented the model from over-fitting the input data, being able to learn more meaningful spatial features than with the 2D FCN.

A graphical example of the predicted affordances with top-1% and top-5% confidences are shown in Figure 17. Watching this graphical results, it can be said that the learned GCN model shows a strong capability to predict affordances for randomly placed completely new parts.

5.2.2. Gripper

As it can be seen in Figure 18, the precision of both, FCN and GCN decreased considerably when the models were assessed in scenes with totally new objects. Particularly in the case of the GCN, the performance worsened even more than in the case of the FCN, suggesting that the learned model focused more on learning specific features of the training data.

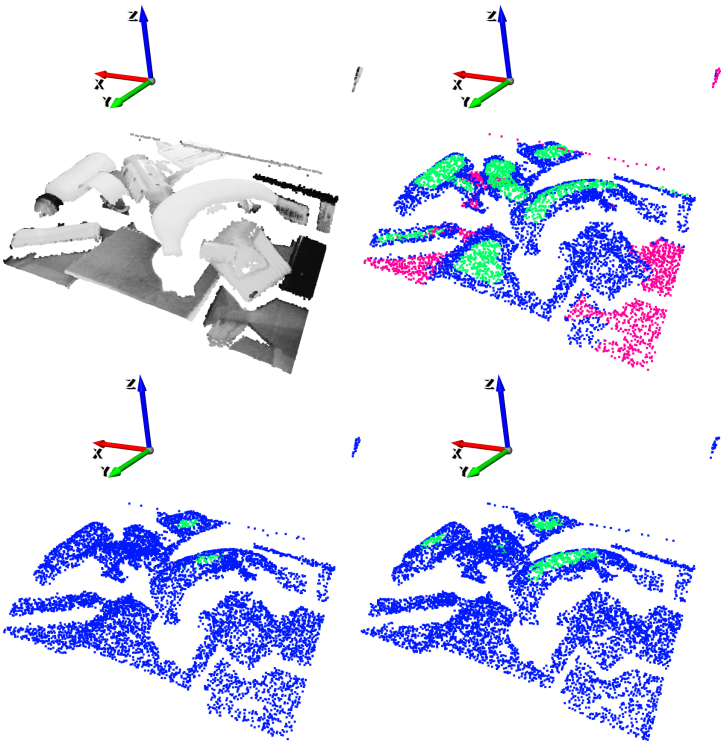
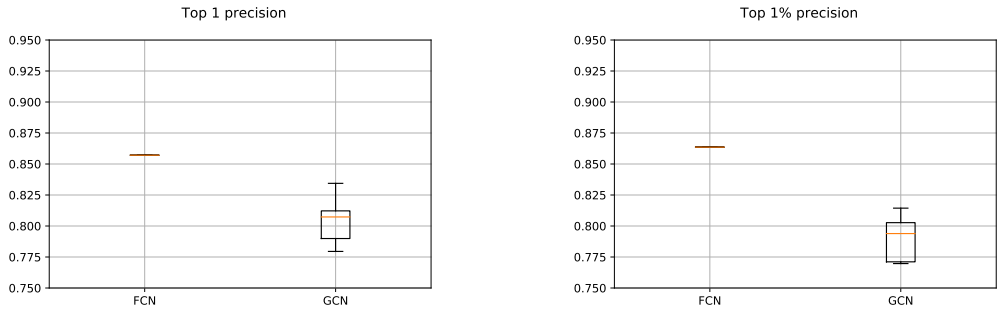


Figure 17. Suction result example with totally new objects. Top-left: point cloud of the scene. Top-right: ground-truth annotation. Bottom-left: top-1% predictions. Bottom-right: top-5% predictions.



(**a**) Top-1 scores for the gripper in test 2.

(**b**) Top-1% scores for the gripper in test 2.

Figure 18. Obtained precision scores per confidence percentiles for the gripper models in Test 2.

For the gripper, the random sampling was considered as a data augmentation method, due to the fact that the input data varied at each training step. Nonetheless, it was not enough to learn generic grasp representations for the gripper. Contrary to the case of the suction where the GCN model was able to correctly extrapolate how to predict affordances in new parts, in the case of the gripper it seemed that the model focused too much in the particularities of the training objects. In Figures 19 and 20, the affordances obtained for two different vertical angles for the gripper are shown, for $n = 2$ and $n = 3$ respectively.

In these graphical results we can see that, in some cases the model assigned high confidence values to points that were not annotated as good and, indeed, were not good grasping points. As an example, in spite of the fact that the top-1 grasping point predicted in Figure 19 had the correct orientation, it was not an adequate grasping point for the gripper due to the width of the part. In some other cases as in Figure 20, however, the model found a correct top-1 grasping point although it assigned high affordances also to points that did not deserve it.

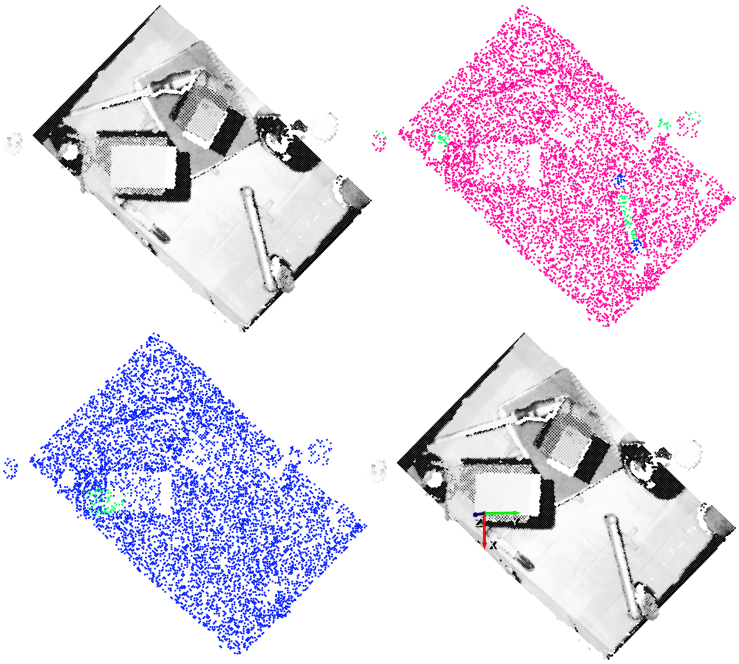


Figure 19. Gripper result example for $n = 2$. Top-left: Point cloud of the scene. Top-right: Ground-truth annotation. Bottom-left: Top-1% predictions. Bottom-right: Top-1 grasping point. The vertical orientation for the gripper is determined by the y (green) axis.

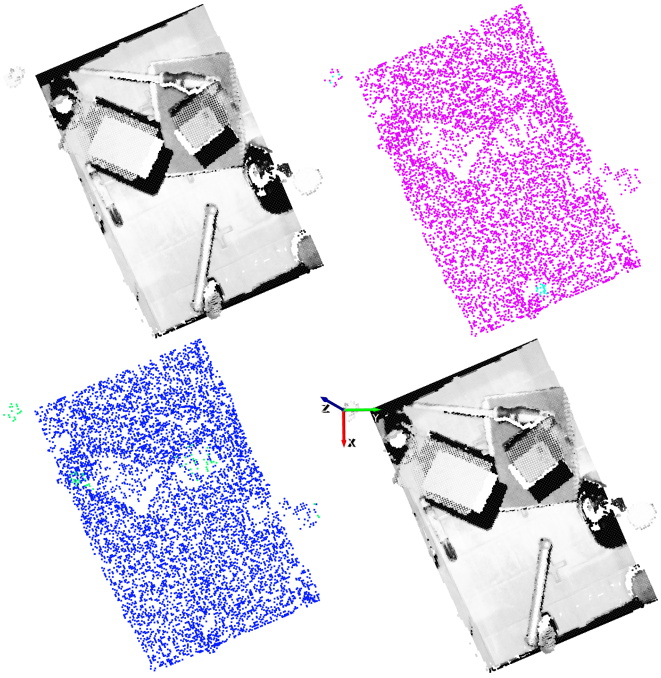


Figure 20. Gripper result example for $n = 3$. Top-left: point cloud of the scene. Top-right: ground-truth annotation. Bottom-left: top-1% predictions. Bottom-right: top-1 grasping point. The vertical orientation for the gripper is determined by the y (green) axis.

6. Discussion

In the work presented here we used a GCN to test if the assumption that n-dimensional information is crucial for predicting object affordances fulfills. We successfully adapted the *Deep GCN* model that was designed for scene segmentation problems to predict object affordance scores for suction and gripper end effectors. To train the GCNs, we created a dataset composed of industrial bin-picking scenarios with randomly arranged multi-reference parts. To that end, we used a Photoneo Phoxi M camera and obtained highly

accurate 2D/3D acquisitions of multiple scenes. In our application we selected a varied set of rigid and semi-rigid objects with different material, shape, colour and texture that were used in the context of the Pick-Place EU project.

Rather than 2D images containing single views of the scene, we used n-dimensional point clouds offering a richer information of the scene (e.g., multiple viewpoints and multiple features per point). Although the usage of unordered n-dimensional point clouds introduced complexity to the learning process, we showed that it is possible to learn to predict object affordances with GCNs. The innovations introduced in [46] let us create deep GCN models for suction and gripper end effectors. Besides, the designed data processing pipeline contributed to create a system which was agnostic to the bin localization in the scene and to the number of cameras being used. Thus, the designed methodology was easily transferable to new scenarios and setups.

Traditionally, DL based methods need arduous manual annotation processes which sometimes make those applications intractable. This drawback of DL applications highlights the importance of automatic data augmentation methods, to automatically increase the training samples in the dataset. Synthetically augmented datasets are widely used and help to generate much data with little effort. However, simulation based methods increase the simulation to reality gap, due to the difficulty to replicate real world conditions in simulators. We directly augmented real n-dimensional spatial data. As we are using 3D data the process was rather trivial contrary to augmenting RGB images where data is arranged in a grid.

The quantitative performance measures obtained confirm our assumption and we can say that indeed 3D spatial information contributes to predict object affordances more precisely. Test 1 showed that the GCNs correctly predict object affordances with known objects but in completely new scenarios and arrangements, obtaining better precisions than the 2D FCN. On the other hand, Test 2 allowed us to check the generalization capability of the trained models. These models had to predict affordances in new objects with random arrangements. The obtained results demonstrated that the GCN based suction model had strong generalization capabilities to correctly predict affordances in similar but completely new parts. However, the precision scores obtained with the gripper indicate that the FCN generalized better to new parts than the GCN, suggesting that the input data was not significant enough for the GCN in the more complex gripper scenario.

Despite the promising results obtained predicting object affordances by training GCNs with n-dimensional spatial features, the system suffered from various limitations. The former was directly related to the resolution of the data. As each point in the scene is represented as a n-dimensional vector, the computational cost increases proportionally when more features are included. Due to hardware and cycle time restrictions, the amount of data to be processed at each step is limited. In this work, each scene was represented by 8192 points, which seemed to be enough in our case with relatively big objects, but not with very small parts. Even though the general approach with GCNs is to split the point cloud in smaller cubes to gain resolution, the general perspective of the scene is lost, which is crucial in the grasping point detection problem.

The second limitation came when the system had to deal with transparent or shiny parts. As most of the depth sensors fail to reconstruct the 3D information of these parts, typically only visual information can be used to infer their affordances.

As far as the grasping strategies are concerned, the developed method for suction takes full advantage of the 3D space and grasps are predicted and executed with 6-DoF. However, that is not the case for the gripper, where the affordances are only predicted taking into account vertical grasps and discrete angles. Consequently, the computational cost is proportional to the selected number of discrete angles, and that makes the solution hardly scalable.

The developed work allowed us to gain experience and knowledge that should be enriched by testing the models with the real robotic system in a real application to see whether the learned grasping representations are valid to pick real objects. Moreover, the manual

annotation is a very time-consuming process. Although data augmentation techniques somehow alleviate it, the DL models still need too much annotated data to converge. Thus, we must look for ways to learn to predict object affordances, with less manually annotated samples. In spite of the fact that current state-of-the-art GCN based approaches mostly focus only on the gripper end effector, it would be enriching to benchmark our work with other methods in a bin-picking scenario. Lastly, the grasping strategy for the gripper must be extended to 6-DoF in a more flexible way to overcome the current generalization and scalability limitations.

Author Contributions: conceptualization, A.I. and A.A; methodology, A.I., A.A., and E.L.; software, A.I.; formal analysis, A.I., A.A.; data curation, A.I.; writing—original draft preparation, A.I.; writing—review and editing, E.L. and A.A.; and supervision, E.L., A.A. All authors have read and agreed to the published version of the manuscript.

Funding: This Project received funding from the European Union's Horizon 2020 research and Innovation Programme under grant agreement No. 780488.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on demand from the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

GPU	Graphical Processing Unit
TPU	Tensor Processing Unit
SGD	Stochastic Gradient Descent
MLP	Multi Layer Perceptron
DNN	Deep Neural Network
DL	Deep Learning
RL	Reinforcement Learning
CNN	Convolutional Neural Network
FCN	Fully Convolutional Network
GCN	Graph Convolutional Network
RGB	Red Green Blue
RGB-D	Red Green Blue Depth
ARC	Amazon Robotics Challenge
DoF	Degrees of Freedom

References

1. Susperregi, L.; Fernandez, A.; Molina, J.; Iriondo, A.; Sierra, B.; Lazkano, E.; Martínez-Otzeta, J.M.; Altuna, M.; Zubia, L.; Bautista, U. RSAII: Flexible Robotized Unitary Picking in Collaborative Environments for Order Preparation in Distribution Centers. In *Bringing Innovative Robotic Technologies from Research Labs to Industrial End-Users*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 129–151.
2. Kober, J.; Peters, J. Imitation and reinforcement learning. *IEEE Robot. Autom. Mag.* **2010**, *17*, 55–62. [CrossRef]
3. Najafabadi, M.M.; Villanustre, F.; Khoshgoftaar, T.M.; Seliya, N.; Wald, R.; Muharemagic, E. Deep learning applications and challenges in big data analytics. *J. Big Data* **2015**, *2*, 1. [CrossRef]
4. Yu, J.; Weng, K.; Liang, G.; Xie, G. A vision-based robotic grasping system using deep learning for 3D object recognition and pose estimation. In Proceedings of the International Conference on Robotics and Biomimetics (ROBIO), Shenzhen, China, 12–14 December 2013; pp. 1175–1180.
5. Zeng, A.; Song, S.; Yu, K.T.; Donlon, E.; Hogan, F.R.; Bauza, M.; Ma, D.; Taylor, O.; Liu, M.; Romo, E.; et al. Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. In Proceedings of the International Conference on Robotics and Automation (ICRA), Brisbane, Australia, 21–25 May 2018; pp. 1–8.
6. Qi, C.R.; Su, H.; Mo, K.; Guibas, L.J. Pointnet: Deep learning on point sets for 3d classification and segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 652–660.

7. Qi, C.R.; Yi, L.; Su, H.; Guibas, L.J. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In Proceedings of the Advances in Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; pp. 5099–5108.
8. Sahbani, A.; El-Khoury, S.; Bidaud, P. An overview of 3D object grasp synthesis algorithms. *Robot. Auton. Syst.* **2012**, *60*, 326–336. [CrossRef]
9. Prattichizzo, D.; Trinkle, J.C. Grasping. In *Springer Handbook of Robotics*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 955–988.
10. Nguyen, V.D. Constructing force-closure grasps. *Int. J. Robot. Res.* **1988**, *7*, 3–16. [CrossRef]
11. Bicchi, A.; Kumar, V. Robotic grasping and contact: A review. In Proceedings of the ICRA. Millennium Conference. International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065), San Francisco, CA, USA, 24–28 April 2000; Volume 1, pp. 348–353.
12. Ferrari, C.; Canny, J.F. Planning optimal grasps. *ICRA* **1992**, *3*, 2290–2295.
13. Bohg, J.; Morales, A.; Asfour, T.; Kragic, D. Data-driven grasp synthesis—A survey. *IEEE Trans. Robot.* **2013**, *30*, 289–309. [CrossRef]
14. Felip, J.; Morales, A. Robust sensor-based grasp primitive for a three-finger robot hand. In Proceedings of the International Conference on Intelligent Robots and Systems, St. Louis, MO, USA, 10–15 October 2009; pp. 1811–1816.
15. Pastor, P.; Righetti, L.; Kalakrishnan, M.; Schaal, S. Online movement adaptation based on previous sensor experiences. In Proceedings of the International Conference on Intelligent Robots and Systems, San Francisco, CA, USA, 25–30 September 2011; pp. 365–371.
16. Brost, R.C. Automatic grasp planning in the presence of uncertainty. *Int. J. Robot. Res.* **1988**, *7*, 3–17. [CrossRef]
17. Miller, A.T.; Allen, P.K. Graspit! a versatile simulator for robotic grasping. *IEEE Robot. Autom. Mag.* **2004**, *11*, 110–122. [CrossRef]
18. Papazov, C.; Haddadin, S.; Parusel, S.; Krieger, K.; Burschka, D. Rigid 3D geometry matching for grasping of known objects in cluttered scenes. *Int. J. Robot. Res.* **2012**, *31*, 538–553. [CrossRef]
19. Aldoma, A.; Vincze, M.; Blodow, N.; Gossow, D.; Gedikli, S.; Rusu, R.B.; Bradski, G. CAD-model recognition and 6DOF pose estimation using 3D cues. In Proceedings of the International conference on computer vision workshops (ICCV workshops), Barcelona, Spain, 6–13 November 2011; pp. 585–592.
20. Hinterstoisser, S.; Lepetit, V.; Ilic, S.; Holzer, S.; Bradski, G.; Konolige, K.; Navab, N. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In *Asian Conference on Computer Vision*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 548–562.
21. Goldfeder, C.; Allen, P.K. Data-driven grasping. *Auton. Robot.* **2011**, *31*, 1–20. [CrossRef]
22. Caldera, S.; Rassau, A.; Chai, D. Review of deep learning methods in robotic grasp detection. *Multimodal Technol. Interact.* **2018**, *2*, 57. [CrossRef]
23. Zeng, A.; Yu, K.T.; Song, S.; Suo, D.; Walker, E.; Rodriguez, A.; Xiao, J. Multi-view self-supervised deep learning for 6d pose estimation in the amazon picking challenge. In Proceedings of the International conference on robotics and automation (ICRA), Singapore, 29 May–3 June 2017; pp. 1386–1383.
24. Schwarz, M.; Milan, A.; Periyasamy, A.S.; Behnke, S. RGB-D object detection and semantic segmentation for autonomous manipulation in clutter. *Int. J. Robot. Res.* **2018**, *37*, 437–451. [CrossRef]
25. Morrison, D.; Tow, A.W.; Mctaggart, M.; Smith, R.; Kelly-Boxall, N.; Wade-Mccue, S.; Erskine, J.; Grinover, R.; Gurman, A.; Hunn, T.; et al. Cartman: The low-cost cartesian manipulator that won the amazon robotics challenge. In Proceedings of the International Conference on Robotics and Automation (ICRA), Brisbane, Australia, 21–25 May 2018; pp. 7757–7764.
26. Lenz, I.; Lee, H.; Saxena, A. Deep learning for detecting robotic grasps. *Int. J. Robot. Res.* **2015**, *34*, 705–724. [CrossRef]
27. Redmon, J.; Angelova, A. Real-time grasp detection using convolutional neural networks. In Proceedings of the International Conference on Robotics and Automation (ICRA), Seattle, WA, USA, 26–30 May 2015; pp. 1316–1322.
28. Kumra, S.; Kanan, C. Robotic grasp detection using deep convolutional neural networks. In Proceedings of the International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 24–28 September 2017; pp. 769–776.
29. Jiang, Y.; Moseson, S.; Saxena, A. Efficient grasping from rgbd images: Learning using a new rectangle representation. In Proceedings of the International Conference on Robotics and Automation, Shanghai, China, 9–13 May 2011; pp. 3304–3311.
30. Zhou, X.; Lan, X.; Zhang, H.; Tian, Z.; Zhang, Y.; Zheng, N. Fully convolutional grasp detection network with oriented anchor box. In Proceedings of the International Conference on Intelligent Robots and Systems (IROS), Madrid, Spain, 1–5 October 2018; pp. 7223–7230.
31. Detry, R.; Kraft, D.; Kroemer, O.; Bodenhagen, L.; Peters, J.; Krüger, N.; Piater, J. Learning grasp affordance densities. *Paladyn J. Behav. Robot.* **2011**, *2*, 1–17. [CrossRef]
32. Long, J.; Shelhamer, E.; Darrell, T. Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 3431–3440.
33. Nguyen, A.; Kanoulas, D.; Caldwell, D.G.; Tsagarakis, N.G. Detecting object affordances with convolutional neural networks. In Proceedings of the International Conference on Intelligent Robots and Systems (IROS), Daejeon, Korea, 9–14 October 2016; pp. 2765–2770.
34. Zeng, A.; Song, S.; Welker, S.; Lee, J.; Rodriguez, A.; Funkhouser, T. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. In Proceedings of the International Conference on Intelligent Robots and Systems, Madrid, Spain, 1–5 October 2018; pp. 4238–4245.

35. Zeng, A.; Song, S.; Lee, J.; Rodriguez, A.; Funkhouser, T. Tossingbot: Learning to throw arbitrary objects with residual physics. *IEEE Trans. Robot.* **2020**, *36*, 1307–1319. [CrossRef]
36. Mahler, J.; Goldberg, K. Learning deep policies for robot bin picking by simulating robust grasping sequences. In Proceedings of the Conference on Robot Learning, Mountain View, California, 13–15 Novermber 2017; pp. 515–524.
37. Levine, S.; Pastor, P.; Krizhevsky, A.; Ibarz, J.; Quillen, D. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *Int. J. Robot. Res.* **2018**, *37*, 421–436. [CrossRef]
38. Kalashnikov, D.; Irpan, A.; Pastor, P.; Ibarz, J.; Herzog, A.; Jang, E.; Quillen, D.; Holly, E.; Kalakrishnan, M.; Vanhoucke, V.; et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv* **2018**, arXiv:1806.10293.
39. James, S.; Davison, A.J.; Johns, E. Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task. *arXiv* **2017**, arXiv:1707.02267.
40. Tremblay, J.; Prakash, A.; Acuna, D.; Brophy, M.; Jampani, V.; Anil, C.; To, T.; Cameracci, E.; Boochoon, S.; Birchfield, S. Training deep networks with synthetic data: Bridging the reality gap by domain randomization. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, Salt Lake City, UT, USA, 18–22 June 2018; pp. 969–977.
41. Zhang, S.; Tong, H.; Xu, J.; Maciejewski, R. Graph convolutional networks: A comprehensive review. *Comput. Soc. Netw.* **2019**, *6*, 11. [CrossRef]
42. Gezawa, A.S.; Zhang, Y.; Wang, Q.; Yunqi, L. A Review on Deep Learning Approaches for 3D Data Representations in Retrieval and Classifications. *IEEE Access* **2020**, *8*, 57566–57593. [CrossRef]
43. Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; Philip, S.Y. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* **2020**, *32*, 1–21. [CrossRef] [PubMed]
44. Zhou, J.; Cui, G.; Zhang, Z.; Yang, C.; Liu, Z.; Wang, L.; Li, C.; Sun, M. Graph neural networks: A review of methods and applications. *arXiv* **2018**, arXiv:1812.08434.
45. Rong, Y.; Huang, W.; Xu, T.; Huang, J. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In Proceedings of the International Conference on Learning Representations, Addis Ababa, Ethiopia, 26 April–1 May 2020.
46. Li, G.; Muller, M.; Thabet, A.; Ghanem, B. Deepgcns: Can GCNs go as deep as CNNs? In Proceedings of the IEEE International Conference on Computer Vision, Seoul, Korea, 27 October–2 November 2019; pp. 9267–9276.
47. Chiang, W.L.; Liu, X.; Si, S.; Li, Y.; Bengio, S.; Hsieh, C.J. Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Anchorage, AK, USA, 3–7 August 2019; pp. 257–266.
48. Liang, H.; Ma, X.; Li, S.; Görner, M.; Tang, S.; Fang, B.; Sun, F.; Zhang, J. Pointnetgpd: Detecting grasp configurations from point sets. In Proceedings of the International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 20–24 May 2019; pp. 3629–3635.
49. Ni, P.; Zhang, W.; Zhu, X.; Cao, Q. PointNet++ Grasping: Learning An End-to-end Spatial Grasp Generation Algorithm from Sparse Point Clouds. In Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2020, Paris, France, 31 May–31 August 2020; pp. 3619–3625.
50. Qin, Y.; Chen, R.; Zhu, H.; Song, M.; Xu, J.; Su, H. S4g: Amodal single-view single-shot se (3) grasp detection in cluttered scenes. In Proceedings of the Conference on Robot Learning, PMLR, Osaka, Japan, 30 October–1 November 2019; pp. 53–65.
51. Mousavian, A.; Eppner, C.; Fox, D. 6-dof graspnet: Variational grasp generation for object manipulation. In Proceedings of the IEEE International Conference on Computer Vision, Seoul, Korea, 27 October–2 November 2019; pp. 2901–2910.
52. Kovacovsky, T.; Zizka, J. PhoXi® 3D Camera. In *Imaging and Machine Vision Europe*; Landesmesse Stuttgart GmbH: Stuttgart, Germany, 2018; pp. 38–40.
53. Song, K.T.; Wu, C.H.; Jiang, S.Y. CAD-based pose estimation design for random bin picking using a RGB-D camera. *J. Intell. Robot. Syst.* **2017**, *87*, 455–470. [CrossRef]
54. PICK-PLACE. Available online: https://pick-place.eu/ (accessed on 5 October 2020).
55. Mahler, J.; Matl, M.; Satish, V.; Danielczuk, M.; DeRose, B.; McKinley, S.; Goldberg, K. Learning ambidextrous robot grasping policies. *Sci. Robot.* **2019**, *4*. [CrossRef]
56. Danielczuk, M.; Mahler, J.; Correa, C.; Goldberg, K. Linear push policies to increase grasp access for robot bin picking. In Proceedings of the 14th International Conference on Automation Science and Engineering (CASE), Munich, Germany, 20–24 August 2018; pp. 1249–1256.
57. Bréhéret, A. Pixel Annotation Tool. 2017. Available online: https://github.com/abreheret/PixelAnnotationTool (accessed on 5 October 2020).
58. Dutta, A.; Gupta, A.; Zissermann, A. VGG Image Annotator (VIA). Version: 2.0.10; 2016. Available online: http://www.robots.ox.ac.uk/~vgg/software/via/ (accessed on 5 October 2020).
59. Mvtec Halcon. Available online: https://www.mvtec.com/products/halcon/ (accessed on 5 October 2020).
60. Zhou, Q.Y.; Park, J.; Koltun, V. Open3D: A Modern Library for 3D Data Processing. *arXiv* **2018**, arXiv:1801.09847.
61. Aggarwal, C.C.; Wang, H. *Managing and Mining Graph Data*; Springer: Berlin/Heidelberg, Germany, 2010; Volume 40.
62. Hamilton, W.; Ying, Z.; Leskovec, J. Inductive representation learning on large graphs. In Proceedings of the Advances in Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; pp. 1024–1034.
63. Wang, Y.; Sun, Y.; Liu, Z.; Sarma, S.E.; Bronstein, M.M.; Solomon, J.M. Dynamic graph cnn for learning on point clouds. *ACM Trans. Graph.* **2019**, *38*, 1–12. [CrossRef]

64. Armeni, I.; Sener, O.; Zamir, A.R.; Jiang, H.; Brilakis, I.; Fischer, M.; Savarese, S. 3d semantic parsing of large-scale indoor spaces. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 1534–1543.
65. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 26 June–1 July 2016; pp. 770–778.
66. Huang, G.; Liu, Z.; Van Der Maaten, L.; Weinberger, K.Q. Densely connected convolutional networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 4700–4708.

9

Dynamic mosaic planning for a robotic bin-packing system based on picked part and target box monitoring

Authors

Iriondo, Ander; Lazkano, Elena; Ansuategi, Ander; Fernandez, Ane; Maurtua, Iñaki;

Publisher

Springer

Journal

International Journal of Advanced Manufacturing Technology

Year

2023

Quartile (Web of Science / Scimago)

Q2/Q1

DOI

https://doi.org/10.1007/s00170-022-10601-9

The International Journal of Advanced Manufacturing Technology
https://doi.org/10.1007/s00170-022-10601-9

APPLICATION

Dynamic mosaic planning for a robotic bin-packing system based on picked part and target box monitoring

Ander Iriondo[1,2] · Elena Lazkano[2] · Ander Ansuategi[1] · Ane Fernandez[1] · Iñaki Maurtua[1]

Received: 15 June 2022 / Accepted: 24 November 2022
© The Author(s) 2023

Abstract
This paper describes the dynamic mosaic planning method developed in the context of the PICKPLACE European project. The dynamic planner has allowed the development of a robotic system capable of packing a wide variety of objects without having to adjust to each reference. The mosaic planning system consists of three modules: First, the picked item monitoring module monitors the grabbed item to find out how the robot has picked it. At the same time, the destination container is monitored online to obtain the actual status of the packaging. To this end, we present a novel heuristic algorithm that, based on the point cloud of the scene, estimates the empty volume inside the container as empty maximal spaces (EMS). Finally, we present the development of the dynamic IK-PAL mosaic planner that allows us to dynamically estimate the optimal packing pose considering both the status of the picked part and the estimated EMSs. The developed method has been successfully integrated in a real robotic picking and packing system and validated with 7 tests of increasing complexity. In these tests, we demonstrate the flexibility of the presented system in handling a wide range of objects in a real dynamic packaging environment. To our knowledge, this is the first time that a complete online picking and packing system is deployed in a real robotic scenario allowing to create mosaics with arbitrary objects and to consider the dynamics of a real robotic packing system.

Keywords Bin-packing · Logistics · Manipulation · Online mosaic planning · Pick and place

1 Introduction

Object placing is one of the most challenging tasks in robotic manipulation and contrary to simple pick-and-drop approaches, object placing methods try to find the best location to leave the picked part. One of the well-known applications of object placing algorithms is bin-packing, that tries to fit the picked parts in tight spaces with the goal of optimising the space inside the bin. In industrial environments such as warehouses or distribution centres, efficient and automated bin-packing methods are highly demanded, due to the cost of storage and transportation of oversized containers. Currently, in most of those centres, the packing operations are carried out only by humans, leading to error-prone and inefficient packing solutions.

This work is focused on the packing system developed in the context of the PICKPLACE[1] European project which seeks to develop a flexible, safe and dependable picking and packing system considering the real use-cases of *ULMA Handling Systems* (UHS) and *Türk Otomobil Fabrikasi A.Ş.*(TOFAŞ) end-users.

In the distribution centres installed by UHS, the order pick-and-packaging (the process of collecting items to create a package for shipment) is the last and most challenging task in the warehouse automation. Depending on the type of installation, the warehouse operator takes

✉ Ander Iriondo
ander.iriondo@tekniker.es

Elena Lazkano
e.lazkano@ehu.eus

Ander Ansuategi
ander.ansuategi@tekniker.es

Ane Fernandez
ane.fernandez@tekniker.es

Iñaki Maurtua
inaki.maurtua@tekniker.es

[1] Department of Autonomous and Intelligent Systems, Tekniker-Basque Research and Technology Alliance (BRTA), Iñaki Goenaga, 5, Eibar, 20600, Gipuzkoa, Spain

[2] Robotics and Autonomous Systems group (RSAIT), Department of Computer Science and Artificial Intelligence, University of the Basque Country (UPV/EHU), Pº Manuel Lardizabal,1, Donostia-San Sebastián, 20018, Gipuzkoa, Spain

[1] PICKPLACE:https://pick-place.eu/

Published online: 12 January 2023

multiple types of goods at the end of the line and places them in target boxes. In the automotive sector, TOFAŞ deals with more than 63.000 different spare parts that are sent to different dealers in manually prepared packages. This monotonous packing process is currently held in 90% by humans due to the difficulty of replacing human dexterity to handle parts with high variability in size, shape, weight and material.

The feasibility of fully automated systems is currently limited by many factors such as the huge variability of items, system robustness and industrial throughput requirements. Human-robot collaboration seeks to combine the robot's efficiency and the human dexterity to optimise the packing process.

Bin-packing solutions are mainly differentiated depending on the level of knowledge available in the application. Traditionally, bin-packing methods deal with ideal conditions in simulation and the identity and the set of items to be packed, as well as their properties are known. These ad hoc solutions assume that the parts can be packed in any orientation and omit the presence of automatic picking/packing systems such as robots. In other approaches that consider automatic picking/packing systems, they assume that the objects are known and that are always grasped in the same way. Thus, in both cases, the packing order of the items and the mosaic can be calculated offline. The main disadvantage of this kind of methods is that they ignore the uncertainties introduced by a real picking/packing system and assume that the process is ideal. In fact, in dynamic scenarios such as big warehouses where thousands of known and even unknown references need to be managed, those solutions fail, and more flexible systems are required.

Taking into account the high variability of the objects considered in the project as well as the need to handle unknown objects, an affordance-based flexible grasping system was developed in the context of the project [1]. Following this approach, the system predicts grasping points without relying on the identity of the parts but only focusing on the shape, colour and texture of them. Having such a flexible grasping system implies that items are not identified before being picked, and thus the grasping point prediction varies in each execution of the algorithm. Indeed, the way that each object is picked has a big effect in the place operation and, therefore, the grasped item has to be monitored online. Additionally, when dealing with a huge amount of references with highly variable shapes and textures, small movements in the outbound box produced by the human or the warehouse system itself may cause the products to drop. Therefore, online monitoring of the outbound box becomes necessary to avoid collisions.

The patented palletising solution of UHS, dubbed IK-PAL, is composed of a palletising robot and the algorithm that calculates the palletisation mosaic. This system has been successfully deployed in warehouses of big companies such as EROSKI.[2] However, this first version of the mosaic planning algorithm was thought to work with regular items and assuming that the parts would remain static after being packed in the mosaic. Nevertheless, the increasing market of the e-commerce nowadays demands flexible solutions to handle a high variety of items with highly variable shapes. Introducing irregular items to the system makes it impossible to assume that the items will remain in the ideal position once they have been packed, and arises the need for monitoring the outbound box.

In this work, several novelties are introduced to evolve IK-PAL's mosaic planning algorithm into a flexible and dynamic planner. As the grasping point detection system developed in the project is agnostic to the objects' identity and the predicted grasping points are not predefined, we first developed a system to estimate online the dimensions and the orientation of the picked parts. Second, we designed a polynomial-order heuristic method to monitor online the empty volume inside an outbound box. This method computes the free volume in the form of empty maximal spaces (EMSs), which serves as input to the dynamic IK-PAL mosaic planner. With this new method, the grabbed items and outbound containers are monitored online, and the packing pose for each picked part is estimated considering the actual status of the packaging. Therefore, introducing these novelties to the system, we are able to dynamically handle dropped objects and to avoid collisions when items move around inside the bin.

The developed dynamic mosaic planner was finally integrated with an automatic picking system in a real robotic scenario. To the best of our knowledge, this is the first time that a complete online picking and packing robotic system is developed that allows creating mosaics with arbitrary objects and considering the dynamics of a real robotic packing system.

2 Literature review

Object placing, along with picking, have been core problems since early days of robotics. Particularly, the problem of fitting a set of objects inside a destination bin optimising the used space has traditionally attracted a lot of interest. Most of the proposed solutions try to pack regular shaped objects such as boxes with non-overlapping constraints.

Traditionally, the majority of the approaches assume that the set of items to be packed is known, and therefore, the packing mosaic is calculated offline. Currently, there are exact solutions that optimally solve the 3D bin-packing problem offline. A popular method was presented by

[2] EROSKI: https://www.eroski.es/

Martello et al. in [2] and [3]. It uses a branch-and-bound algorithm to optimally solve the problem. This algorithm was subsequently optimised by Boef et al. in [4]. The bin-packing problem belongs to the class of NP-hard methods and, therefore, it cannot be solved in polynomial time. Consequently, much research has been done in approximate methods to solve the problem to near optimality [5]. To mention some, Jakobs et al. first introduced the well known *bottom-left* heuristic in [6]. The *best-fit decreasing* algorithm was developed and analysed by Johnson et al. in [7]. However, these methods were designed to work in 1D or 2D.

The use of metaheuristic-based algorithms led to good approximate solutions for the orthogonal 3D bin-packing problem [8]. In [9], Crainic et al. efficiently solved the orthogonal 3D bin-packing problem applying a two-level *tabu-search* method. Additionally, Faroe et al. applied the *guided local search* (GLS) to solve the same problem [10]. Furthermore, genetic algorithms were also successfully applied in [11] to solve orthogonal 2D and 3D bin-packing problems. State-of-the-art learning-based approximate strategies used *deep reinforcement learning* (DRL) to optimise the packing of orthogonal parts [12]. Nevertheless, these methods were only designed for the orthogonal bin-packing and are only practical when dealing with boxes. Moreover, in these works, it was assumed that the parts could be packed without restrictions in the orientation. Nonetheless, when the items are packed by a robot, the picking location of the parts limits the placing and not all orientations are feasible.

More recently, the bin-packing of both regular and irregular multi-reference parts has become a need in intralogistic applications, particularly caused by the growth of the e-commerce. When the shape of the items is irregular, it is not feasible to find an optimal solution to the problem as the search space is infinite, and only can be solved with approximate solutions. For instance, Zhao et al. used a metaheuristic algorithm to estimate a near-optimal packing mosaic for irregular objects [13]. In this work, the items were represented as meshes and the empty space as a bounding box of the bin. However, the mosaic was calculated offline and it was not considered that objects could fall, which is likely to occur in the real system using irregular parts. Besides, the method was time demanding and could not be used in real time. In [14], the authors proposed an heuristic-based method that considered the offline packing of geometrically complex 3D objects with stability constraints. On the one hand, the picked part was represented as a 2D heightmap that was generated using ray-casting. On the other hand, the availability of the outbound box was monitored using a second 2D heightmap that was updated whenever a new object was placed. Although the packing method checked the status of the outbound box heightmap to look for a stable packing pose, falls were not considered since this process was executed offline, before picking any item. Besides, the system was only assessed in simulation.

The stability of the packed mosaic is guaranteed dealing with regular mono-reference items under controlled scenarios. However, this is not the case when it comes to multi-reference applications, particularly when irregular parts are also considered. Therefore, when each object is packed, it cannot be assumed that the constructed tote will remain constant and hence, the mosaic must be calculated online. Indeed, it is required to analyse the status of both the picked part and the packaging at each iteration due to the following reasons:

1. When picking a huge number of multi-reference items ad hoc grasping solutions usually fail and flexible solutions are needed. Flexible grasping systems are not able to guarantee that the items will always be picked in the same way. Therefore, the packing position cannot be calculated until the object is picked. The picking pose limits the possible packing orientations.
2. When dealing with irregular parts, objects are likely to fall off or move around after being packed. Thus, the theoretical and real mosaics differ and the destination boxes need to be monitored online whenever a new product is packed.

Regarding online bin-packing methods, Ha et al. proposed a bin-packing heuristic algorithm to pack orthogonal objects online at arrival time without any prior knowledge about the items to be packed [15]. This work, however, did not consider a picking system that would restrict the packing orientations. In fact, it was only implemented for ideal setups in simulation. For instance, the framework developed by Wang et al. [16] dealt with the packing of a known set of objects arriving in a non-deterministic order. The proposed solution was composed of an offline planner and a verification algorithm that checked the feasibility of all permutations of arrival orders. Although the system handled both regular and irregular objects, in that work object falls were not considered, and the status of the destination box was not monitored. Additionally, authors claim that their work was only practical with a small subset of objects. Hong et al. went one step further and solved the irregular 3D object bin-packing in an online manner [17]. In that approach, a depth camera was used to estimate the area (highest contour) of the picked object. To that end, the depth camera was located below the picked part and it was aligned with the end effector of the robot. Concerning the destination box monitoring, the status of the bin was represented as a 2D depth-map and an analysis of the available contact-free 2D positions was performed, using the contour information of the picked part. Additionally, some other criteria such as

placing the objects in the lowest possible position or bringing things together were applied. However, the authors did not use a real picking and packing system (only simulated some manipulation error) which could have a large effect on the structure of the final mosaic. Recently, learning algorithms such as DRL have been successfully applied to solve the online bin-packing problem with orthogonal parts, for instance, in [18]. This approach, however, only dealt with orthogonal items, it lacked an automatic picking system and it was only tested under ideal simulated conditions. Zhao et al. also propose a DRL-based packing system for orthogonal objects that has been tested in a real robotic system [19]. In that work, once the robot left the picked object in the target box, they did not monitor it online and assumed that the object would be ideally placed.

Summing up, most of the works in the literature used orthogonal items, assumed that the packing process was ideal and, therefore, they did not provide online monitoring of the status of both the picked part and the outbound box. However, this is crucial particularly in real unstructured scenarios where the packing conditions are variable and when irregular multi-reference parts are considered in the system. Additionally, the fact that a huge amount or even unknown items need to be handled in intralogistic centres requires to have a flexible grasping system. Indeed, the grasping pose has a large effect in how the part can be packed, but none of the analysed works uses a real picking system. In this work, we present the online packing solution developed in the context of the PICKPLACE European project that is able to pack both regular and irregular items considering the packing status at each iteration. The contributions of the paper are as follows:

- Firstly, we develop an online monitoring system for the picked part. In PICKPLACE, a flexible grasping system is used to handle a wide variety of parts. Therefore, the grasping points are not predefined, and thus, it is necessary to monitor the object in order to know how the robot has picked it up.
- We also implement an algorithm to dynamically monitor the destination box and heuristically estimate the free volume as EMSs. When packing a wide variety of different shaped objects, it is necessary to dynamically monitor the destination box to take into account the uncertainties (e.g. drops, movements) that may occur once placed in the box.
- Finally, we develop a dynamic version of IK-PAL mosaic planner that considering the picked part and the destination box monitoring heuristically calculates the packing pose for the picked part.
- The grasping point detection algorithm developed in [1], the dynamic picked part and destination box monitoring methods and the dynamic IK-PAL planner are integrated in a real pick-and-place system.

(a) UHS pilot.

(b) TOFAŞ pilot.

Fig. 1 Pilots in the PICKPLACE project

In this manuscript, we show that the developed packing system is practical to generate mosaics in an online manner using a real robot, with a wide variety of items and being able to also consider uncertainties such as falls in the outbound box.

3 System overview

In PICKPLACE, three pick-and-packing scenarios are considered, which are implemented both in the UHS and TOFAŞ pilots (Fig. 1):

1. Order preparation in UHS: Mono-reference to multi-reference.
2. Order return in UHS: Multi-reference to mono-reference.
3. Order preparation in TOFAŞ: Multi-reference to multi-reference.

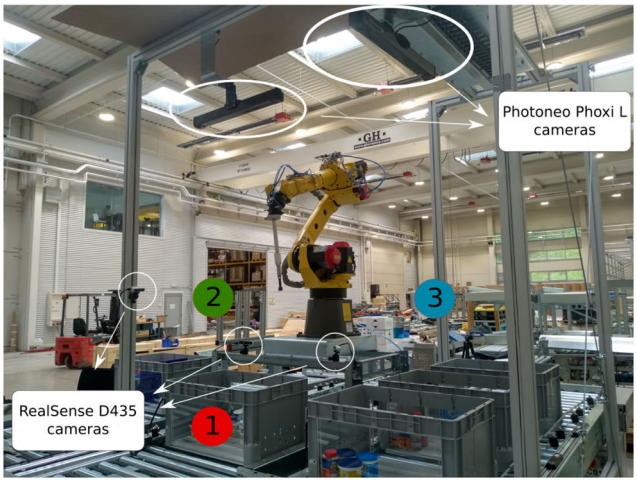
In the first case, after the system receives an order from the warehouse management system (WMS), it determines the arrival order of the mono-reference bins to the picking area. The WMS generates the sequence for a new order preparation or for an order return process, and is used in both prototypes. Once the robot picks the required object, it packs the item in the bin of the corresponding order, creating a multi-reference mosaic. In the second use case, the returned multi-reference bin goes directly to the picking area of the UHS pilot. Items are picked according to the order determined by the flexible picking system, until no more items are left inside the bin. Once each object has been picked, its dimensions are estimated and, at the same time, the product is identified. Finally, each product is returned to the corresponding bin of the product's reference, creating a mono-reference mosaic. The third and last use case is implemented in the TOFAŞ pilot and tackles the multi-reference to multi-reference picking and packing problem. Once each product is picked from the picking box, it is identified with a bar-code reader and it is finally placed in the corresponding multi-reference mosaic.

In practice, almost the same workflow is followed in all the use-cases due the following reasons: (1) The flexible grasping system lets us abstract from the type of inbound box (mono/multi-reference). (2) With both known and unknown items, the dimensions and orientation of the picked items need to be estimated online, since those are dependent on how the parts have been picked. Furthermore, the online mosaic generation system abstracts from the category and is able to handle novel parts. The main difference, indeed, is that in the use case 1, the items do not need to be identified after being grasped, as the system knows the category of the mono-reference bin. However, in the use cases 2 and 3, the parts are grasped from a multi-reference bin using a flexible grasping system and therefore, an identification step is required to decide the target bin for the grasped item.

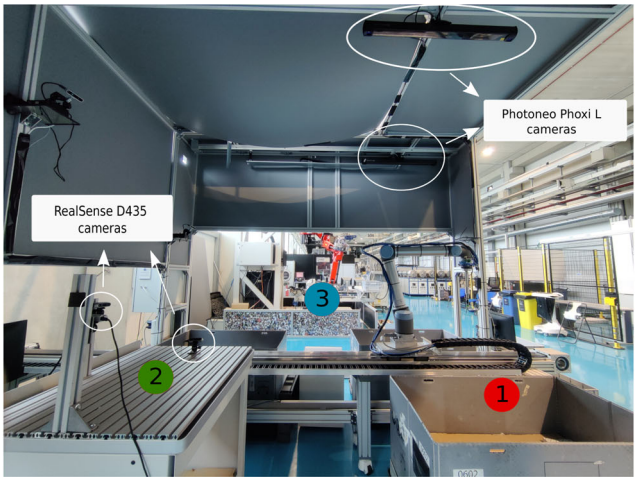
3.1 UHS and TOFAŞ pilots

Both UHS and TOFAŞ pilots are composed of 3 main areas (see Fig. 2): (1) The inbound area, (2) the picked part monitoring area and (3) the outbound area. In both cases, the inbound area contains a single input bin from which the objects are picked. However, the outbound area contains up to three containers in UHS and two containers in TOFAŞ.

The robot used in the UHS prototype is the 6-DoF Fanuc m10ia/10M industrial robot equipped with a suction end effector. In the case of TOFAŞ, the 7-DoF UR10 collaborative robot is used, also with a suction end effector. In the first case, the robot is fixed on a base, from which it is able to reach the 3 areas. In the latter, however, the manipulator is attached to a mobile track which enables the robotic arm to move from one area to another.



(a) Components of the UHS pilot.



(b) Components of the TOFAŞ pilot.

Fig. 2 Main components of UHS and TOFAŞ pilots. (1) Inbound area, (2) part monitoring area, (3) outbound area

In both pilots, over the inbound and outbound areas, there are two *Photoneo Phoxi L* cameras that are used to estimate the grasping points and to monitor the outbound container respectively. In the first case, the camera is fixed as there is only a single bin in the inbound area. In the second case, however, the camera is attached to a linear track that enables the monitoring of up to two/three outbound bins with a single camera, depending on the prototype. As far as the picked part monitoring area is concerned, three extrinsically calibrated *RealSense D435* cameras are used, both in UHS and TOFAŞ, all of them with a fixed location in the layout. This enables the system to have a full view of the item leading to a better estimation of the dimensions and the orientation. The TOFAŞ layout is covered with a housing to

reduce the impact of external light on the cameras. All the software runs in an Intel i7-8700@3.2GHz × 12 CPU, with 32GB of RAM and a Nvidia GeForce RTX 2080 Ti GPU with 11GB of memory.

3.2 Software architecture

The application and all its submodules are developed on the *Robot Operating System* (ROS) framework [20]. Indeed, our implementations are based on the layered control architecture dubbed *Robotframework*, which lets us easily develop robotic applications reusing functionalities that are common in many robotic applications [21]. This framework is divided into the following four layers: user interface, application, abilities and drivers (see Fig. 3).

- **Driver layer:** This layer implements all the communication interfaces for data exchange between robots, tools and sensors in simulated environments and real setups.
- **Abilities layer:** It includes the modules that provide the system with high-level functionalities such as manipulation, grasping or perception. It is the middleware between the application level and the drivers.
- **Application layer:** This level contains the robotic applications that provide a solution to the multi-reference random bin picking problem, defined as one of the main goals in PICKPLACE. In our case, each use case is implemented as an operation mode. The robot manager is the module in charge of controlling the overall system, maintaining its status all the time. In addition, it offers all the functionalities through a REST API.
- **User interface layer:** It includes the modules in charge of the interaction between the human and the robotic system, such as the WMS or the Graphical User Interface (GUI).

The diagnostics module collects information from the drivers and the abilities layers and is used to check the status of all the modules, to collect historical data or to make decisions based on the type or errors. The Logger module is responsible for offering log capabilities to all the layers in the architecture.

Among the different functionalities developed in the context of the project, this paper focuses on the packing system. The core of that system is composed of three main modules that are part of the *perception* and *packing mosaic planning* abilities:

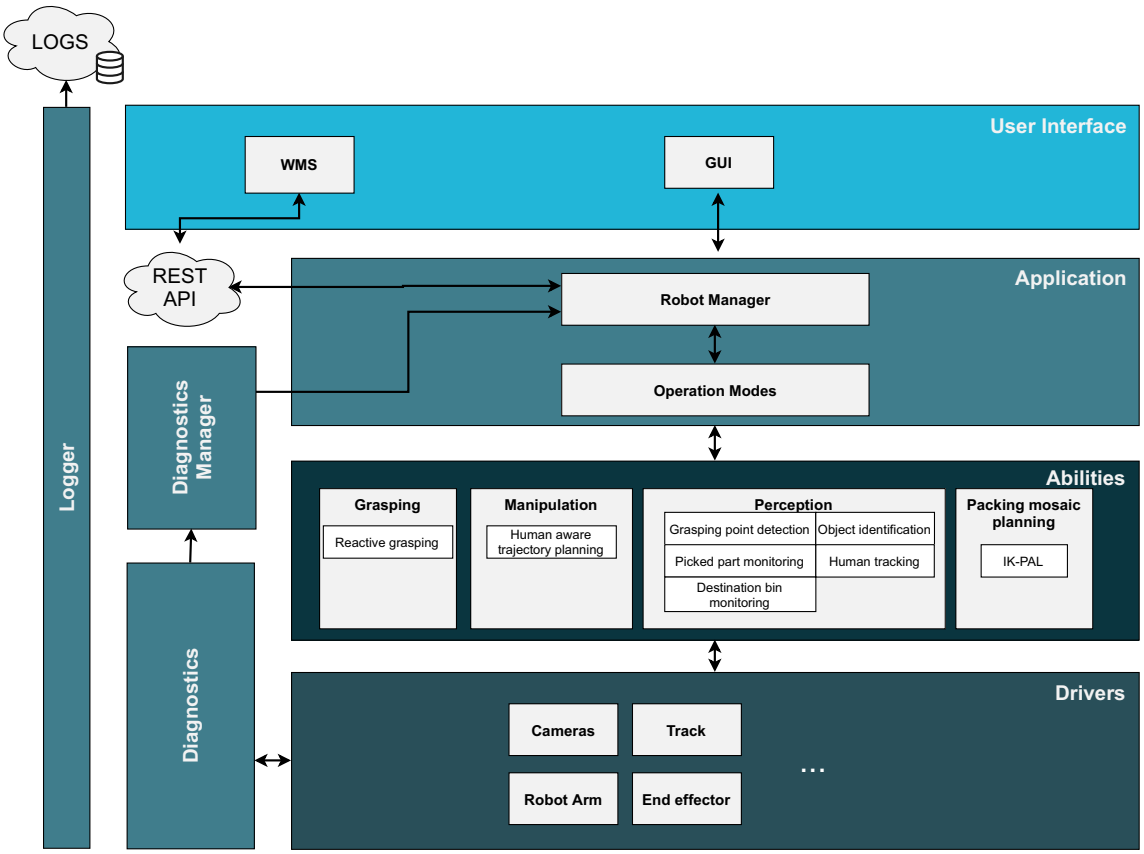


Fig. 3 Robotframework

1. Picked part monitoring.
2. Destination bin monitoring.
3. Dynamic IK-PAL mosaic planning.

These modules are explained in more detail in Sections 3.3, 3.4 and 3.5 respectively.

3.3 Picked part monitoring

The flexible grasping system is capable of handling a large variety of items, and thus the way objects are picked is not predefined. Therefore, the dimensions and the orientation of each picked item need to be estimated online. In this work, the object's bounding box is used as an estimate of the dimensions of the picked part. The bounding box is defined by its origin $bbox_o = [orig_x, orig_y, orig_z, orig_{yaw}]$ (the xyz coordinates and the rotation in z axis of a specific corner of the bounding box with respect to the end effector) and the dimensions $bbox_d = [dim_x, dim_y, dim_z]$.

The bounding box is estimated using three extrinsically calibrated *RealSense D435* cameras. The use of three viewpoints lets us have a full view of the picked item, vital to properly estimate the dimensions of the part. Then, the following steps are performed before estimating the bounding box:

1. Combine the point clouds of the 3 *RealSense* cameras using extrinsic calibrations.
2. Downsample the combined point cloud with a voxel-downsampling algorithm with a voxel-size of $V_b^3 \text{mm}^3$.
3. Define a monitoring volume of dimensions M_d with respect to a predefined pose in the space. The robot's end effector moves to a fixed position inside the volume to show the picked part to the cameras.
4. Remove the end effector from the point cloud. Since its dimensions and its position in the monitoring volume are known, a simple point cloud crop is done.
5. Apply a radius filter to remove the noise from the point cloud, with a filtering radius F_r and considering F_n points.

At this point, it is assumed that all the remaining points inside the monitoring volume belong to the item.

The calculated bounding box is finally optimised in the vertical z axis, as it is the most critical orientation during the packing process. The bounding box is not optimised in the x and y axes, as it is assumed that the error in these axes disappears due to gravity when the item is packed.

To optimise the bounding box in the z axis, the origin of the point cloud is translated to its centroid and the cloud is rotated N discrete times with respect to the z axis, checking at which rotation the bounding box is smaller. The bounding box is always estimated following the axes of the origin of the cloud. Finally, the estimated bounding box with the smallest volume is transformed again from the translated coordinate frame into the end effector's coordinate frame (see Fig. 4).

3.4 Heuristic method for destination bin monitoring

The heuristic algorithm to monitor the empty space in the destination box is based on the EMS concept that has been previously used in multiple works [11, 15, 22, 23]. Generally speaking, this concept was applied to 2D and 3D bin-packing problems, always with orthogonal objects assuming ideal conditions in simulated worlds. Moreover, these works assumed that the generated mosaics were static and, therefore, the EMSs were estimated in the theoretical representation of the mosaic. However, this cannot be assumed when irregular multi-reference elements are considered in the system as the uncertainty this introduces to the system generally causes the theoretical and real mosaics to differ. Thereby, our heuristic method tries to estimate the EMSs after each new item has been packed with the aim of having a faithful representation of the real status of the bin.

The system gets as input the point cloud of the scene (CL) and the localisation of the bin (bin_p), and outputs the empty usable volume represented as a list of 3D empty cubes (S) of maximum possible size. Figure 5 depicts a sample scene and its point cloud. Each solution cube $s \in S$ is defined by its origin coordinates with respect to the origin of the bin $s_o = [orig_x, orig_y, orig_z]$ and its dimensions $s_d = [dim_x, dim_y, dim_z]$. The origin of the bin is located in a lower corner of it. A representation of the generated EMSs is depicted in Fig. 6.

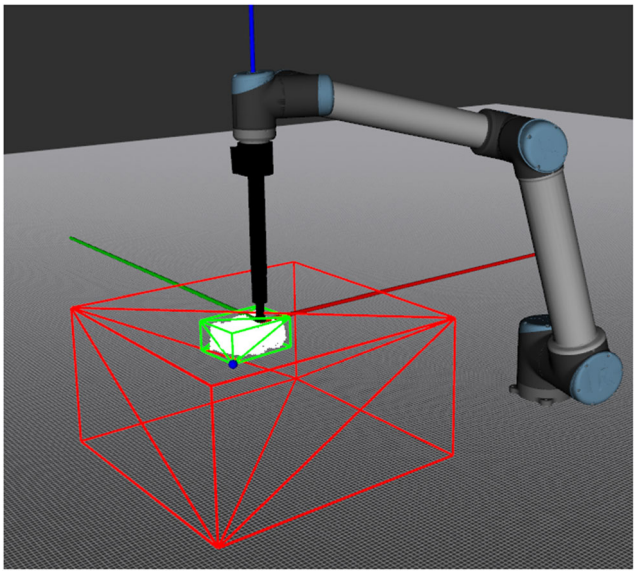
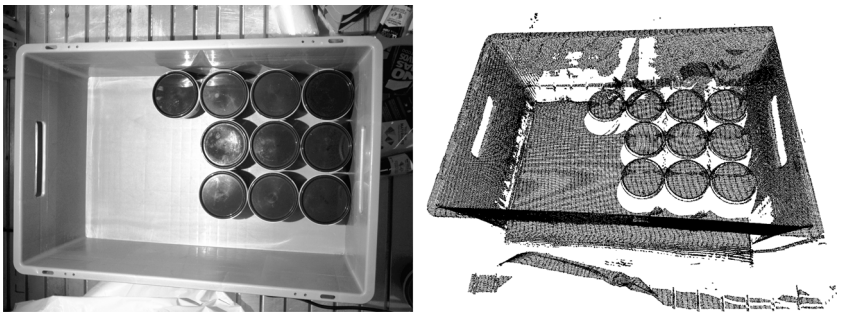


Fig. 4 Bounding box estimation of the picked part. The monitoring volume and the bounding box are represented as red and green cubes respectively. The blue sphere indicates the origin of the bounding box



(a) Original scene.

(b) Point cloud of the scene.

Fig. 5 Original scene image and point cloud

The process runs as follows. First, the bin is localised in the point cloud of the scene with the aim of discarding the points that lay outside of it. To that end, a previously developed 3D model matching algorithm is used [24]. As the dimensions of the bin (bin_d) are known, the bin localisation algorithm creates the 3D model of the upper edges of the bin and matches it with the point cloud of the scene, finally to obtain its pose (bin_p). In both pilots, a *Photoneo Phoxi L* 3D industrial camera is used to monitor the destination bins.

Once the bin is localised, the point cloud of the scene is transformed into the bin's coordinate frame and the points that are outside of it are discarded. This process is followed by a filtering step in which two methods are applied to reduce the noise in the cloud, both of them available in the *PCL* library [25]. First, a clustering-based filtering helps to remove small groups of points, not attached to the main cloud, that are due to the sensor noise and light reflections. Then, a statistical outlier filter removes the noise attached to the main cloud that prevents regular surfaces. The designed heuristic method uses a 3D occupancy grid $\mathbb{O}$ as a representation of the occupied space inside the outbound box. $\mathbb{O}$ is represented as a 3D array, where each position belongs to a voxel in the space. To fill it, we divide the space inside the bin in voxels of size $V_s{}^3\text{mm}^3$ using *octrees*. This data structure lets us efficiently check the status of each voxel and fill $\mathbb{O}$. An example of the generated

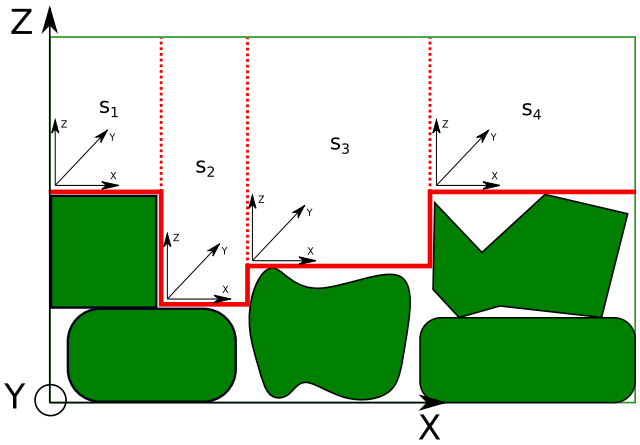
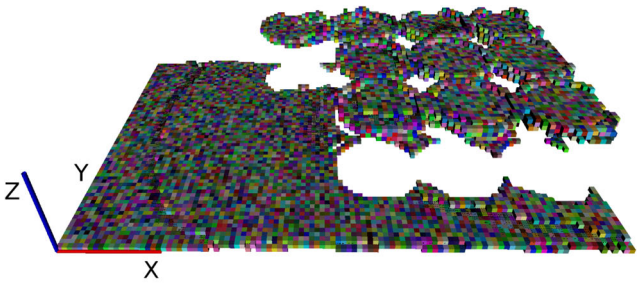
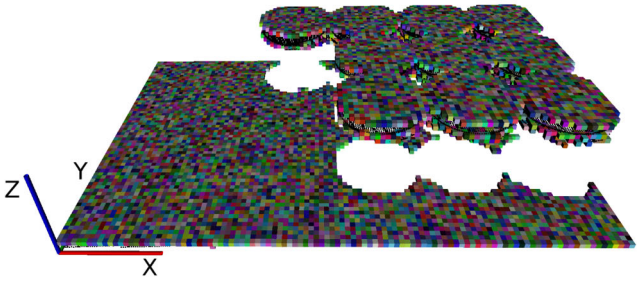


Fig. 6 Side view of the empty cubes generated in an example scene. Green items represent already packed objects and s_1, s_2, s_3, s_4 are the generated EMSs for that scene



(a) Initial occupancy grid with $V_s = 5mm$.



(b) Occupancy grid after the stabilization phase with $H = 5$.

(c) Occupancy grid after square generation phase with $Q = 20$.

Fig. 7 Occupancy grid after each processing phase

occupancy grid is shown in Fig. 7a. The method *occupied* indicates the occupancy of a voxel $v_{i,j,k}$ on the occupancy grid and is defined in Eq. (1).

$$occupied(\mathbb{O}, i, j, k) = \begin{cases} \text{true}, & \text{if } \mathbb{O}[i, j, k] == 1 \\ \text{false}, & \text{if } \mathbb{O}[i, j, k] == 0 \end{cases} \quad (1)$$

When the items are cubic and the mosaic representations used are ideal and not based on real observations, the candidate EMSs are finite. However, the possible EMSs are infinite in real 3D scans of the scene, particularly with irregular and rounded items. Thus, the problem is simplified by representing the space as voxels. The smaller the voxels, the higher the resolution but also the system load.

At this point, the algorithm first processes $\mathbb{O}$ and then calculates the empty cubes with the remaining free space on it. The idea behind this method is to smooth and to fill holes in the occupancy grid with the goal of creating regular surfaces and removing not usable space. The processing done at occupancy grid level is parameterised with two thresholds, H and Q that are explained in detail later on. At the end of the voxel-level processing of $\mathbb{O}$, the empty cubes are generated using all the remaining space inside the box. Indeed, each point in the empty space must belong to one and only one cubic solution. Finally, if J is enabled, a cube-level processing is applied to join neighbour cubes that are at the same height, to maximise their size.

The proposed heuristic method is shown in Algorithm 1 and is composed of four main steps. As explained before, the first processing steps are performed at voxel-level, directly modifying the occupancy grid. However, the final step is performed at the cube level.

The second step of the proposed heuristic method, dubbed the *heightmap stabilisation* phase, makes use of a height threshold H to smooth $\mathbb{O}$ (see Algorithm 2).This method starts from the highest occupied voxel $v_{i,j,k}$ and

Input: CL , V_s,H, Q,J, bin_d, bin_p
Result: List of empty cubic volumes S
initialisation;
Function `Main(`CL , V_s, H, Q, J, bin_d, bin_p`)`**:**
  `// Step 1: Preprocessing`
  $\mathbb{O} \leftarrow$ `preprocessPointCloud(`CL, V_s, bin_d, bin_p`);`
  `// Step 2: Heightmap stabilisation`
  $\mathbb{O} \leftarrow$ `stabiliseOccupancyGrid(`$\mathbb{O}, H, V_s, bin_d$`);`
  `// Step 3: Square generation`
  $\mathbb{O} \leftarrow$ `squareOccupancyGrid(`$\mathbb{O}, Q, V_s, bin_d$`);`
  `// Step 4: Generate solutions`
  $S \leftarrow$ `generateSolutions(`$\mathbb{O}, V_s, bin_d$`);`
  if J **then**
    $S \leftarrow$ `joinSolutions(`S`);`
  end
  return S;

Algorithm 1 Calculate the list of empty cubes S.

finds the values $\alpha, \beta, \gamma, \delta \geq 0$ to update the occupancy of the neighbour voxels from $v_{i-\alpha,j-\beta,k}$ to $v_{i+\gamma,j+\delta,k}$ and set them as occupied if:

$$D_z(a, b, k) \leq H \text{where}, a = \{i - \alpha, ..., i + \beta\}, \quad b = \{j - \gamma, ..., j + \delta\} \quad (2)$$

The function D_z measures the distance in voxels from a voxel $v_{i,j,k}$ to the closest occupied voxel below it and in the

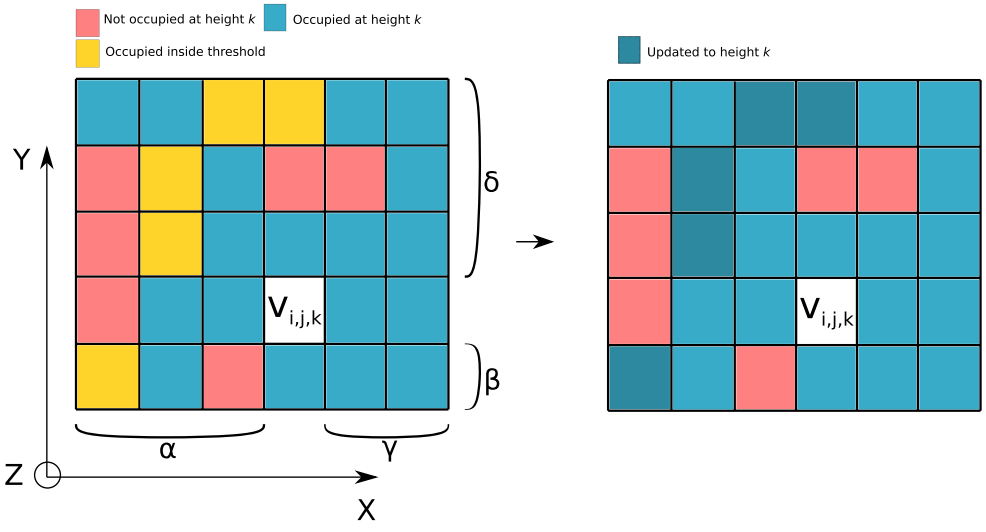


Fig. 8 Square formed by the voxels $v_{i-\alpha,j-\beta,k}$ and $v_{i+\gamma,j+\delta,k}$ at height k of the occupancy grid. In blue, the occupied voxels at height k. In yellow, the voxels where $D_z \leq H$, which are updated to height k. In red, the voxels that are not occupied at height k and $D_z > H$

same column i, j, and is defined in Eq. (3).

$$D_z(i, j, k) = 1 + \sum_{w=1}^{k} 1 \text{ if} \neg\, occupied(\mathbb{O}, i, j, k - w) \quad (3)$$

The values $\alpha, \beta, \gamma, \delta$ are calculated in such a way that in the square formed by the corner voxels $v_{i-\alpha,j-\beta,k}$ and $v_{i+\gamma,j+\delta,k}$, in each row and column at least one voxel is at height k or can be updated to height k (see Fig. 8).

The voxel columns whose occupancy has been changed are marked as modified and this process is executed iteratively until all the voxel columns are analysed. An example of the status of the occupancy grid after the stabilisation phase is depicted in Fig. 7b.

Input: $\mathbb{O}$, H, V_s, bin_d
Result: Stabilised occupancy grid $\mathbb{O}$.
Function
`stabiliseOccupancyGrid(`$\mathbb{O}, H, V_s, bin_d$`)`**:**
  $x\_voxels \leftarrow bin_d[0]/V_s$;
  $y\_voxels \leftarrow bin_d[1]/V_s$;
  $z\_voxels \leftarrow bin_d[2]/V_s$;
  $modified \leftarrow false$;
  // $bool\_array(x\_voxels, y\_voxels)$
  for $k \in \{z\_voxels - 1\ , ...,\ 0\}$ **do**
    for $j \in \{0\ , ...,\ y\_voxels - 1\}$ **do**
      for $i \in \{0\ , ...,\ x\_voxels - 1\}$ **do**
        if $occupied(\mathbb{O}, i, j, k)$ & $\neg modified[i][j]$ **then**
          Get $\alpha, \beta, \gamma, \delta$;
          for $a \in \{i - \alpha, ..., i + \beta\}, b \in \{j - \gamma, ..., j + \delta\}$ **do**
            if $D_z(a, b, k) \leq H$ **then**
              // Update the height of the column a, b to k
              `updateHeight(`$\mathbb{O}, a, b, k$`)` ;
              $modified[a, b] \leftarrow true$;
            end
          end
        end
      end
    end
  end
  return $\mathbb{O}$;

Algorithm 2 Occupancy grid stabilisation method.

The third processing step, called the *square generation* phase, is in charge of filling holes and giving squared shape to the planes generated in the stabilisation step with the intuition of removing not usable space. As the final empty cubes have a square floor plan and are generated only over occupied planes using all the remaining free space, it is important to fill the occupancy grid creating planes with regular shape. To that end, a second threshold Q is used. This threshold is applied in the x and y axes and is used to determine the maximum size of the holes that will be filled in each plane, without considering the vertical distance D_z to the closest occupied voxel. Therefore, the maximum number of consecutive empty voxels that are filled in each axis is Q, which could lead to filling holes up to an area of Q^2 voxels, considering both x and y axes. The main goal of filling small holes in the planes generated in the stabilisation phase is to reduce unusable small volumes and to generate bigger planes that will contribute to the generation of bigger empty cubic volumes.

This method also starts in the highest occupied voxel $v_{i,j,k}$ in $\mathbb{O}$, and finds the values $\epsilon, \zeta, \eta, \theta \geq 0$ in such a way that from $v_{i-\epsilon,j-\zeta,k}$ to $v_{i+\eta,j+\theta,k}$ there is no hole with a larger dimension (x or y) than Q (Fig. 9). The analysed voxels are marked as visited and the same process is executed until all the voxels are processed (see Algorithm 3). An example of the result of this process is shown in Fig. 7c.

After executing the voxel-level operations, all the remaining free space above the occupied zone is assumed to be usable space. At this point, cubic solutions are generated and those are further processed to create solutions as big as possible.

The solution generation method starts in the highest empty voxel that is over an occupied plane, and computes the dimensions of the plane that it belongs to. The height of the cube is determined by the distance between each plane and the bin's top. The solutions are generated in such a way that (1) all the voxels under the floor of each cubic solution must be occupied and, (2) each empty voxel inside the bin must belong to one solution only. The procedure to generate the cubic solutions is described in Algorithm 4.

As the cubic solutions are generated in a non-optimal way, joining them can lead to maximised size solutions. Therefore, in a final step, the neighbour solutions with the origin at the same height in the z axis are combined in pairs by following these criteria:

1. If the cubic solutions have the same origin in axes x or y and have the same dimensions in the corresponding axis, the cubes are joined to generate a bigger cube (see Fig. 10a).
2. If the cubic solutions have the same origin in axes x or y but not the same dimension in the corresponding axis, then two new cubes are generated if the union of both cubes results in a bigger cube than every single one (see Fig. 10b).

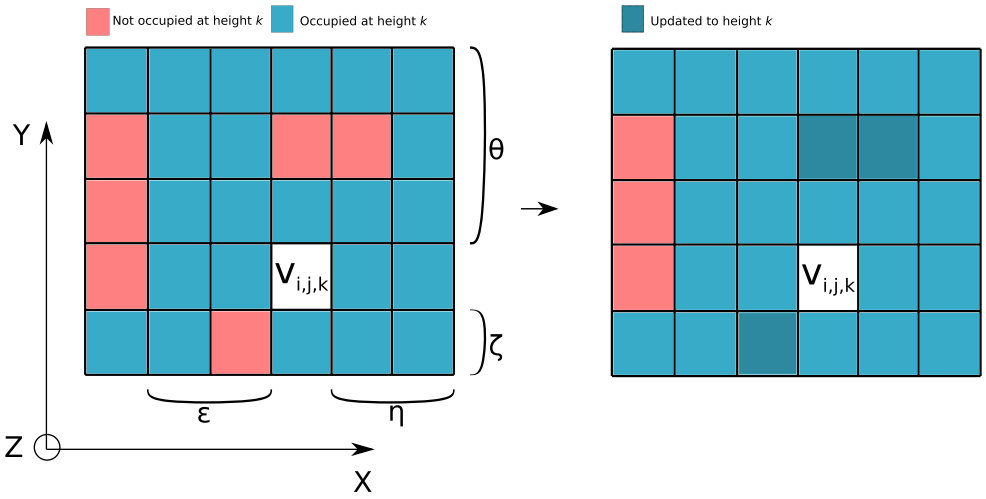


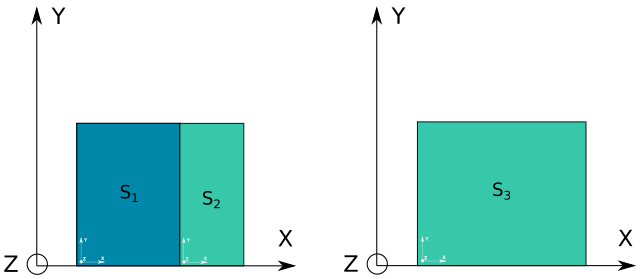
Fig. 9 Example with $Q = 2$. The voxels in the square formed by $v_{i-\epsilon,j-\zeta,k}$ and $v_{i+\eta,j+\theta,k}$ are updated to height k without considering the distance D_z

3. If the cubic solutions are neighbours but they do not share a common origin on any axis, then three new cubes are generated if the union of both cubes results in a bigger cube than any original one (see Fig. 10c).

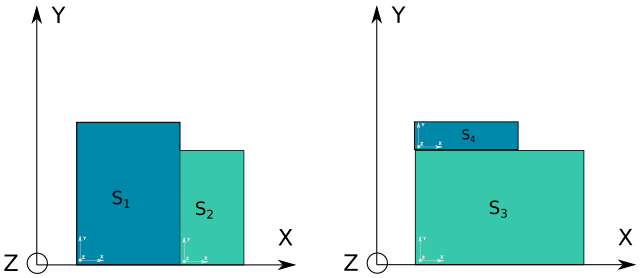
Input: $\mathbb{O}$, Q, V_s, bin_d
Result: Squared occupancy grid $\mathbb{O}$.
Function
`squareOccupancyGrid(`$\mathbb{O}$, Q, V_s, bin_d`):`

```
x_voxels ← bin_d[0]/V_s;
y_voxels ← bin_d[1]/V_s;
z_voxels ← bin_d[2]/V_s;
modified ← false;
// modified : bool_array(x_voxels, y_voxels)
for k ∈ {z_voxels − 1 , ..., 0} do
    for j ∈ {0 , ..., y_voxels − 1} do
        for i ∈ {0 , ..., x_voxels − 1} do
            if occupied(O, i, j, k) &
            ¬modified[i][j] then
                Get ϵ, ζ, η, θ considering Q;
                for a ∈ {i − ϵ, ..., i + ζ}, b ∈
                {j − η, ..., j + θ} do
                    // Update the height
                       of the column a, b
                       to k
                    updateHeight(O, a, b, k);
                    modified[a, b] ← true ;
                end
            end
        end
    end
end
return O;
```

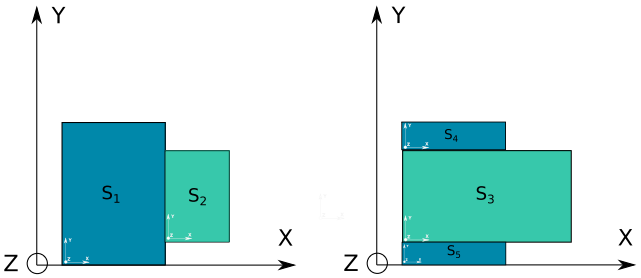
Algorithm 3 Occupancy grid square generation method.



(a) Cubes s_1 and s_2 with common origin and dimension in y axis are joined to generate s_3.



(b) Cubes s_1 and s_2 with same origin in y axis are combined if the combination (s_3) is bigger than s_1 and s_2. As a result s_3 and s_4 are generated.



(c) Cubes s_1 and s_2 are neighbours and are combined if the combination (s_3) is bigger than s_1 and s_2. As a result s_3, s_4 and s_5 are generated.

Fig. 10 Joining criteria for the cubic solutions

```
Input: 𝕆, V_s, bin_d
Result: List of empty cubic volumes S.
Function generateSolutions(𝕆, V_s, bin_d):
    // occupied: Whether voxel v_{i,j,k} is occupied
    // hasObstacleAbove: Whether voxel v_{i,j,k} has any higher occupied voxel in column i,j
    // isInSolution: Whether voxel v_{i,j,k} already belongs to a solution
    // isAboveObstacle: Whether voxel v_{i,j,k} has any lower occupied voxel in column i,j
    x_voxels ← bin_d[0]/V_s;
    y_voxels ← bin_d[1]/V_s;
    z_voxels ← bin_d[2]/V_s;
    S ← [ ];
    inSolution ← false ;
    // inSolution : bool_array(x_voxels, y_voxels)
    for k ∈ {z_voxels − 1 , ..., 0} do
        for j ∈ {0 , ..., y_voxels − 1} do
            for i ∈ {0 , ..., x_voxels − 1} do
                if ¬occupied(𝕆, i, j, k) &
                ¬hasObstacleAbove(𝕆, i, j, k) &
                ¬inSolution[i, j] &
                isAboveObstacle(𝕆, i, j, k) then
                    s_o ← [i, j, k];
                    // Calculate the dimensions of the cube
                    s_d ← getSolDims(𝕆, i, j, k);
                    S.insert(s);
                    inSolution[i, j] ← true;
                end
            end
        end
    end
    return S;
```

Algorithm 4 Cubic solution generation method.

The projection of the generated 3D cubic solutions is depicted in Fig. 11.

3.5 Dynamic IK-PAL mosaic planning

IK-PAL is the proprietary mosaic planner algorithm developed by UHS. The first version of this algorithm was an offline mosaic planner that was able to plan full mosaics with regular parts. This offline planner assumed that the items were picked and packed always in the same way, using the same grasping points and that the set of items to be packed was previously known. Moreover, this method was designed to be used with regular parts only, assuming that after each object was placed in the destination box, the mosaic would remain static. In fact, a manual intervention of the operator was needed whenever an unexpected movement occurred or any item fell off. However, this static mosaic assumption does not hold when irregular items are introduced in the system. Thus, the mosaic has to be created online considering the packing status after each packing operation.

Therefore, in the context of the project, a new dynamic version of the algorithm has been designed and implemented. The new version of the mosaic planner gets as input the list of estimated EMSs in the destination bin, as well as the bounding box of the picked item. As a result, it returns the 3D coordinate inside the destination bin where the origin of the bounding box has to be located, besides to the required rotation of the picked item on the z axis. Since the picked parts are represented as boxes, the rotation in z axis is defined by a binary variable, where a rotation of 90° is applied when it is enabled.

The dynamic IK-PAL is an heuristic-based mosaic planner that only considering the current packing status estimates the next optimal packing pose with the following criteria:

1. First, the EMSs where the bounding box fits in are extracted. In each selected empty volume, there are two possible poses for the bounding box: As it is or with the rotation enabled.
2. A tree data structure is generated with all possible solutions.
3. Criteria such as the degree of filling, compactness or the size of the remaining empty volumes are considered to give a rating to them.
4. The solution with the highest rating is selected.

As this module is a proprietary software of UHS that is under license, the implementation details are not available.

4 Experimentation

This section explains the details of the experimentation carried out to validate the developed algorithms.

4.1 Picked part monitoring

The main objective of the experimentation with the picked part monitoring algorithm was to select the appropriate

Fig. 11 Projections of the cubic solutions that represent the empty space inside the bin

parameters for the calculation of the dimensions of the picked object. In addition, we measured the accuracy of the proposed system by comparing its estimates with the measured dimensions of several objects.

As previously explained in Section 3.3, three *RealSense-D435* cameras were used to estimate the dimensions of the object picked up. As expected, the quality of the 3D information they provide was not comparable to industrial cameras such as the *Photoneo Phoxi L* which were used both to detect grasping points and to monitor the destination bin. In fact, such cameras were very sensitive to ambient light conditions. Therefore, the selection of the parameters was done experimentally and taking into account the lighting conditions of the real environment, and these can be seen in Table 1.

To measure the error made by the algorithm, 10 representative objects with different materials and shapes were selected. For each object, once the robot picked it up and once at the monitoring station, a measurement of the object's dimensions was performed. In addition, keeping the robot static in the monitoring volume, the algorithm was run 10 times (since the point cloud of the scene changed slightly at each acquisition), and the average dimensions estimated by the algorithm were calculated. Figure 12 depicts the measurements process with one of the objects used.

As it can be seen in Table 2, the error committed on each axis was approximately 1 cm on average. According to the camera specifications, the error that these cameras make in estimating depth information is 2%. In our case, the objects were approximately 50 cm far from each camera, and thus, the error could be up to 1 cm per axis. Although there were other factors that could introduce noise into the estimation, such as the extrinsic calibration of the cameras or reflections caused by certain materials such as shiny metals or semi-transparent objects, it was concluded that the measurements of the algorithm were accurate and in accordance with the camera specifications.

Table 1 Tuning parameters for the picked part monitoring algorithm

Picked part monitoring	
V_b	$2mm$
F_r	$10mm$
F_n	5
M_d	$600 \times 600 \times 300mm$
N	64

4.2 Destination bin monitoring

At this experimental step, the goal was to find the combination of parameters that best adapted on average to highly variable scenes. For this purpose, the algorithm was evaluated with a set of 168 scenes taken from the dataset published in [1], where the number, type and placement of objects were random. The dataset was composed of scenes, similar to the one showed in Fig. 5, where the localisation of the bin in each scene was also available. To carry out the search for the optimal parameters, we opted for a grid-search approach, where the possible values for each variable were manually defined, and were as follows:

$$\begin{aligned} V_s &= [1, 5, 10] H = [False, \frac{10}{V_s}, \frac{30}{V_s}, \frac{50}{V_s}] \\ Q &= [False, \frac{10}{V_s}, \frac{50}{V_s}, \frac{100}{V_s}] J = [False, True] \end{aligned} \tag{4}$$

The False option in the parameters H, Q and J indicates that the heightmap stabilisation, square generation or/and solution joining processing phases were disabled respectively. In all the tests, both the clustering-based and the statistical outlier filters were enabled to remove the noise from the point cloud scenes. Thus, per each scene 96 combinations were tested, which gave a total of $96 \cdot 168 = 16128$ trials. In each test, the following metrics were used to measure the quality of the generated solutions:

- Usable volume ratio: One of the objectives of the algorithm was to maximise the total usable volume (i.e. the sum of the volume of all the estimated EMSs that represent usable free space). To do so, we measured the

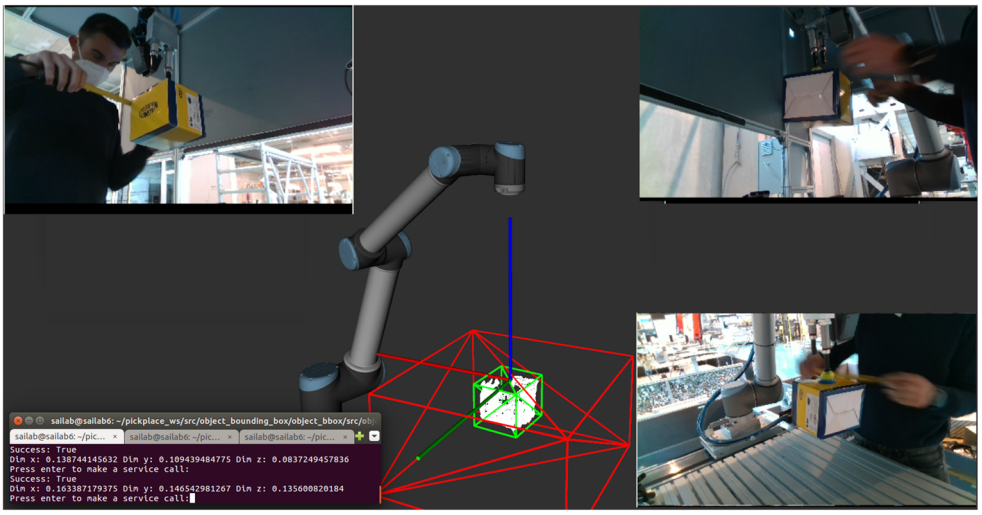


Fig. 12 Bounding box error measurements

ratio of estimated usable space to total space, which told us what percentage of the actual free space was estimated as usable space.

- Execution time: The second objective was to optimise the algorithm's execution time (i.e. given a point cloud of the scene the time needed to estimate the EMSs), which was vital to guarantee the application's cycle time.
- Number of EMSs: The last aspect to optimise was the number of estimated EMSs. Specifically, the aim was to minimise the number of EMSs that represent the usable space inside the box.

Finally, for each of the 96 parameter combinations, the average values of the previously defined metrics were estimated. Since we were interested in optimising the parameters based on three different criteria, following the general approach, we tried to find a balance between them. Figure 13 represents the mean values of all the combinations available in the grid-search process. In red are represented the non-dominated combinations that belong to the Pareto front, which are considered as equally good solutions. Conversely, the combinations shown in blue are dominated solutions that do not belong to the Pareto front. In our case, we selected the parameter combination that is represented by the green star among the non-dominated candidate solutions. The selected combination of parameters and the estimated mean scores per each criteria are shown in Table 3.

To have a clearer view on the behaviour of the algorithm using the selected combination of parameters, we evaluated it in scenes with different complexity. To that end, the validation scenes were divided into three degrees of complexity in which the heuristic was run. Figure 14 shows the execution times, usable volume ratios and the generated EMSs in scenes with low, medium and high complexities.

Table 2 Errors in the bounding box estimation during the picked part monitorisation

Item	Measured		(cm)	Estimated		(cm)	Error		(cm)
	x	y	z	$\overline{x}$	$\overline{y}$	$\overline{z}$	x	y	z
1	10	12	6	10.5	13.8	6	0.5	1.8	0
2	14	12.5	12	16.3	14.6	13.5	2.3	2.1	1.5
3	20.5	20.5	16.5	22.2	21.8	18.1	1.7	1.3	1.6
4	39	18.5	6	40.3	18.7	7.6	1.3	0.2	1.6
5	14.6	23	14	14.1	22	12	0.5	1	2
6	32	20	24.5	31.5	19.7	23.2	0.5	0.3	1.3
7	16.5	21.5	10	17.3	22.1	9.4	0.8	0.6	0.6
8	10	6.5	5	11.6	7.8	5.7	1.6	1.3	0.7
9	17	18	13	17	17.2	13.9	0	0.8	0.9
10	35.5	11	13	37.5	12	13.5	2	1	0.5
$\overline{x}$							1.12	1.04	1.07

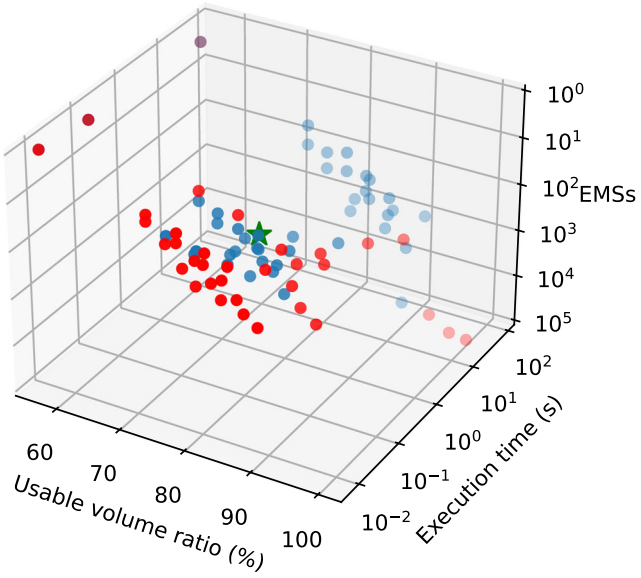


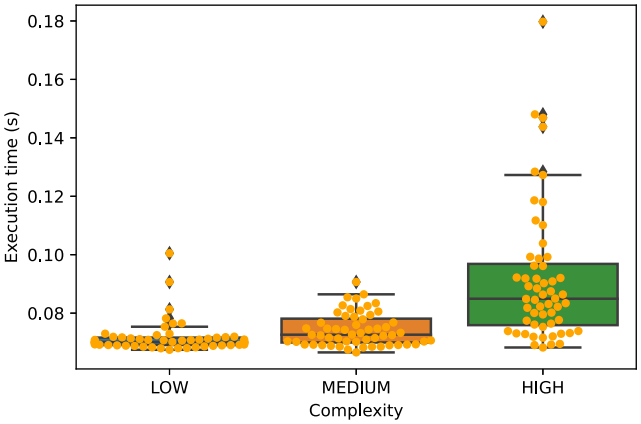
Fig. 13 The scores of the 96 parameter combinations considering the average usable volume ratio, average execution time and the average number of generated EMSs. In red, the non-dominated parameter combinations that belong to the Pareto front. The green star represents the scores of the selected parameter combination

The scenes with less than 10 objects were categorised as of "low complexity", scenes with 10 to 20 objects as of "medium complexity" and the scenes with more than 20 objects as of "high complexity".

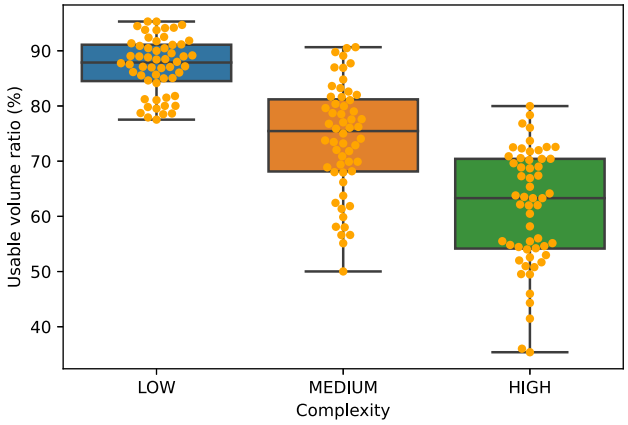
As far as execution times are concerned (see Fig. 14a), although it can be seen that on average there was a slight increase as the complexity of the scene increased, we can see that between the best case of "low complexity" and the worst case of "high complexity", there was an increase of approximately three times. The same occurred within the "high complexity" scenes that although there was not much variation between the number of objects, the execution time did vary. This happened because the computational cost was not directly related to the number of objects in the scene but rather to the regularity of the scene. Despite the scene itself, it did not have any effect on the processing steps performed at the occupancy grid level, the more irregular the scene was, the higher was the number of EMSs generated (see Fig. 14c). An increase in the number of EMSs generated had an effect on the solution joining phase where adjacent

Table 3 Tuning parameters and estimated mean scores for the destination bin monitoring heuristic

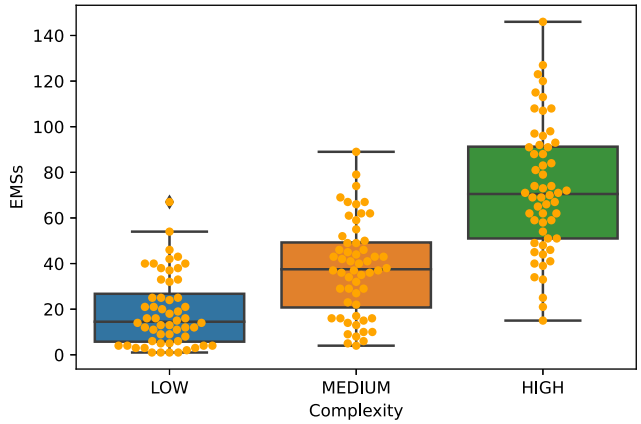
Selected parameters	
V_s	$5mm$
H	2
Q	20
J	$True$
Estimated mean scores	
Usable volume ratio	83.67%
Execution time	0.074s
EMSs	26.9



(a) Execution times.



(b) Usable volume ratios.



(c) Generated EMSs.

Fig. 14 Execution times, usable volume ratios and generated EMSs with respect to scenes with low, medium and high complexity

solutions were exhaustively searched for in order to join them, causing an increase of the execution time. In general, scenes with more objects tended to be more irregular, which also affected the usable volume ratio. This was due to the fact that in irregular scenes, there were many gaps that were not usable and were eliminated in the stabilisation and square generation phases (see Fig. 14b).

As we have seen, as the complexity of the scenes increased, the number of EMSs generated also increased. Therefore, we also measured the average volume of the generated EMSs among the 168 scenes to see how it changed as the complexity of the scene incremented. In order to get an idea of their size, we compared them with the smallest, the medium-sized and the largest objects among the representative objects selected in Section 4.1, and split them into 4 groups. In addition, we also measured the percentage of the total volume each of the groups represented. Figure 15 shows the percentage of EMSs generated in each of the 4 groups and the percentage of the total volume they represent for scenes with low, medium and high complexity.

As it can be seen in Fig. 15, the average percentage of EMSs with a volume smaller than the smallest object ranged from 36 to 63% as the complexity of the scenes increased. Although these were relatively high percentages since these volumes were not usable, the total volume they represented ranged from 0.8 to 6% in the worst case. In the opposite case, the percentage of EMSs generated with a higher volume than the largest object ranged from 36.5% in the simplest scenes to 4.3% in complex scenes. This indicates that when the scenes were less complex,

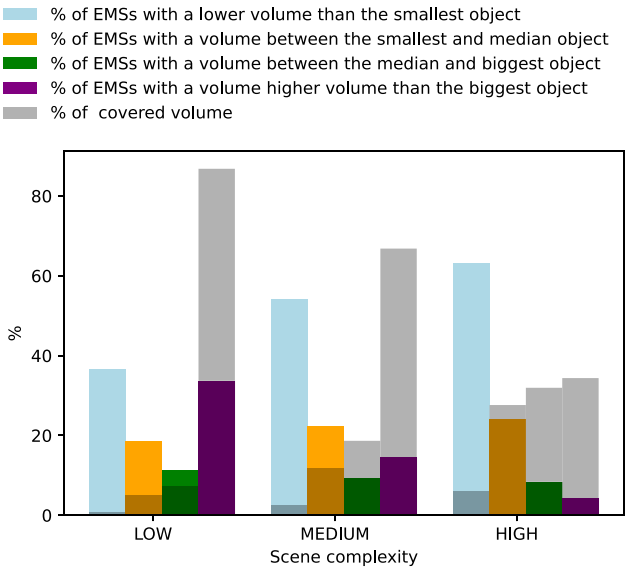


Fig. 15 Size and covered volume of the generated EMSs with respect to the smallest, median and biggest objects in scenes with low, medium and high complexity

the algorithm was able to generate large volumes, which on average represented the 86.8% of the total volume. However, as complexity increased, the number of large EMSs decreased to 4% but still represented 34.3% of the space. In summary, we can say that the size of the EMSs generated was directly related to the complexity of the scene. Although the algorithm had the tendency to generate small solutions, they represented a small percentage of the total volume. In general, these were small unusable spaces that could not be joined with larger EMSs due to the current configuration of the algorithm, but this effect could be reduced by increasing the H and Q thresholds. Figure 16 shows an example where due to the rounded shape of the bottle, a solution with a very small volume was generated.

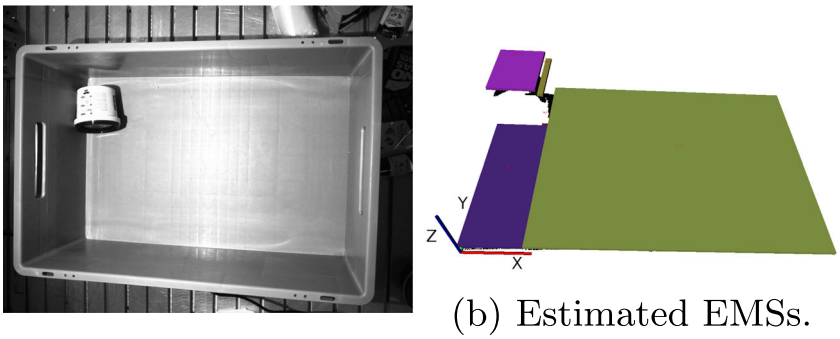
5 Validation in the real robotic system

The validation of the complete packing system was carried out in the TOFAŞ pilot. To that end, the developed modules were deployed in the real system and those were integrated with the automatic picking system previously mentioned. We selected the multi-reference to multi-reference use-case as it is the most complex and provides a clear view of the strengths and weaknesses of the system. For the sake of simplicity, we only used a single bin in both inbound and outbound areas.

To validate the system, we performed 7 different tests which were divided in three blocks (easy, medium and complex). The main goal of these tests was to show the flexibility and usefulness of the system to create mosaics with several types of objects and configurations inside the inbound bin, without the need of ad-hoc configurations. Since there was no similar online bin-packing system publicly available, we recorded the tests performed to demonstrate the usefulness of the system. The tuning parameters used during the validation of the complete packing system are those that were selected during the experimentation, detailed in Section 4. Note that we used the same configuration in all the tests.

We performed the following tests:

- **Easy:**
  1. *Test 1:* Mono-reference boxes were placed in the inbound box, all of them with the same orientation.
  2. *Test 2:* Multi-reference boxes were manually thrown inside the inbound box, ordered from big to small considering their volume.
- **Medium:**
  1. *Test 3:* Multi-reference boxes were mixed in the inbound box, all of them in the most stable



(a) Image of the scene. (b) Estimated EMSs.

Fig. 16 Example of the generation of a small not usable space due to the rounded shape of the bottle

orientation. Once the algorithm was unable to pack more objects in that orientation, the orientation of the remaining objects was manually changed.

2. *Test 4:* Multi-reference cans and bottles were placed in the inbound box, ordered from big to small considering their volume.

- **Complex:**

1. *Test 5:* Multi-reference boxes were placed in the inbound box, with random order and orientation.
2. *Test 6:* A reference of boxes and another reference of bottles were mixed in the inbound box.
3. *Test 7:* Random objects were randomly thrown into the inbound box.

Figure 17 shows the flow of the bin-packing application during the tests, which were conducted in the following way:

1. The number of objects for each test was selected based on the requirements of the test and the availability of the type of objects required.
2. All the items were put in the inbound box at the beginning of the test except in test 2 and test 4, where items were manually placed (ordered from big to small).
3. At each iteration, the grasping point detection module decided which object to pick. If the grasp failed, the next grasping point was selected and a new grasp was executed. This procedure was repeated $N = 3$ times, and if the system was not able to pick any object, the test was concluded.
4. The objects were packed in the same order that had been picked. We did not perform any object identification since a single destination bin was used and all the picked items were packed in the same bin.
5. The test ended when there were not objects left in the input bin, when the robot was not able to pick more objects or when the system was not able to pack the picked item.

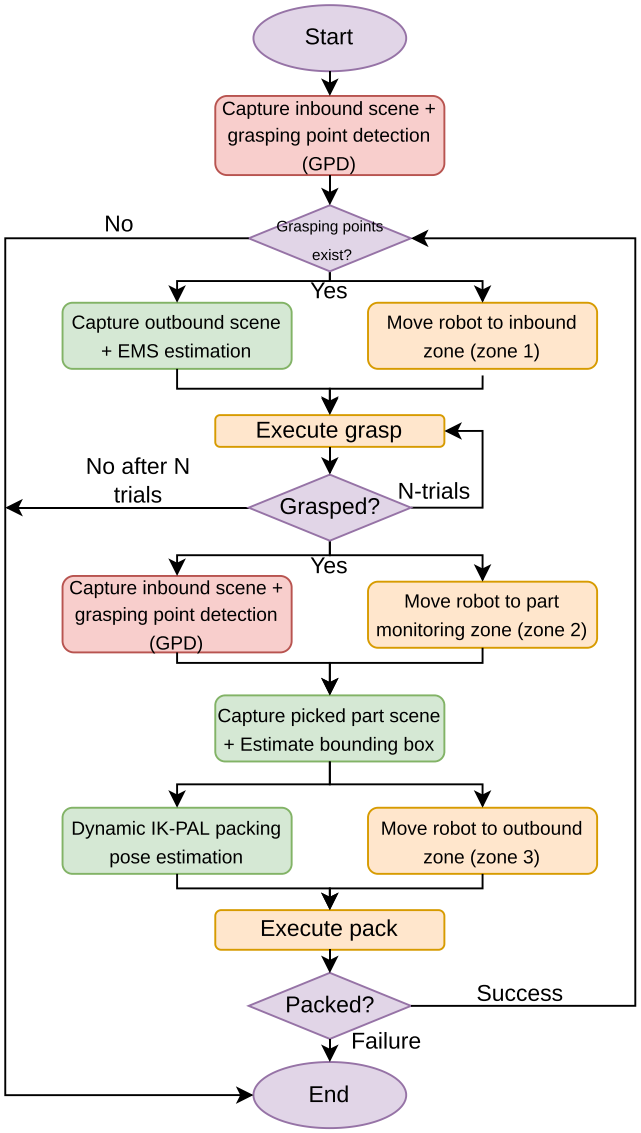


Fig. 17 Flowchart of the complete bin-packing application. Red boxes indicate tasks related to the grasping of the objects. Green boxes belong to tasks related to the packing of the picked part. In orange the tasks that imply the motion of the robot

The tests were recorded and the videos are publicly available at the following link: https://bit.ly/3aWc0ke.

In the recorded videos, it can be observed that the main strength of the system was the flexibility, being able to pack online a wide variety of items. The analysis of the destination bin at each iteration let us consider fallen objects or unexpected movements inside the bin that happened after releasing the parts.

Concerning the first test, the aim was to see how the system behaves with single reference boxes placed in their most stable orientation. Although due to their low weight some objects moved/rotated slightly during placement, these movements were taken into account in each iteration, thanks to the monitoring of the target box. However, as a consequence of these rotations, the separation between objects increased. The constructed mosaic was stable due to the orientation of the products in the inbound bin.

In the second and third tests, the system was assessed with multi-reference boxes. Although in both cases the system was able to generate a mosaic in a flexible way and taking into account the variations caused by the robotic packing system, as expected the packing order had a great effect on the final result. In both tests, it can be appreciated that, when certain objects were packed, they sometimes touched nearby objects. This happened due to the error introduced in the monitorisation of the picked part, since the dimension estimates were smaller than the real ones. Although this caused the movement of nearby objects inside the box, the system took this uncertainty into account when packing the next object.

Although we have seen that the system generates stable mosaics with multi-reference boxes placed in their most stable pose, this is not the case when the orientation of the boxes is random (test 5). In this case, the generated mosaic was unstable, which caused some objects to fall when they were packed on top of other unstable objects. Although the system successfully considered such movements in the target box, the quality of the final mosaic decreased considerably.

In a similar way to test 2, in test 4, we also tried to evaluate the behaviour of the system when the objects were ordered from bigger to smaller, but this time using cans and bottles. The system worked correctly, and although some small friction occurred when placing an object, which modified the final pose of the object to be left, the system took it into account correctly in the following iteration. As can be guessed, it is also key the cans and bottles to be ordered from larger to smaller the constructed mosaic to be stable.

In test 6, a box reference was mixed with a bottle reference in the inbound box, and the picking system determined the packing order. As can be seen, the generated mosaic was irregular and sometimes unstable, particularly when objects of different sizes were packed. This caused some objects to fall after placing an object, making it even harder to generate a stable mosaic. Although the separation between objects and, in general, the use of space could be improved, it is clear that the strong point of the system is the ability to adapt to unforeseen situations and to plan online taking into account these uncertainties.

Finally, in test 7, we sought to demonstrate the ability of the system to pack a wide variety of objects without the need to configure the system for each reference. It can be seen that the system showed great flexibility and was capable of dynamically packing objects taken in random order. Of course, with such a variety of objects which are totally unknown to the system until the moment they are picked, it is difficult to build a stable and space-efficient mosaic.

5.1 Timeliness analysis

As mentioned above, the system has been developed on the ROS framework, which does not guarantee real-time execution of robotic applications. The aim of this test is to evaluate the timely execution of our system and to demonstrate that although we do not use a real-time framework, the system maintains a cycle time without major variations.

For this, the application was run 10 times following the flow shown in Fig. 17. In each run, 10 objects that were easily picked with the suction tool (boxes and cans) were selected and randomly dropped into the input container. In each picking/packing cycle, the execution times of each main component of the application were recorded, which are shown in Table 4. The measurement of the average total execution time has been made taking into account that not all tasks are executed sequentially. As can be seen in Fig. 17, while the robot is moving from one area to another, other tasks are carried out in parallel, thus reducing the cycle time. The target cycle time was set at 25 s.

A total of $10 \cdot 10 = 100$ picking/packing cycles were executed with an average execution time of $22.32s$. Considering the worst case, the cycle time was $24.70s$, which supposes an increase of $2.38s$ compared to the average. However, we can see that even in the worst case scenario, the system meets the cycle time. It should be noted that the maximum execution times for each phase may not have been obtained in the same run.

As can be seen, the most time-consuming phases were especially those that required the movement of the robot (i.e. phases 3, 4, 5, 8 and 9). Especially, the most time-consuming phase was the execution of grasping, as the robot makes several grasping attempts in case it is not able to grasp an object the first time. The phases of grasping point identification and EMS estimation were executed in parallel while the robot is moving between zones, thus not

Table 4 Average, standard deviation and maximum duration of each phase of the application

Phase	Task	$\bar{t}(s)$	$\sigma(s)$	Max.(s)
1	Capture inbound scene	0.8151	0.0545	0.9437
	GPD	0.3018	0.0210	0.3555
2	Capture outbound scene	0.8248	0.0492	0.9736
	EMS estimation	0.071	0.0302	0.1819
3	Move to zone 1	2.9987	0.0212	3.0680
4	Execute grasp	4.711	1.1455	7.3174
5	Move to zone 2	4.8324	0.0200	4.8899
6	Capture bbox scene	0.3002	0.0435	0.3814
	Estimate bbox	0.1169	0.0365	0.1836
7	Packing pose estimation	0.2089	0.0915	0.4638
8	Move to zone 3	5.4005	0.0198	5.4445
9	Execute pack	3.9558	0.2968	4.7045
Total	–	22.326	1.1689	24.7017

influencing the cycle time. This was possible because the computation time required for these two phases, even in the worst case, was considerably less than the duration of the robot's movement. However, the monitoring of the grasped object did take place online, and in the worst case, it could take up to $0.56s$.

The cycle times obtained for our system show that there is little variation in time. Furthermore, it can be seen that even in the worst case our system meets the target cycle time.

6 Conclusion and further work

This work describes the dynamic bin-packing system that was developed and validated in the context of the PICKPLACE European project. To the best of our knowledge, this is the first time a complete online bin-packing system has been deployed to a real scenario which allows creating mosaics with arbitrary objects and considering the dynamics of a real robotic packing system.

The software architecture is based on ROS and follows the layered architecture proposed in *Robotframework*. In this work, three main modules of the packing system are presented that let us pack a wide variety of objects in an online manner and without the need of ad hoc configurations. The developed system has been; successfully deployed and validated in the real robotic environment, where 7 tests were performed and recorded in video.

In the recorded videos, it can be observed that the main strength of the system was the flexibility, being able to pack online a wide variety of items. The analysis of the destination bin at each iteration let us consider fallen objects or unexpected movements inside the bin that happened after releasing the parts. In spite of the fact that the system had shown to be useful and flexible, there were some limitations that led to a sub-optimal space utilisation in the destination bin:

Picking order and part orientation As expected, the picking order had a large effect in the final mosaic. As can be seen in tests 2 and 4, when the products were manually ordered from big to small forcing the robot to pick them in order, the quality of the final mosaic improved. In addition, the orientation of the products in the inbound box had a big impact in the stability of the mosaic. Nevertheless, those are aspects that directly affect both the quality/stability of the mosaic and the space utilisation, but are difficult to improve in a multi-reference to multi-reference setup. The grasping system has the capability to decide to some extent the picking order, but this is limited to the visible and graspable objects in the inbound bin.

Inaccurate bounding box estimation Although very accurate industrial cameras were used for both grasping point estimation and outbound box monitoring, this was not the case for the bounding box estimation, where cheaper *RealSense D435* cameras were used. The depth quality of those cameras was not good enough (up to 1 cm of error in our bounding box estimation region) and usually led to bigger estimations. This caused the object to be released in an inaccurate pose and this had a direct effect in the separation between objects.

Product's weight On the one hand, depending on the weight of the product, the movement of the robot towards the bounding box estimation area sometimes caused the product to oscillate. This oscillation led to inaccuracies in the

estimation of the bounding box. On the other hand, if the height of the product was not properly estimated and the product was released further than expected, the weight of the dropped product could cause the object to end in an unexpected final position. Although the online outbound container monitoring considered these movements for the next iterations, the quality of the mosaic was negatively affected.

Occlusions The online mosaic planner decided the optimal release position for the picked part at each iteration, only based on the outbound box monitoring and the bounding box estimation. Due to several criteria that are used to determine the release position, many times object towers were created which increased the likelihood to create occlusions.

External lighting The changing lighting conditions had a large effect on the 3D depth cameras. Although the pilot was covered with a housing and some noise filtering algorithms were used, this caused an increase in the sensor noise. This led to inaccurate depth estimations, particularly with shiny objects, which could cause collisions in the destination box.

Regarding the timeliness of the system, the experimentation carried out shows that there is very little variation in the cycle times. However, we are aware that in order to industrialise such an application and guarantee a fixed cycle time, a real-time implementation in a framework such as ROS2 [26] would have to be used.

In this article, we only considered the suction end effector to pick and place the items; nonetheless, it would be interesting to also consider other end effectors such as grippers. The gripper, compared to suction, needs at least two contact points to manipulate the items and this would introduce more complexity to the system. Furthermore, those contact points are usually located on the perimeter of the parts, which complicates the packing of objects close to each other.

The online mosaic planner plays a key role in the mosaic generation process since it is in charge of deciding the final position for each picked object. This work was conditioned by the UHS's dynamic IK-PAL planner. Although in most cases it offered coherent results, sometimes it was difficult to understand the reason behind some decisions made by the planner. Therefore, the use of an open source planner would improve the traceability of the system. To the best of our knowledge, there is no planner with the requirements of our system, and this would require the implementation of a new planner from scratch.

Author contribution Conceptualisation: A.I., A.F. and I.M.; methodology: A.I., E.L. and A.A.; implementation: A.I. and A.F.; original draft preparation: A.I.; review and editing: E.L. and A.A.; supervision: I.M.

Funding This article has been funded by the European Union's Horizon 2020 research and Innovation Programme under grant agreement No. 780488, and the project "5R- Red Cervera de Tecnologías robóticas en fabricación inteligente", contract number CER-20211007, under "Centros Tecnológicos de Excelencia Cervera" programme funded by "The Centre for the Development of Industrial Technology (CDTI)".

Declarations

Competing interests The authors declare no competing interests.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Iriondo A, Lazkano E, Ansuategi A (2021) Affordance-based grasping point detection using graph convolutional networks for industrial bin-picking applications. Sensors 21(3):816. https://doi.org/10.3390/s21030816
2. Martello S, Vigo D (1998) Exact solution of the two-dimensional finite bin packing problem. Manag Sci 44(3):388–399. https://doi.org/10.1287/mnsc.44.3.388
3. Martello S, Pisinger D, Vigo D (2000) The three-dimensional bin packing problem. Oper Res 48(2):256–267. https://doi.org/10.1287/opre.48.2.256.12386
4. Den Boef E, Korst J, Martello S, Pisinger D, Vigo D (2005) Erratum to the three-dimensional bin packing problem: robot-packable and orthogonal variants of packing problems. Oper Res 53(4):735–736. https://doi.org/10.1287/opre.1050.0210
5. Coffman EG, Garey MR, Johnson DS (1984). In: Ausiello G, Lucertini M, Serafini P (eds) Approximation algorithms for bin-packing–an updated survey. Vienna: Springer Vienna; p 49–106. Available from. https://doi.org/10.1007/978-3-7091-4338-4_3
6. Jakobs S. (1996) On genetic algorithms for the packing of polygons. Eur J Oper Res 88(1):165–181. https://doi.org/10.1016/0377-2217(94)00166-9
7. Johnson DS, Demers A, Ullman JD, Garey MR, Graham RL (1974) Worst-case performance bounds for simple one-dimensional packing algorithms. J Comput 3(4):299–325. https://doi.org/10.1137/0203025
8. Ali S, Ramos AG, Carravilla MA, Oliveira JF (2022) On-line three-dimensional packing problems: a review of off-line and on-line solution approaches. Comput Ind Eng. p 108122. https://doi.org/10.1016/j.cie.2022.108122
9. Crainic TG, Perboli G, Tadei R (2009) TS2PACK: a two-level tabu search for the three-dimensional bin packing problem. Eur J Oper Res 195(3):744–760. https://doi.org/10.1016/j.ejor.2007.06.063
10. Faroe O, Pisinger D, Zachariasen M (2003) Guided local search for the three-dimensional bin-packing problem. INFORMS J

Comput 15(3):267–283. https://doi.org/10.1287/ijoc.15.3.267.16080
11. Gonċalves JF, Resende MG (2013) A biased random key genetic algorithm for 2D and 3D bin packing problems. Int J Prod Econ 145(2):500–510. https://doi.org/10.1016/j.ijpe.2013.04.019
12. Hu H, Zhang X, Yan X, Wang L, Xu Y (2017) Solving a new 3d bin packing problem with deep reinforcement learning method. ArXiv.1708.05930
13. Zhao Y, Rausch C, Haas C (2021) Optimizing 3D irregular object packing from 3D scans using metaheuristics. Adv Eng Inform 47:101234. https://doi.org/10.1016/j.aei.2020.101234
14. Wang F, Hauser K (2019) Stable bin packing of non-convex 3D objects with a robot manipulator. In: International conference on robotics and automation (ICRA). IEEE; p 8698–8704 Available from. https://doi.org/10.1109/ICRA.2019.8794049
15. Ha CT, Nguyen TT, Bui LT, Wang R (2017) An online packing heuristic for the three-dimensional container loading problem in dynamic environments and the physical internet. In: European conference on the applications of evolutionary computation. Springer; p 140–155. Available from. https://doi.org/10.1007/978-3-319-55792-2_10
16. Wang F, Hauser K (2020) Robot packing with known items and nondeterministic arrival order. Trans Autom Sci Eng 18(4):1901–1915. https://doi.org/10.1109/TASE.2020.3024291
17. Hong YD, Kim YJ, Lee KB (2020) Smart pack: online autonomous object-packing system using RGB-D sensor data. Sensors 20(16):4448. https://doi.org/10.3390/s20164448
18. Duan L, Hu H, Qian Y, Gong Y, Zhang X, Wei J et al (2019) A multi-task selected learning approach for solving 3D flexible bin packing problem. In: Proceedings of the international conference on autonomous systems and multiagent systems (AAMAS) Available from. https://www.ifaamas.org/Proceedings/aamas2019/pdfs/p1386.pdf
19. Zhao H, Zhu C, Xu X, Huang H, Xu K. (2022) Learning practically feasible policies for online 3D bin packing. Sci China Inf Sci 65(1):1–17. https://doi.org/10.1007/s11432-021-3348-6
20. Quigley M, Conley K, Gerkey B, Faust J, Foote T, Leibs J et al (2009) ROS: an open-source robot operating system. In: ICRA workshop on open source software. vol 3. Kobe, Japan. p 5. Available from. http://robotics.stanford.edu/ang/papers/icraoss09-ROS.pdf
21. Martin J, Ansuategi A, Maurtua I, Gutierrez A, Obregón D, Casquero O et al (2021) A generic ROS-based control architecture for pest inspection and treatment in greenhouses using a mobile manipulator. IEEE Access 9:94981–94995. https://doi.org/10.1109/ACCESS.2021.3093978
22. Parreño F, Alvarez-Valdés R, Oliveira JF, Tamarit J. M. (2010) A hybrid GRASP/VND algorithm for two-and three-dimensional bin packing. Ann Oper Res 179(1):203–220. https://doi.org/10.1007/s10479-008-0449-4
23. Gonçalves JF, Resende MG (2012) A parallel multi-population biased random-key genetic algorithm for a container loading problem. Comput Oper Res 39(2):179–190. https://doi.org/10.1016/j.cor.2011.03.009
24. Susperregi L, Fernandez A, Molina J, Iriondo A, Sierra B, Lazkano E, et al. (2020) RSAII: flexible robotized unitary picking in collaborative environments for order preparation in distribution centers. In: Bringing innovative robotic technologies from research labs to industrial end-users. Springer; p 129–151. Available from. https://doi.org/10.1007/978-3-030-34507-5_6
25. Rusu RB, Cousins S (2011) 3d is here: point cloud library (pcl). In: International conference on robotics and automation. IEEE; p 1–4. Available from. https://pointclouds.org/assets/pdf/pcl_icra2011.pdf
26. Macenski S, Foote T, Gerkey B, Lalancette C, Woodall W (2022) Robot Operating System 2: design, architecture, and uses in the wild. Sci Robot 7(66):eabm6074. https://doi.org/10.1126/scirobotics.abm6074

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

10

Learning positioning policies for mobile manipulation operations with deep reinforcement learning

Authors

Iriondo, Ander; Lazkano, Elena; Ansuategi, Ander; Rivera, Andoni; Lluvia, Iker; Tubio, Carlos;

Publisher

Submitted to Springer. Minor revisions pending.

Journal

International Journal of Machine Learning and Cybernetics

Year

2023

Quartile (Web of Science / Scimago)

Q2/Q2

DOI

-

Learning positioning policies for mobile manipulation operations with deep reinforcement learning

Ander Iriondo [1,2*], Elena Lazkano [2], Ander Ansuategi [1], Andoni Rivera [1], Iker Lluvia [1] and Carlos Tubío [1]

[1]Department of Autonomous and Intelligent Systems, Tekniker - Basque Research and Technology Alliance (BRTA), Iñaki Goenaga, 5, Eibar, 20600, Gipuzkoa, Spain.
[2]Robotics and Autonomous Systems group (RSAIT), Department of Computer Science and Artificial Intelligence, University of the Basque Country (UPV/EHU), Pº Manuel Lardizabal,1, Donostia-San Sebastián, 20018, Gipuzkoa, Spain.

*Corresponding author(s). E-mail(s): ander.iriondo@tekniker.es;
Contributing authors: e.lazkano@ehu.eus;
ander.ansuategi@tekniker.es; andoni.rivera@tekniker.es;
iker.lluvia@tekniker.es; carlos.tubio@tekniker.es;

Abstract

This work focuses on the operation of picking an object on a table with a mobile manipulator. We use deep reinforcement learning (DRL) to learn a positioning policy for the robot's base by considering the reachability constraints of the arm. This work extends our first proof-of-concept with the ultimate goal of validating the method on a real robot. Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm is used to model the base controller, and is optimised using the feedback from the MoveIt! based arm planner. The idea is to encourage the base controller to position itself in areas where the arm reaches the object. Following a simulation-to-reality approach, first we create a realistic simulation of the robotic environment in Unity, and integrate it in Robot Operating System (ROS). The drivers for both the base and the arm are also implemented. The DRL-based agent is trained in simulation and, both

the robot and target poses are randomised to make the learnt base controller robust to uncertainties. We propose a task-specific setup for TD3, which includes state/action spaces, reward function and neural architectures. We compare the proposed method with the baseline work and show that the combination of TD3 and the proposed setup leads to a **11%** higher success rate than with the baseline, with an overall success rate of **97%**. Finally, the learnt agent is deployed and validated in the real robotic system where we obtain a promising success rate of **75%**.

Keywords: Mobile manipulation, pick and place, deep reinforcement learning, sim-to-real transfer

1 Introduction

Pick and place are basic operations in robotics applications, whether in industrial setups (e.g., machine tending, assembly, or bin-picking) or in service robotics domain (e.g. agriculture or home). Although in some structured scenarios pick and place operations have shown a strong performance, this is not the case in less structured uncertain environments. The efficiency and robustness of the solutions are a bottleneck for industrial applications, and those have not reached the market yet.

The majority of robotics applications focus either on navigation aspects of mobile platforms (e.g. industrial transportation systems, guide robots), or the manipulation of goods with robotic arms (e.g., bin-picking applications). Nonetheless, few applications consider mobile manipulation itself combining both robotic tasks. Despite there are several commercial mobile manipulators in the market, there is a lack of real applications due to the complexity and uncertainty introduced by combining both, manipulation and navigation.

The present work focuses on the picking operation of a randomly placed object from a table with a mobile manipulator. Due to the particular morphology of robotic arms, their scope is limited, and not all the positions of the base near the table enable a successful picking.

Traditionally, such mobile manipulation operations have been solved using analytical planning and control methods [1]. These methods require explicit programming of the skills which can be very costly and error-prone particularly in problems where decision making is complex. The performance of these models depends on how well the reality fits the assumptions made by the model. Due to the impossibility of predicting all the cases that may occur in dynamic and unstructured environments, these methods are generally impractical.

In many explicitly programmed mobile manipulation applications, the base navigates first to zones where the object is within reach (only considering distance), and then carry out the picking trial. However, other works such as the one proposed by Stulp et al in [2] challenge this and raise the following questions: *"Does well-in-reach always imply that the target can really be reached, given the hardware and control software of the robot?, Can we have a least-commitment realisation of 'places' such that the robot can refine a 'place' as it learns more about the context (e.g. clutteredness) of the surroundings?, How can such a concept of 'place' take into account uncertainties about the robot's self-localisation and the estimated target position?"*.

As the authors mention, explicit programming to account for these factors is tedious and impractical, and more flexible solutions are needed.

Alternatively, data-driven methods allow learning directly from real data [3]. Recently, the combination of deep learning (DL) and reinforcement learning (RL), known as deep reinforcement learning (DRL), has allowed to tackle complex decision-making problems that were previously unfeasible [4]. In fact, DRL has proven to be the state-of-the-art technology for learning complex robotic behaviours through the interaction with the environment and solely guided by a reward signal.

The proposed method was originally based on the idea proposed in [2]. In that work authors propose to learn a data-driven model of "place" (areas where the base of the mobile manipulator has to be placed to guarantee the reach of the arm to the object) by interacting with the environment and using machine learning (ML). In that way, they take into account the limits of the robot hardware as well as the uncertainty in the localisation of both the robot and the target. While ML-based methods are generally used for offline forecasting, DRL is generally used online in sequential decision-making problems [5]. In fact, DRL allows to autonomously learn complex control policies through trial and error and only guided by a reward signal. In the case of robotics, the most common use case is to use such algorithms to model agents capable of performing continuous control of robots.

In a first proof-of-concept, we extended the idea and learnt a positioning policy for the mobile manipulator's base using state-of-the-art DRL algorithms [6]. We proposed a novel method which was based on learning a DRL-based controller for the base taking into account the reachability constraints of the arm. To do so, we proposed to use feedback from a traditional arm trajectory planner to optimise the base controller. In fact, the learnt base controller was able to drive the mobile manipulator to areas with a high probability of picking success. Although we showed the feasibility of applying DRL to learn such a complex behaviour, the experimentation was carried out only in a simplistic simulated environment, which was far from reality.

Following the same idea, in this work we go one step further and propose a fully improved DRL-based learning framework for the mobile manipulation task, with the final goal of deploying the learnt model in the real robot. The contributions of this article are as follows:

1. Based on our previous work [6], we developed an improved control architecture that allows learning the task in simulation and an easy deployment in the real system. In fact, the proposed approach combines the Twin Delayed Deep Deterministic Policy Gradient (TD3) [7] DRL algorithm to control the base with traditional planning and control algorithms for the arm. To the best of our knowledge, this is the first time that TD3 has been applied to learn a mobile manipulation skill.
2. We propose a novel reward formulation, task-specific state/action space definition, and neural architecture selection for TD3 that lead to a robust and stable controller. The reward function carefully combines dynamic variables related to distance to target, speed/acceleration, collision checking and feedback from the arm planner to encourage the robot to navigate towards areas with a high probability of pick success.
3. We develop a realistic simulation of both the robot and the environment in Unity [8] from scratch. The simulation is integrated into the Robot Operating System

(ROS) [9] following the OpenAI Gym standard [10]. The drivers needed to command both the base and the arm are also developed. Robot and target position randomisation functionalities are also offered through ROS.

4. A thorough evaluation is conducted to assess the proposed method and compare it with the baseline using standard metrics. Finally, the learnt agent is deployed and validated in the real system. The results obtained show the feasibility of the proposed approach.

The mobile manipulator used was previously developed by Tekniker and is depicted in Figure 1.

Fig. 1: The robot, developed by Tekniker, combines a commercial Segway omnidirectional mobile platform and a KUKA iiwa arm.

The rest of the paper is organised as follows. Section 2 reviews the literature. Section 3 explains the details of the design and implementation of the DRL-based control architecture for the mobile manipulator. In Section 4 we benchmark our method with the baseline approach, using multiple DRL algorithms. Then, details about the deployment and validation in the real robotic system are presented in Section 5. Finally, in Section 6 some conclusions are drawn, and the next steps are discussed.

2 Literature review

Traditionally, well known planning and control methods have been widely used for scheduling mobile manipulation behaviours [11], for example using the ROS navigation stack [12] for navigation and MoveIt! [13] for manipulation. For instance, Dömel et al focus on fetch and carry operations in industrial environments [14]. In that work, both the arm and the base are considered as independent systems and,

although an reachability study is carried out, it is done offline and thus ignoring the environment's dynamics. In addition, the reachability study is done for the sole purpose of establishing the design of the application. Xu et al propose to use an inverse kinematics database to estimate the feasible positions of the base to solve a pick-and-place operation with objects stored in trays [15]. However, the database of feasible locations is composed of discrete poses that are estimated offline, and consequently they also ignore the dynamics of the environment. In other methods such as [16–18] both the base and the arm are seen as one and also rely on traditional path planning and control methods to schedule the task. Nevertheless, the computational cost of such a high dimensional planning is very high and although they attempt to model uncertainty in dynamic environments, the system is limited to the expected casuistry when programming the behaviour. Instead of considering the entire robot as a single system, works such as [19, 20] treat the arm and base independently, but use the force feedback from the arm to generate velocity commands for the base.

Although traditional methods have led to promising mobile manipulation skills in some specific tasks, they require the explicit programming of hard-to-engineer behaviours and often fail in more complex tasks where the decision-making process is hard. In addition, such solutions are generally very inflexible and error-prone due to the impossibility of modelling all the uncertainty of dynamic industrial environments when those are programmed.

Alternatively, data-driven approaches address the main limitations of traditional methods and propose to learn robotic behaviours from real experience [3], thus alleviating the cost of modelling complex behaviours. For instance, in one of the first approaches Lin and Goldenberg use DL to model the motion control of a mobile manipulator using real experience [21]. This approach allows them to use deep neural networks to model the uncertainties of the environment, which leads to a more robust controller compared to traditional ones. Later, Konidaris et al propose to use RL to automatise the skill acquisition on a mobile manipulator [22]. Unlike DL, RL allows to automatically obtain the experience needed to learn robotic skills through trial-and-error and allows to learn complex decision-making policies.

Other works such as [2] use ML to learn the positioning of a mobile manipulator. In that work Stulp et al propose to learn a concept of "place" (areas where the base of the mobile manipulator has to be placed to guarantee the reach of the arm to the object) for mobile manipulation operations using ML. Specifically, the authors propose to learn a probabilistic ML model using experience obtained through trial and error. The learnt model allows them to predict offline poses of the base which allow the arm to reach the object. As the authors claim, the explicit modelling of the problem is no longer required since the learnt models are grounded in real experience.

Recently, the combination of DL and RL, also known as DRL, has made it possible to tackle complex decision-making problems that were previously unfeasible. It combines the ability of DL to model very high dimensional data with the ability of RL to model decision-making agents through trial and error. In fact, it has become the technology of choice for learning complex robotic behaviours using the experience gained through interaction with the environment [23]. DRL has been successfully applied in a wide variety of areas such as robotics, computer vision and gaming [4]. Taking into account the difficulty of modelling complex decision-making robotic skills, DRL offers a promising way to take advantage of the experience gathered interacting with the environment to autonomously learn complex robotic behaviours.

In particular, the field of DRL applied to robotics has recently gained popularity due to the remarkable performance obtained in applications with high decision-making and control complexity. Applications range from manipulation [24, 25], to autonomous navigation [26] and locomotion [27, 28].

An example of the potential of DRL based methods applied to robotic manipulation can be seen in [29]. In that work Kalashnikov et al propose a vision-based self-supervised DRL framework, called *Qt-opt*, to learn a hand-eye coordinated grasping policy. In fact, they learn a controller for a real robotic arm to pick both known and novel items, by only using an over-the-shoulder RGB camera. Recently, DRL has been applied to learn to manipulate nonrigid objects such as cloths [30]. Specifically, the method proposed by Jangir et al is based on the Deep Deterministic Policy Gradient (DDPG) [31] DRL algorithm and is only applied in simulation. Recently, Kim et al [32] used the TD3 DRL algorithm to solve the path planning problem with 2/3-DoF manipulators, and showed that TD3 can be used to plan smoother paths compared to traditional algorithms such as Probabilistic Roadmap Planning [33]. In that work the experimentation is carried out fully in simulation.

DRL has also been successfully applied to learn robot navigation policies. For instance the problem of mapless navigation is tackled in [34], where a velocity controller for the mobile base is learnt making use of an asynchronous variant of DDPG. In fact, the controller is trained in simulation using as input 10-dimensional laser sensor readings, besides to the relative position of the robot with respect to the target, and predicts the target linear and angular velocities for the mobile base. The method is finally assessed in the real robotic system. The problem of large-scale 3D navigation of unmanned aerial vehicles is also addressed in [35] using a recurrent variant of the DDPG algorithm, called RDPG. In that work, authors use RDPG to solve the 3D large scale navigation as a Partially Observable Markov Decision Process (POMDP), with partially observable and uncertain states. Their system is only assessed in simulation.

Concerning locomotion applications, TD3 has also been successfully used to learn the continuous control of biped and quadruped robots in simulation [36, 37]. Unlike the general approach of training DRL agents in simulation, Haarnoja et al showed that its possible to learn complex control skills with real-world experience. In fact, authors designed the Soft Actor Critic (SAC) sample efficient algorithm and applied it to learn locomotion skills in the real world [27].

Although much work was done in the fields of manipulation, navigation and locomotion, the application of DRL in mobile manipulation was a totally unexplored world until recently. Seeing the excellent results that DRL has given in the previously mentioned fields, in a first proof-of-concept we demonstrated that it is possible to apply DRL to learn mobile manipulation behaviours [6]. Based on the idea proposed in [2], we learnt a DRL-based positioning policy to drive the robot's base with speed commands to areas that guarantee the reach of the arm to the target object. Unlike the original work in which they learn a classifier that is used offline to predict target poses from the base, our goal was to learn an online control policy for the robot. This approach allows the dynamics of the environment to be taken into account, as decision-making is reactive to the state of the environment. The aim of this work was to demonstrate the feasibility of the approach in simulation.

Subsequently, works such as [38, 39] tried to learn how to jointly control both the base and the arm of a mobile manipulator with DRL. On the one hand, Kindle et al

use the Proximal Policy Optimisation (PPO2) [40] algorithm to perform the whole-body control of a holonomic mobile manipulator to solve the task of picking up an object from a shelf [38]. On the other hand, the approach proposed by Wang et al was similar to the previous one but applied to a non-holonomic robot. In this case, they also use PPO2 to learn the task of picking up an object on a table [39]. Both applications follow the simulation-to-reality approach, so they perform the training in simulation and then directly transfer the learnt behaviour to reality. Nevertheless, in both applications they limit the degrees of freedom of the arm to simplify the control problem, which can greatly limit the robot's reach.

In this paper, we extend our first proof-of-concept with the aim of improving the method to finally validate it in a real environment. Contrary to some state-of-the-art approaches that try to learn a whole-body reactive controller, we propose to learn a positioning policy for the base taking into account arm's range constraints. To that end, we propose to combine a traditional path planner for the arm with a DRL-based reactive controller for the base. In this way, we avoid limiting the degrees of freedom of the robotic arm, allowing for maximum flexibility in picking. We build on the idea that traditional planning and control methods such as the ones offered in MoveIt! provide the precision needed to manipulate objects. In addition, using a DRL-based reactive controller for the base allows us to take into account the dynamics of the environment during positioning. In fact, we believe that the feedback from the arm planner can help optimise the base controller to take into account arm's range constraints.

For the first time we use the state-of-the-art TD3 algorithm to model a mobile manipulation behaviour. Furthermore, we propose an improved setup composed of a reward function, a definition of state/action spaces, and a selection of neural architectures to learn a robust positioning policy for the base. Following a simulation-to-reality approach, we first train and validate the method in a realistic simulation of the real workshop. In the benchmark carried out in simulation between the baseline and our approach, we show both, that TD3 and the proposed setup outperform the algorithms and the setup proposed in the baseline work. Once the performance and reliability of the controller is assessed in simulation, the controller is deployed and validated in the real robot.

3 Methodological approach

The developed method focuses on the picking operation of a randomly placed item on a table, using a mobile manipulator. In particular, we concentrate on learning a reactive controller for the base of the mobile manipulator, which by sending velocity commands will drive the robot to areas where the reachability of the arm to the object is ensured. The key novelty of our method is that the velocity controller for the base is optimised considering the feedback of the arm's path planner. The rationale behind our approach is to encourage the base controller to position the robot's base in zones where the arm's planner will likely succeed planning a trajectory up to the target object. In fact, we propose to learn the base controller with real experience obtained through the interaction with the environment. By doing so (1) the robot learns its own physical limitations; (2) it takes into account the uncertainty in both the robot's location and the target object; and (3) the decision making is learnt automatically, thus avoiding the programming of such behaviour.

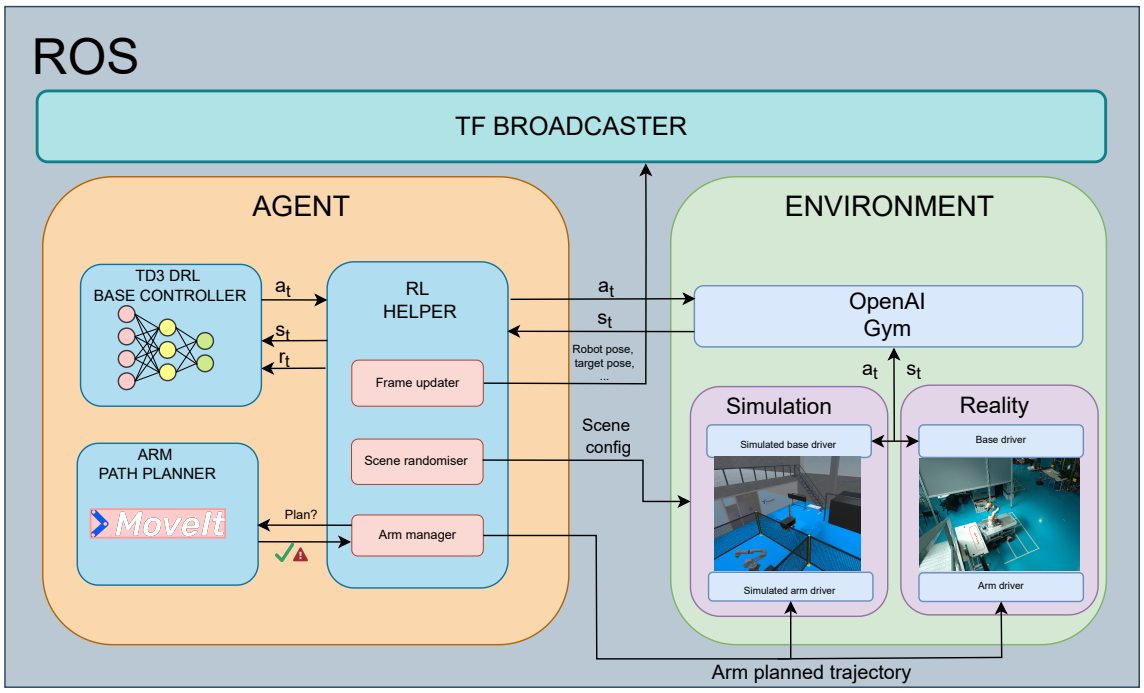


Fig. 2: Developed control framework. ROS nodes are represented with blue boxes. The main libraries used inside the ROS nodes are shown in red. Black arrows indicate ROS service calls.

The developed framework, depicted in Figure 2, is based on ROS[9] and all the modules of the system are implemented as ROS nodes. The control architecture has been designed with the possibility of being executed both in the simulated and the real environments. On the one hand, we call it agent to the set of software modules involved in the decision-making of the actions to be executed by the robot. On the other hand, we call it environment for all the modules that participate in the execution of the predicted actions, both in simulation and in reality.

As far as the agent is concerned, its main module is the mobile manipulator base controller which is modelled using the TD3 DRL algorithm. In our case, the aim is to train TD3 to be a reactive controller to drive the robot's base with velocity commands and considering the scope of the arm. More details about the agent are given in Section 3.2.

To model the environment, we follow the widely used OpenAI Gym interface that is used in most state-of-the-art DRL algorithms [10]. OpenAI Gym is a library designed to develop and compare DRL algorithms by providing a standard API to communicate between learning algorithms and environments. Indeed, this communication interface has become a standard to design DRL algorithms and environments and most of the libraries follow it. The use of this API eases the simulation-to-reality transfer of the learnt controller and makes the environment being used transparent to the agent. The details of the implementation of the environment can be found in Section 3.1.

3.1 Environment

As previously mentioned, the environment has been developed following the OpenAI Gym interface, which facilitates the integration with the agent. Furthermore, it makes

it transparent to the agent what type of environment the predicted actions are being executed in (simulated or reality), as both the input and output always have the same format.

The main objective of this work is to demonstrate the feasibility of the proposed method in a real environment. Nonetheless, due to the material and time costs involved in training DRL-based agents in a real environment, following the general approach, it is proposed to do so in a simulated environment. For this purpose, a simulation with a high degree of realism has been developed, which is detailed in Section 3.1.1. In addition to simulating the robot itself, we have also implemented the necessary drivers to control both the base and the arm of the mobile manipulator in the same way than the real ones.

In order to reduce the gap between simulation and reality and to learn an uncertainty-robust controller, the simulation offers services to randomise both the pose of the robot and the target object. The idea is to perform the training in the realistic simulation, introducing a certain degree of randomisation, so that the learnt controller can be used directly on the real robot.

3.1.1 Simulated environment in Unity

Unity is a real-time 3D development platform that consists of a rendering and physics engine, in addition to a graphical user interface [8]. Although this simulator has been used extensively for game development, it has also been used in many other areas such as the automotive, engineering, and film industries. Unity allows the development of environments rich in visual, physical, and task complexity, which is vital for robot-learning applications. It offers an easy integration and useful toolboxes to work with AI-based models.

The developed simulation has been integrated into the ROS environment using the ROS# library for the C# programming language [41] and following the OpenAI Gym interface. Figure 3 shows the main components of the simulated system, i.e. the mobile manipulator and the table integrated in a realistic environment. The robot is modelled following the URDF format, which is easily loaded to the simulated environment using ROS#.

In order to be able to control the mobile base with velocity commands, a low-level driver is developed taking the idea of the *gazebo planar move* plugin. This plugin implements a simple controller that receives as input a twist command and moves the simulated robot in the xy plane. The idea is to replicate the real driver of the base of the robot which allows it to be controlled by speed commands. Additionally, we introduce accelerations to make the simulation more realistic. The acceleration coefficient used has been experimentally measured in the real robot. In addition, the simulated base driver is in charge of publishing the localisation of the robot in the scene. It is widely known that real localisation systems are noisy, and therefore, the localisation is not perfect. However, the simulation tool used does not introduce localisation uncertainty. To somehow overcome this lack and make the system robust to imprecise localisation we add Gaussian noise to the robot pose given by the system at each time step. The modifications made in the localisation are detailed in Equation

1.

$$\begin{aligned} x_r^w &= x_r^w + \mathcal{N}(0.0, 0.02) \\ y_r^w &= y_r^w + \mathcal{N}(0.0, 0.02) \\ \alpha_r^w &= \alpha_r^w + \mathcal{N}(0.0, \frac{\pi}{100}) \end{aligned} \tag{1}$$

To control the simulated 7-DoF arm, instead, we follow the idea of the *gazebo ros control* [42] to implement a simulated driver under the *ROS control* framework [43]. This driver acts as a bridge between Unity and the joint trajectory controller offered in MoveIt! and executes the trajectories given by the joint trajectory controller in simulation.

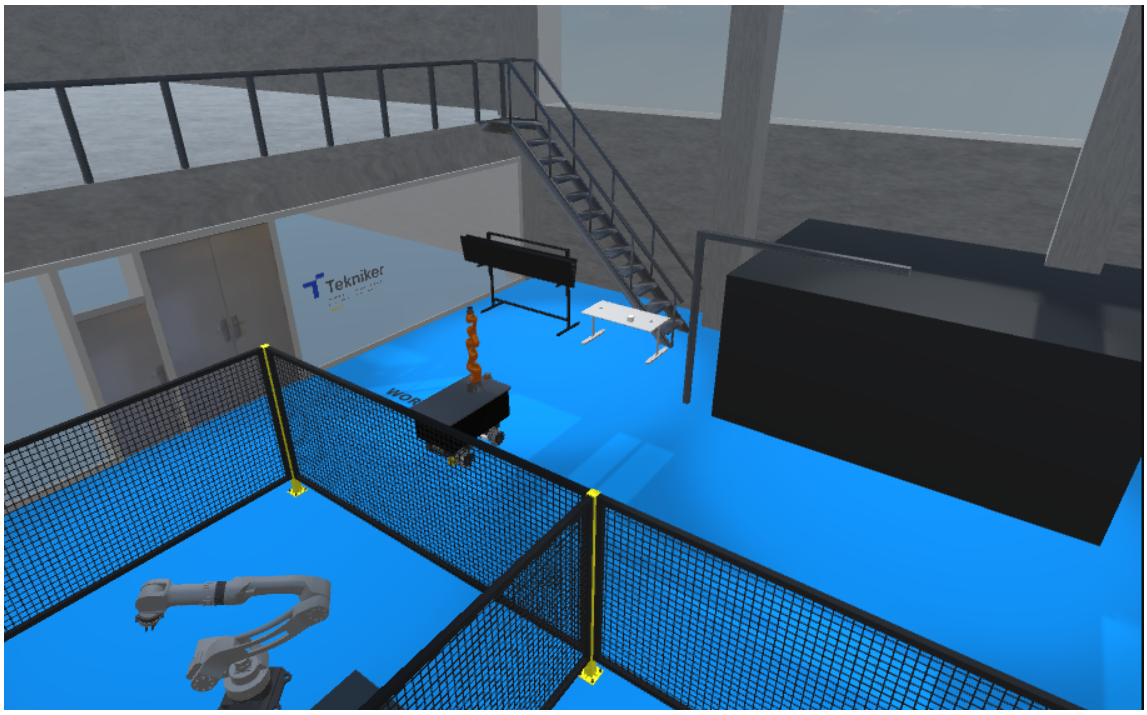


Fig. 3: Simulated environment in Unity.

3.2 Agent

According to Sutton and Barto [44], a RL solution to a control problem is defined as a finite-horizon Markov Decision Process (MDP). At each discrete time-step t the agent observes the current state of the environment $s_t \in S$, takes an action $a_t \in A(s_t)$, receives a reward $r : S \times A \rightarrow \mathbb{R}$ and observes the new state of the environment s_{t+1}. At each episode of T time-steps, the environment and the agent are reset to their initial poses. The goal of the agent is to find a policy, deterministic $\pi_\theta(s)$ or stochastic $\pi_\theta(a|s)$, parameterised by θ under which the expected reward is maximised.

In our case, we implement the **DRL base controller** using the TD3 algorithm. At each time step $t \in T$ the algorithm takes as input the state of the environment s_t and predicts the optimal velocity command a_t that leads to the expected maximum

reward at the end of the episode. In our case, the maximum reward is achieved when the robot positions itself in a zone where the target is within the reach of the arm.

Furthermore, we use two additional nodes that play a key role during learning. On the one hand, we have **Arm path planner**, which determines in each case whether the target is within the reach of the robot arm. As it is later explained in Section 3.2.1, the result of the arm planner is used to calculate the reward for the base controller. If the base controller has been able to drive the robot to an area where the arm is able to reach the object, it will receive the maximum possible reward. The idea is to encourage the base controller to drive the robot to areas with a high probability of grasping success. Arm's path planning is done using the MoveIt! library [13] which is, in fact, the default planning and control library for manipulation applications in ROS.

On the other hand, the **RL helper** node is in charge of orchestrating both the learning and the execution of the agent and acts as a bridge to communicate with the environment. The process runs as follows: In an episode of T time-steps, at time instant $t = 0$, it randomises both the pose of the robot and the target using the "scene randomiser" library. The goal is to introduce variability while training to make the controller robust to uncertainties. During the time instants $t = 1$ to $t = T - 1$, it receives the velocity commands a_t predicted by TD3 and executes them in the environment (simulated or real). As a consequence, it obtains the new state of the environment s_{t+1} and on that basis calculates the reward r_t. In addition, using the "frame updater" library, it obtains the state of the environment and spreads the transformations between coordinate systems (robot pose, target, etc.) in ROS using the TF broadcaster. The TF broadcaster node is built on top of the TF library of ROS [45], and allows to dynamically modify transformations between coordinate frames. Finally, at the last instant of time of the episode $t = T - 1$ the robot navigation is stopped. At this instant, the "arm manager" library previously developed and with a higher abstraction level than MoveIt!, calls the planner to see if the object is in the scope of the arm to reward or penalise the base controller accordingly.

3.2.1 TD3 as base controller

To model the base velocity controller, the TD3 algorithm has been chosen, which solves some major stability issues of its predecessor DDPG. Although DDPG has shown great performance learning some robotic skills, it usually tends to be brittle with respect to the tuning hyper-parameters. TD3 follows the actor-critic architecture and, similarly to its predecessor, learns a deterministic policy π_θ in an off-policy way, called actor. Both DDPG and TD3 are derived from the deterministic policy gradient theorem [46]. Based on the idea of a deep Q network (DQN) [47], TD3 also uses action-value functions Q_w to guide the learning process, also dubbed critic. Both the actor and the critic are parameterised functions and usually are implemented as non-linear function approximators. Similar to DDPG, TD3 is able to learn robust value functions due to two innovations: First, the networks are trained off-policy, getting experience samples from a replay buffer to eliminate temporal correlations. In addition, target networks, π' and Q' are used for both the actor and critic respectively, which are updated slower leading to consistent targets y during temporal difference (TD) learning [48].

Figure 4 depicts the network architecture of TD3. TD3 introduces three main novelties: First, it uses a second critic network to improve the stability of the learning

process [7]. Thus, the actor is a parameterised policy $\pi_\theta(s) = a$, and critics are action-value functions, $Q_{w_1}(s, a)$ and $Q_{w_2}(s, a)$, which evaluate the quality of the execution of action a in state s. Second, the actor is trained slower than the critics. These less frequent policy updates result in a lower variance action-value estimate that leads to a more robust policy. And third, it adds noise to the action predicted by the target policy, to make it harder to the policy to exploit Q-function errors.

In short, TD3 is made up of one actor and two critic networks, plus the target network of each of them, resulting in 6 neural networks.

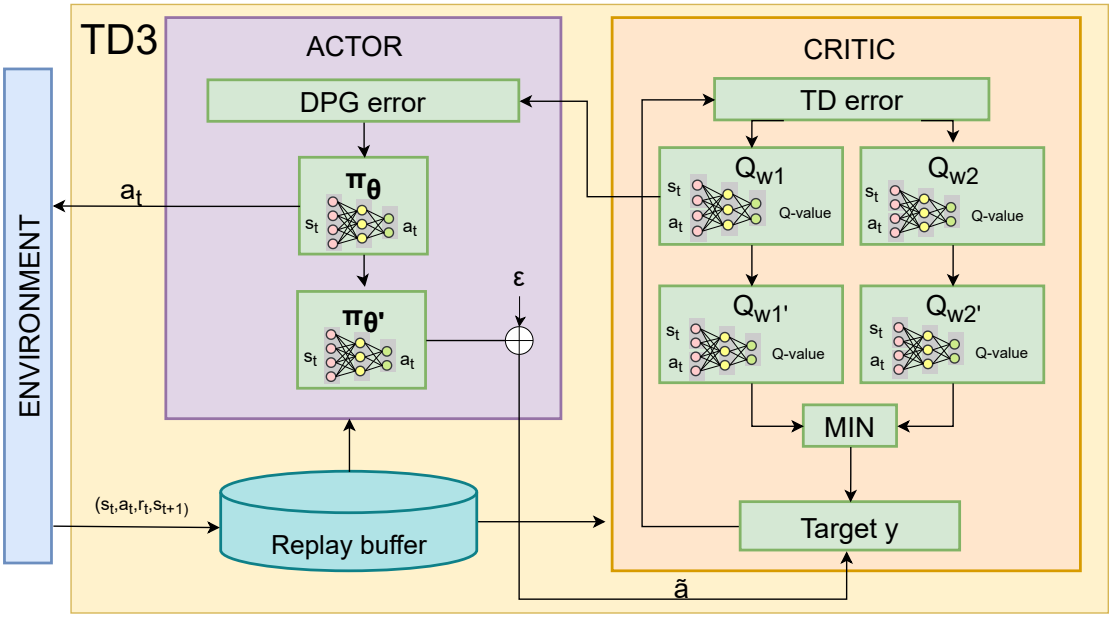


Fig. 4: Network architecture of TD3.

To train the networks, in each training iteration, a mini-batch of N transitions (s_t, a_t, r_t, s_{t+1}) is sampled from the replay buffer. On the on hand, to update the weights of the critics w the TD error is used [48], which can be seen in Equation 2. On the other hand, the actor is updated using the deterministic policy gradient (DPG) theorem [46] which is shown in Equation 3. However, as previously mentioned the actor is updated slower, usually once per two training steps.

$$\begin{aligned} \tilde{a} &\leftarrow \pi_{\theta'}(s) + \epsilon, \qquad \epsilon \approx clip(\mathcal{N}(0, \tilde{\sigma}), -c, c) \\ y &\leftarrow r + \gamma min_{i=1,2} Q'_{w'_i}(s_{t+1}, \tilde{a}) \\ w_i &\leftarrow min_{w_i} \frac{1}{N} \sum (y - Q_{w_i}(s, a))^2 \end{aligned} \tag{2}$$

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum \nabla_a Q_{w_1}(s, a)|_{a=\pi_\theta(s)} \nabla_\theta \pi_\theta(s) \tag{3}$$

Concerning the target networks, those are a weighted copy of the original networks and are usually updated at the same frequency than the policy as shown in Equation 4.

$$\begin{aligned} w'_i &\leftarrow \tau w_i + (1 + \tau) w'_i \\ \theta'_i &\leftarrow \tau \theta_i + (1 + \tau) \theta'_i \end{aligned} \tag{4}$$

For this research, the implementation of the *stable baselines* library is used [49], which already uses the OpenAI Gym interface.

3.2.2 State/action spaces and neural architectures

In the proposed setup, we model both the state and action as continuous spaces as described in Equations 5 and 6 respectively. The state observation $s \in \mathbb{R}^{13}$ is composed of the variables defined in Table 1. We modify the baseline state space by adding the observed robot's velocity. It must be noted that in the baseline setup, the predicted action in the previous time step a_{t-1} and the observed velocity v' are the same, but this does not happen in our setup as accelerations are considered. This modification lets the agent be aware of its current velocity in order to decide the velocity command for the next time-step.

1. The pose of the base of the arm with respect to the world that is composed of the x and y coordinates and α rotation, where α is the rotation in z axis	$p_r^w = [x_r^w, y_r^w, \alpha_r^w]$
2. The predicted action in the previous time-step	$a_{t-1} = [vx_{t-1}, vy_{t-1}, \omega z_{t-1}]$
3. The observed linear and angular velocities	$v' = [vx', vy', \omega z']$
4. The position of the target with respect to the world	$p_{target}^w = [x_{target}^w, y_{target}^w]$
5. The distance between the base of the arm and the target	d_r^{target}
6. The normalised time-step in the episode	$\frac{t}{T}$

Table 1: Variables that define the environment state.

The action $a \in \mathbb{R}^3$, instead, is composed of the target linear and angular velocities to be sent to the base of the holonomic robot.

$$s = [p_r^w, \quad a_{t-1}, \quad v', \quad p_{target}^w, \quad d_r^{target}, \quad \frac{t}{T}] \in \mathbb{R}^{13} \tag{5}$$

$$a = [vx, \quad vy, \quad \omega z] \in \mathbb{R}^3 \tag{6}$$

Concerning the implementation of both the actor and the critics, both of them have been implemented as Multi Layer Perceptron (MLP) feed-forward networks [50].

On the one hand, the MLP of the policy $\pi(s) = a$ gets as input the state observation $s \in \mathbb{R}^{13}$ in the input layer and it is followed by two hidden layers, with 400 and 300 neurons respectively. Finally, the output layer predicts the action $a \in \mathbb{R}^3$. The $ReLU$ activation function is used for the hidden layers, and *Tanh*, instead, for the output layer to bind the actions between -1 and 1. The variables that compose the state observation s are normalised before feeding the policy MLP using the observation space limits shown in Table 2. The opposite happens with the predicted action that originally is in the [-1,1] range and is converted to the action space range (also shown in Table 2).

On the other hand, the critics $Q(s, a)$ get the $[s, a] \in \mathbb{R}^{16}$ feature vector as input. Then, the input layer is also followed by two hidden layers with 400 and 300 neurons respectively and, finally, the output layer predicts the $\mathbb{R}^1$ Q-value for the state-action pair. The $ReLU$ activation function is used for the hidden layers, and for the output layer, however, no activation function is used. In this case, the inputs s and a are also normalised using their respective limits before feeding the critic MLPs.

Observation space limits			
x_r^w	$[-2.0, 5.6]\ m$	x_{target}^w	$[4.1, 4.2]\ m$
y_r^w	$[-2.8, 2.8]\ m$	y_{target}^w	$[-0.7, 0.4]\ m$
α_r^w	$[-\frac{\pi}{2}, \frac{\pi}{2}]\ rad$	d_r^{target}	$[0 - 6.26]\ m$
$vx', vy', vx_{t-1}, vy_{t-1}$	$[-0.5, 0.5]\ m/s$	$\frac{t}{T}$	$[0 - 1]$
$\omega z', \omega z_{t-1}$	$[-0.5, 0.5]\ rad/s$		
Action space limits			
vx, vy	$[-0.5, 0.5]\ m/s$	ωz	$[-0.5, 0.5]\ rad/s$

Table 2: Observation and action space limits.

3.2.3 Reward function

The design of the reward function is one of the most important steps during the modelling of a DRL-based agent, as it is the only signal that guides the learning process. As explained above in Section 3.2.1, the rewards are used to optimise the critic networks of TD3, the output of which is then used to optimise the actor. The most logical thing to do would be to reward the robot only when it achieves the goal, i.e. to position itself in such a way that the target object is within the arm's reach. Nevertheless, this usually does not work and it is necessary to reward the agent for achieving sub-goals in order to guide the learning process until an optimal one is achieved. This concept is known as reward shaping [51]. In our case, in addition to rewarding the agent when the main goal is met, we use other criteria such as distance to the goal or speed/acceleration to "shape" the reward function.

The designed reward function is detailed in Equation 7 and its main components are the following:

- Distance reward, D: The closer the robot is from the target, the higher is the distance reward. When it gets closer than d_{thresh}, the distance reward becomes constant to avoid crashing with the table. This component is described in Equation 8.
- Velocity penalty, V: The agent is penalised with a discount factor for moving with high velocities, particularly when the robot is closer than d_{thresh} from the goal (see Equation 9). This discount factor is applied to D.
- Collision penalty, C: If there is a collision between the robot and any element in the environment, this variable takes the value 1. Otherwise, its value is 0.
- Acceleration penalty, W: The agent is penalised for predicting consecutive actions that would require high acceleration (Equation 10). To this end, the L^2 norm of the difference between a_t and a_{t-1} is used.
- Grasp reward, G: The robot tries to plan a trajectory up to the target in the last time-step of the epoch $t = T$. If planning succeeds, this variable takes the value 1 and, instead, takes the value -1. In the other time-steps of the episode its value is 0.

The weights w_d, w_c, w_w, w_g are used to determine the importance of each component of the reward function.

$$r = w_d \cdot D \cdot V \cdot (1 - C) - w_c \cdot C - w_w \cdot W + w_g \cdot G \tag{7}$$

$$D = \begin{cases} c_1, & \text{if } d_r^{target} < d_{thresh} \\ \frac{1}{d_r^{target}}, & \text{if } d_r^{target} \geq d_{thresh} \end{cases} \tag{8}$$

$$V = \begin{cases} (c_2 - min(\|v'\|_2, c_3))^{\beta}, & \text{if } d_r^{target} < d_{thresh} \\ (c_4 - max(\|v'\|_2, c_5))^{\beta}, & \text{if } d_r^{target} \geq d_{thresh} \end{cases} \tag{9}$$

$$\text{where, } \beta = \frac{c_6}{max(d_r^{target}, c_7)}$$

$$W = \|a_t - a_{t-1}\|_2 \tag{10}$$

Both the weights of each component and the constant values have been obtained experimentally and are shown in Table 3.

Reward function weights and constant values			
w_d	1.0	w_c	1000.0
w_w	10.0	w_g	1000.0
c_1	100.0	c_2	1.0
c_3	1.0	c_4	2.0
c_5	0.5	c_6	2.0
c_7	0.65	d_{thresh}	0.8

Table 3: Weights and constant values used for the reward function.

The most critical novelties introduced with respect to the reward function used in [6] are related with the distance reward and the velocity penalty.

On the one hand, in the baseline reward function the robot is rewarded for getting as close as possible to the target. However, this introduces a potential risk of collision with the table and, in addition, this could prevent the arm from successfully planning a trajectory to the target for being too close to it.

On the other hand, the baseline simulated low-level driver that is in charge of receiving velocity commands from the DRL agent and moving the robot does not consider accelerations. Because of this the robot is able to stop immediately from one instant of time to another, regardless of its speed. Nonetheless, this makes the baseline simulation to be far from reality and the learnt behaviour in simulation is hardly applicable in the real system. In fact, in the reward function proposed in the baseline setup, the robot is not penalised for moving fast near the goal, and this causes the robot to aggressively approach the target when accelerations are considered, which leads to collisions with the table.

The simulation developed in this work it does consider base accelerations and, thus, the aforementioned issues need to be solved. To that end, on the one hand, we modify the distance reward by giving a constant reward to the robot when the target is within the robot arm's reach. This helps to reduce the importance of the distance at this point and gives more importance the robot to reach to the last the time-step of the episode to perform the grasp. On the other hand, we introduce a velocity penalty to penalise the robot for moving fast near the goal. The absence of the velocity discount factor produces aggressive approximation to the table at high speed and thus, the probability of collision increases. The main goal of this discount factor is to give higher rewards for moving slowly near the target. Moreover, the smoother approximation to the table leads to a better exploration behaviour, and the robot becomes able to wait for the last time-step of the episode in a correct grasping place.

3.3 Training procedure

The training of the agent is done following an episodic scheme. Each training episode consists of $T = 512$ discrete time-steps, with a time-step duration of $T_s = 30ms$. The duration of each episode is around 15s. At each discrete time-step of the episode $t \in T$, the agent observes the environment state $s_t \in S$ (see Equation 5), predicts an action $a_t \in A(s_t)$ (see Equation 6), receives a reward signal $r_t : S \times A \rightarrow \mathbb{R}$ (see Equation 7) and observes the new state of the environment s_{t+1}. As detailed in Section 3.2.1, the transition (s_t, a_t, r_t, s_{t+1}) is finally stored in the replay buffer. At the beginning of each episode, both for training and validation, the initial poses of the robot and the target are randomly selected. The limits of the observation and action spaces used are constant during all the training episodes and are shown in Table 2. The detailed training algorithm of the agent can be seen in Algorithm 1 of Appendix A.

In this way, at each episode the robot has T time-steps to complete the positioning of the robot's base in a suitable zone for the arm to plan a trajectory up to the target. To that end, at each time-step the base controller observes the environment state, predicts the optimal velocity command, and after executing the action, it receives a reward signal. At time-step $t = T - 1$ the path planning trial is done with the arm and the robot is rewarded or penalised depending on whether the planning succeeds or not.

The localisation of the robot is published at $\approx 100Hz$ and the robot control is done at $\approx 33Hz$. The training is done using an Intel Core i7-8700 CPU(@ 3.20 GHz x 12) with a Nvidia Geforce GTX 1060 GPU and each training is executed during 4M steps in simulation. Whenever the agent reaches a terminal state, the simulation is reset, and a new episode starts. Three terminal states are considered:

1. A collision occurs between the robot and any other element in the scene.
2. The localisation of the robot falls outside the limits defined in the observation space.
3. The robot reaches the last time-step of the episode $t = T - 1$.

The main hyper-parameters used to train the TD3 agent are shown in Table 4.

Learning rate (α)	$1e-3$	Gradient steps	100
Target network update rate (τ)	$5e-3$	Batch size (N)	128
Discount factor (γ)	0.99	Policy delay (d)	2
Replay buffer size	$1e6$	Action noise (ϵ)	$\mathcal{N}(0, 0.2)$
Learning starts	$1e4$	Target noise clip (c)	0.5
Training freq	100	Target policy noise (ϵ')	$\mathcal{N}(0, 0.2)$

Table 4: Training hyper-parameter for the TD3 algorithm.

4 Benchmark in simulation

The main objective of the experimentation is to evaluate the method we propose in comparison with the baseline [6]. First, we want to demonstrate that the TD3 algorithm fits better to the problem at hand and allows learning a more robust behaviour compared to DDPG and PPO2. And second, we want to show that the setup (state/action spaces, the reward function and neural architectures) we propose

in this work leads to better performing behaviour, regardless of the DRL algorithm used.

To do this, in addition to TD3, we use the DDPG and PPO2 algorithms proposed in the baseline work to model the base controller. We train each algorithm with both the baseline setup and the one proposed in this paper in the realistic simulation detailed in Section 3.1.1. In this way, 6 training runs are carried out in total, one per DRL algorithm/setup combination, and as a result 6 trained agents are obtained.

During the benchmark, the performance and reliability of the agents are quantitatively measured using standard metrics. On the one hand, the comparison during training gives us an idea of the convergence of each algorithm with each setup, as well as the reliability shown during learning. On the other hand, the comparison during the evaluation (using trained agents) shows the real performance and reliability of the agents. The idea is, based on the latter, to select the best algorithm/setup combination and then deploy and validate it on the real robot.

The state/action spaces, reward function, and neural architectures used to train the TD3-based agent are reported in Section 3.2.1. In addition the hyperparameters used to tune TD3 are detailed in Section 3.3. The baseline setup and the hyperparameters used both for DDPG and PPO2 are those reported in [6].

The algorithm used to evaluate the agents during both training and evaluation can be found in Algorithm 2 of the Appendix A.

4.1 Training runs

While the agents are training, a 5-episode evaluation period is performed at all $1e4$ training steps to assess the quality of the agent learnt at that time. The training procedure followed is detailed in Section 3.3. In each evaluation episode, first both the initial poses of the robot and the target are randomly assigned, and then the agent is executed to carry out a picking trial. Taking into account these evaluation periods, the performance and reliability of each algorithm/setup combination is analysed. On the one hand, performance is used to see how fast each algorithm converges based on the chosen setup. On the other hand, the reliability of the model shows how stable the learning process is and what risk there is in the short and long term of a large drop in performance. The metrics used for this are defined in Section 4.1.1.

4.1.1 Metrics

To measure the performance of the algorithm/setup combinations while training the following metrics are used:

- Average accumulated rewards: In each evaluation episode the accumulated reward (i.e. the sum of the rewards during the T time-steps of the episode) is computed. Finally, the average accumulated reward is calculated among the 5 evaluation episodes of each evaluation period.
- Success rates: Per each evaluation period of 5 episodes, a success rate is computed which tells us how many times the learnt base controller has been able to drive the mobile manipulator's base to areas where the target is in the scope of the arm. For illustration purposes we show the average success rates considering 5 evaluation periods.

To quantitatively measure and compare the reliability of the algorithms during training, we use the metrics proposed in [52], that focus on measuring the risk and the dispersion of DRL algorithms:

- Dispersion across Time (DT): Measures the dispersion of the average accumulated rewards during training. The dispersion is measured using the inter-quartile range (IQR) [53], in this case the distance between the $75th$ and $25th$ percentiles. This metric measures the short-term variability that corresponds to a noisy training.
- Short-term Risk across Time (SRT): The goal of this metric is to measure the most extreme reward drop from one evaluation step to the next. To that end, the conditional value at risk (CVaR) or the expected shortfall [54] of the differences in the rewards between successive evaluations is measured. CVaR measures the expected value below the α-quantile in the distribution formed by the reward differences. In our experiments we used the default value of $\alpha = 0.05$.
- Long-term Risk across Time (LRT): This metric measures long-term risk during each training run by measuring the most extreme drop in reward with respect to the peak, also called *drawdown* [55]. In this case, CVaR is applied to *drawdown*.

Because each setup has a different reward range, to compute the reliability metrics across the setups, those are converted into rankings as proposed in [52]. In fact, the rankings are first calculated per setup, and finally, the across-setup mean rankings are obtained.

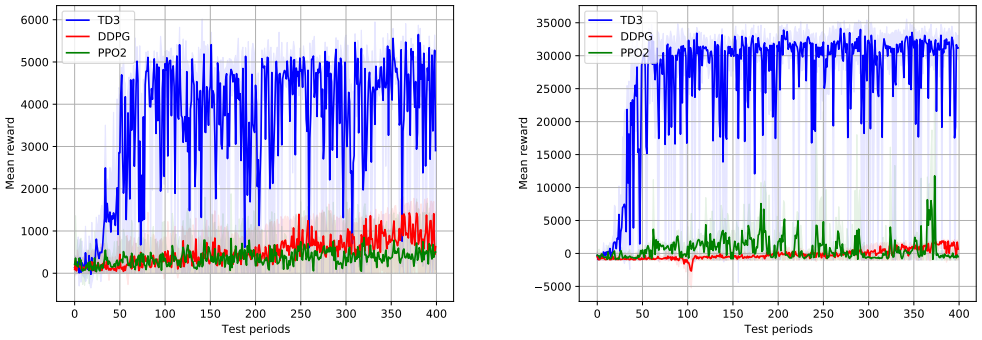
4.1.2 Results

The average accumulated rewards and the success rates obtained during training are depicted in Figure 5, both for the baseline and the proposed setups.

As can be observed in Figures 5a and 5b, in both the baseline and the proposed setups, TD3 outperforms DDPG and PPO2 algorithms by far. The average accumulated rewards obtained by TD3 are much higher than the rest, indicating that from the beginning of learning, it has been able to learn a considerably more robust behaviour. Although looking at the rewards we can see the superior behaviour of TD3, it is not easy to know at which points the robot has been able to successfully complete the task. This is due to reward shaping, as in addition to rewarding the robot for completing the task, it also rewards for a correct approach. Thus, Figures 5c and 5d show the average success rates obtained during the training process. The success rates confirm the clearly superior performance of TD3 during training.

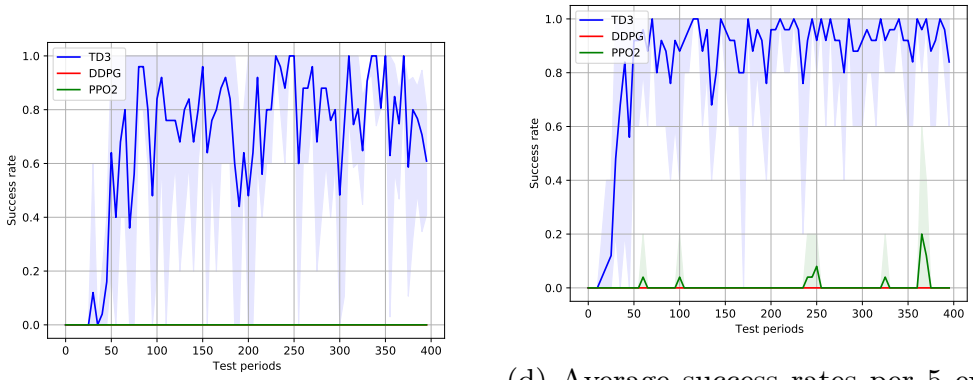
Looking at the rewards it is difficult to assess in which setup TD3 achieves better behaviour, as the reward scales are different. However, in Figures 5c and 5d it can be clearly seen that with the proposed setup the success rates are higher and more stable during the learning process. Additionally, the combination of TD3 and the proposed setup leads to a faster convergence. While the baseline setup achieves a success rate higher than 80% around the 75th evaluation period, the proposed setup achieves it around the 40th evaluation period.

The dispersion and risk metrics rankings are shown in Figure 6. The DT metric indicates that the algorithm with the highest dispersion across time is TD3, which can be clearly seen in the average accumulated rewards of Figures 5a and 5b. Nonetheless, the SRT and LRT metrics indicate that it is the algorithm with the lowest short-term and long-term risk. Despite the less noisy average reward curves for DDPG and PPO2 shown in Figures 5a and 5b, success rates in Figures 5c and 5d indicate failure in solving the task. In contrast, the success rates for TD3 in the proposed



(a) Average accumulated rewards per each evaluation period, using the baseline setup.

(b) Average accumulated rewards per each evaluation period, using the proposed setup.



(c) Average success rates per 5 evaluation periods, using the baseline setup.

(d) Average success rates per 5 evaluation periods, using the proposed setup.

Fig. 5: Training average rewards and success rates with PPO2, DDPG and TD3 algorithms with the baseline and the proposed setups.

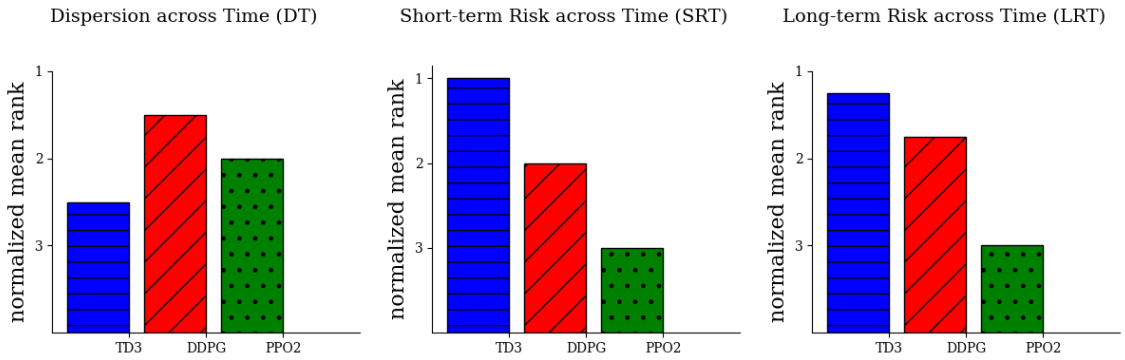


Fig. 6: Mean across-setup reliability rankings of TD3, DDPG and PPO2 algorithms considering the average reward training curves. Rank 1 always means the best reliability.

setup suggest a more stable learning curve, with an average success rate above 80% for almost all of the training process.

4.2 Evaluation runs

Although the training performance and reliability metrics give us an intuition of how good each algorithm/setup pair work while the policy is being trained, the final evaluation must be done with fixed policies. Evaluation with fully trained models will give a clear view of the real performance and reliability of each agent. Therefore, once the training runs are completed, for each algorithm/setup pair, we select the model with the highest average accumulated reward for the evaluation. To evaluate them, we run 100 episodes of T time-steps with each fixed model. In each evaluation episode the initial pose of both the robot and the target box are randomly selected before executing the picking trial. Note that the randomly selected poses are common in all the tests, the results to be comparable. This random selection is done using the limits of the observation space (detailed in Table 2).

4.2.1 Metrics

During the evaluation of the trained models, the metrics used to measure the performance where slightly different comparing to those used during the training runs:

- Accumulated median rewards: Since a single evaluation period of 100 episodes is carried out, we just measure the median of accumulated rewards across all the runs.
- Success rate: This metric tells us how many times the learnt base controller has been able to position the mobile manipulator's base in areas where the target is in the scope of the arm. A single success rate is calculated considering the 100 trials.

Similarly as in the training runs, we measure the reliability of the algorithms during the evaluation using the following metrics, also proposed in [52]:

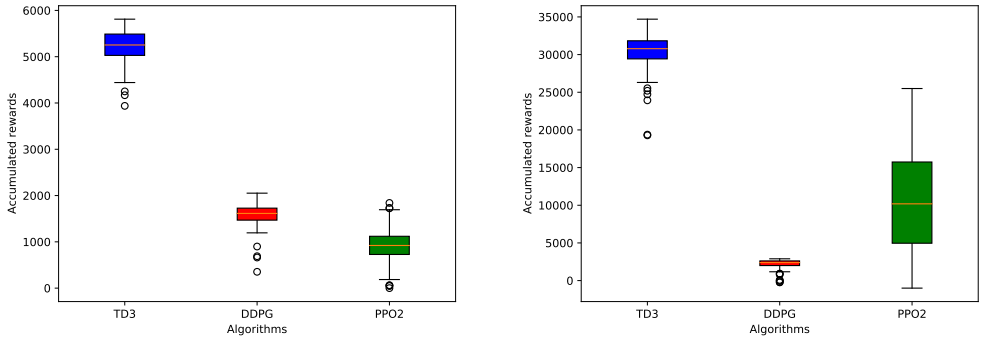
- Dispersion across Fixed-Policy Rollouts (DF): This metric measures the variability in performance when the same policy is run multiple times. This variability is measured using the IQR on the performance of the evaluation runs.
- Risk across Fixed-Policy Rollouts (RF): This metric measures the worst-case scenarios across the evaluation runs under a fixed policy. To that end, CVaR is applied to the performances in the evaluation runs.

4.2.2 Results

As expected, the TD3 algorithm outperforms DDPG and PPO2 and obtains a higher accumulated median reward in both the baseline and the proposed setups (see Figure 7). Similar to training, due to the reward shaping, only by looking at the accumulated rewards we cannot know with which of the two setups the TD3 algorithm performs better. Therefore, considering the 100 evaluation runs, we also measure the task success rates shown in Table 5. In contrast to training, where success rates are calculated throughout the learning process, in this case, the evaluation is performed on fixed models that have already been trained, so we obtain a single success rate per training.

The highest success rate is achieved by the combination of TD3 and the proposed setup, and surprisingly, DDPG fails to succeed in both cases. These results show how sensitive DRL-based agents are both to the setup and to the environment in which the agents have been trained.

The reliability metric rankings obtained from the evaluation are shown in Figure 8. The DF metric indicates that the algorithm with the lowest variability in performance is DDPG. However, its median performance leaves much to be desired in both setups, and the success rates indicate failure in solving the task. Furthermore, according to the RF metric, the algorithm with the lowest risk is TD3, similar to the training runs.



(a) Accumulated median rewards during the 100 evaluation periods with the baseline setup.

(b) Accumulated median rewards during the 100 evaluation periods with the proposed setup.

Fig. 7: Accumulated median rewards during the 100 evaluation periods with TD3, DDPG and PPO2.

Success rates (%)	TD3	DDPG	PPO2
baseline	86	0	0
proposed	**97**	0	50

Table 5: Success rates for each algorithm/setup combination during the 100 evaluation periods.

On the one hand, considering the performance and reliability metrics it can be said that TD3 is the best algorithm out of the 3 candidates to learn a positioning policy for the mobile manipulator's base that drives it to successful picking areas. On the other hand, the success rates show that the proposed setup enables TD3 to better learn the mobile manipulation task.

As the success rate indicates how many times the mobile manipulator has positioned itself in zones that enable a successful path planning for the arm up to the target, we also assess the quality of the trajectories generated by each controller. To that end, only focusing on the TD3-based agents, we measure the average distance travelled, considering the 100 evaluation runs. The travelled distance is measured

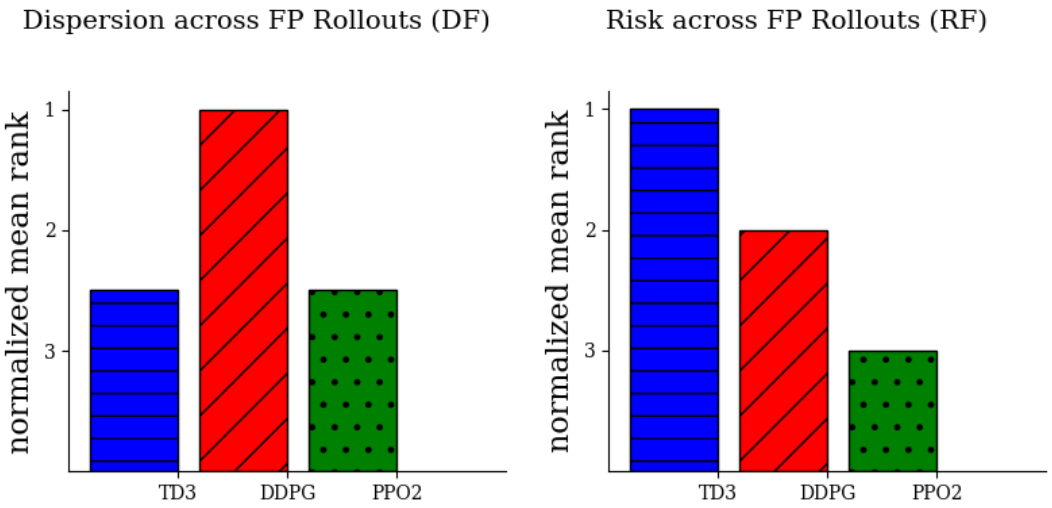


Fig. 8: Mean across-setup reliability rankings of TD3, DDPG and PPO2 algorithms considering the accumulated rewards during the 100 evaluation runs. Rank 1 always means the best reliability.

taking into account the 3-DoF of the omnidirectional base during the positioning operation. Considering that both the set of initial and target poses are common across tests, a shorter average travelled distance indicates higher quality navigation. Since there are no obstacles between the initial pose and the table on which the target is located, the shortest path is always the best. As can be seen in Table 6, the combination of TD3 with the proposed setup leads to 8.3% shorter trajectories on average.

Travelled distance (m)	$\tilde{d}$	$\overline{d}$	σ
baseline	5.787	6.084	1.543
proposed	4.753	5.573	2.655

Table 6: Median, mean and standard deviation of the distances travelled by the base of the mobile manipulator during the 100 evaluation periods with the TD3 algorithm.

5 Validation in the real system

Details about the deployment and validation of the learnt controller in the real robotic environment are explained in this section.

5.1 Considerations before the deployment

5.1.1 Localisation of the base

One of the most important issues is related to the localisation of the robot. In the real robotic system, the pose of the robot is provided by the odometry, which is known to contain a cumulative error. Thus, we use a 2D localisation system with the aim of correcting the robot's pose given by the odometry using a map of the environment.

To that end, the Adaptive Monte Carlo localisation (AMCL) algorithm is used as the 2D localisation algorithm for the robot [56]. This probabilistic algorithm represents the localisation of the robot on a map as a particle filter. In fact, it fuses multiple data sources such as the odometry or laser scans to estimate the position of the robot in the map. We created the map of the real workshop using the *Gmapping* [57] algorithm (Figure 9). Nevertheless, one of the main limitations of AMCL is that the corrected odometry pose is estimated at a very low frequency $(3 - 7Hz)$ due to the computational load of the sensor fusion process.

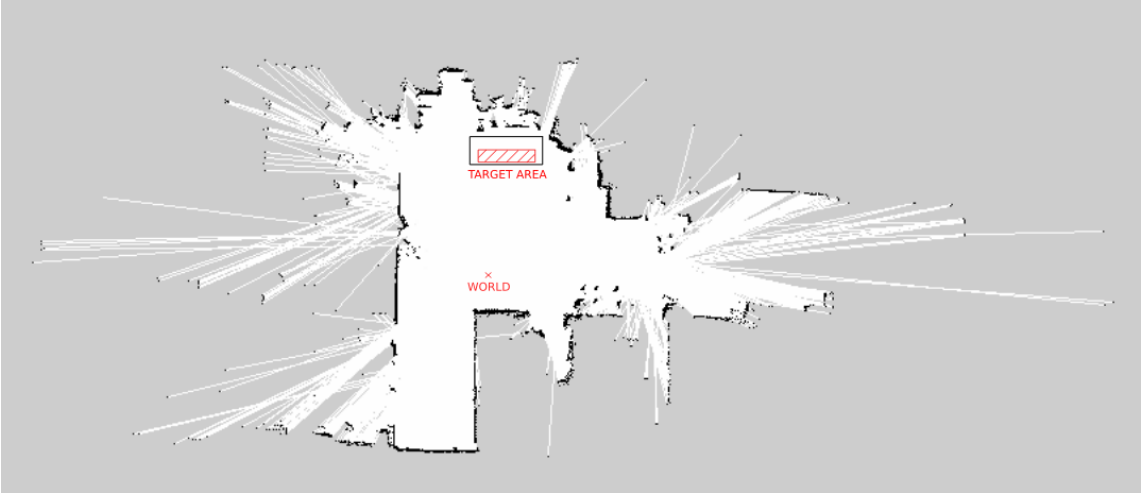


Fig. 9: Map of the real scenario.

5.1.2 Arm's path planning to the target

As in the simulation, the path planning for the real mobile manipulator's arm is performed using MoveIt!. The drift between the real and the estimated robot's pose provided by AMCL, however, causes the estimation of the relative pose between the base of the arm and the target not to be accurate enough, and consequently, the planned path can be invalid for manipulating the item. Indeed, an on-board vision system would be needed to correct the localisation error and successfully pick the part.

The main objective of this work is to demonstrate that the DRL-based agent is able to position the mobile manipulator in areas where the target object is within the range of the arm. Therefore, in addition to using the base location given by AMLC to perform the arm planning, we also use the real location calculated manually for verification purposes. The aim of doing this is to remove the localisation error from the system to see if the object is really within range.

5.2 Validation procedure

We select the TD3-based agent which has been trained with the proposed setup to carry out the evaluation in the real robotic system (see Figure 10). The evaluation consists of 20 trials where the robot's initial pose is randomly selected in each trial, and the position of the target, instead, is randomly selected once per 5 trials. The tests are carried out as follows:

1. The robot is set in a randomly selected initial pose and the DRL-based base controller is executed for T time-steps.
2. After T time-steps of navigation, once the robot has been positioned, at time-step $t = T - 1$ the arm tries to plan a trajectory up to the target. A successful plan indicates that the target object is within the reach of the robot. This first planning is done using the AMCL localisation. Finally, a success rate is computed considering the results of the plans.
3. The localisation uncertainty makes it impossible to use the robot's pose to deduce whether the target is reachable by the robot arm. Therefore, with the aim of removing the localisation error, the real localisation of the robot is manually measured. This lets us check whether the target is really reachable by the arm according to the arm's planner, and to use it as ground truth.

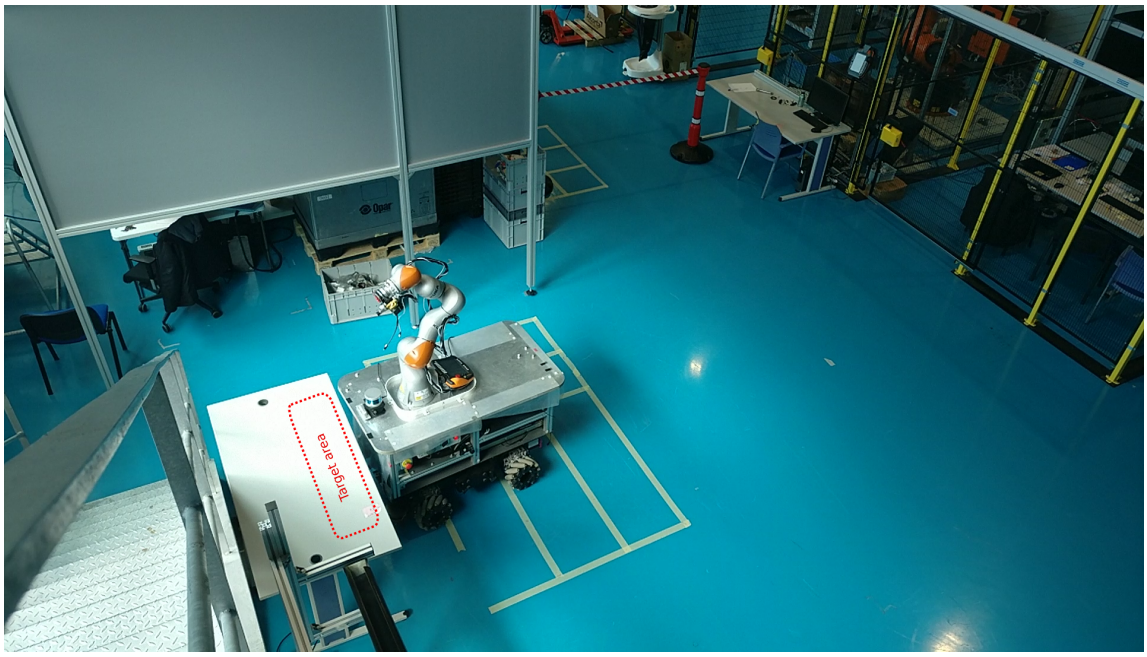


Fig. 10: Real scenario.

During each test, we take the following measurements:

- Difference between the observed and the measured real distance between the base of the arm and the target (Δd).
- Difference between the observed and real rotation on the z axis of the base of the arm with respect to the target ($\Delta\alpha$).
- Whether the base of the robot collides with the table when approaching the target (*coll.*).
- The attempt is considered successful if (1) there have been no collisions during the positioning and (2), the robot has been able to position itself in such a way that the arm is able to plan a trajectory to the object. The planning is done using both the localisation given by AMCL (*amcl_success*) and the real measured localisation (*real_success*).

5.3 Results

The measurements taken in the real validation environment are shown in Table 7. According to *amcl_success*, it can be seen that in 75% of the trials the robot is able to position itself in such a way that enables a successful planning of the arm up to the target, considering the localisation given by AMCL.

Trial	Δd (m)	$\Delta\alpha$ (rad)	*coll.*	*amcl_success*	*real_success*
1	0.051	0.088	False	True	True
2	0.092	0.060	True	False	False
3	0.021	0.091	False	True	True
4	0.048	0.072	True	False	False
5	0.187	0.178	True	False	False
6	0.018	0.082	False	True	True
7	0.047	0.065	False	True	True
8	0.003	0.125	False	True	True
9	0.032	0.260	False	True	True
10	0.083	0.059	False	True	True
11	0.034	0.061	False	True	True
12	0.056	0.057	False	True	True
13	0.037	0.047	False	True	True
14	0.019	0.040	False	True	True
15	0.050	0.066	True	False	False
16	0.040	0.081	False	True	True
17	0.088	0.062	False	True	True
18	0.052	0.090	False	True	True
19	0.087	0.097	True	False	False
20	0.053	0.078	False	True	True
Avg.	0.055 ± 0.039	0.084 ± 0.05	25%	75%	75%

Table 7: Measurements taken in the real environment.

However, if we focus on the mean error between the estimated and manually measured robot-target distances and rotations at time-step $t = T-1$ (Δd and $\Delta\alpha$), it can be seen that in average there is an error of $0.055m$ and $0.084rad$ respectively. At a glance, the average error magnitude seems important enough to cause unsuccessful manipulations. Note that the real execution of the manipulation plan is out of the scope at this phase because we want to see the performance of the base moving strategy itself. In fact, the main goal is to see if the DRL controller guarantees the reachability of the arm to the object.

Therefore, to eliminate the localisation error, in each test, a new plan is performed with the arm, considering the manually measured real localisation of the robot (*real_success*). In that case, the success rate is also 75%, which indicates that whenever the robot succeeds the planning using the AMCL localisation, it also succeeds with the real localisation. This indicates that in all cases the object is really within arm's reach. Consequently, we can conclude that although there is indeed an error in localisation, it is not large enough to cause the object to move out of the arm's reach.

The main decrease in performance is caused by small brushes with the table. The main causes for those brushes are the following:
Noisy localisation estimation: The localisation of the robot has a huge importance in the state representation of the base controller. Indeed, the reactive base controller makes its decision at each time-step based on the observation of the state. Although AMCL uses multiple data sources to correct the odometry, it still introduces an error in the localisation. This has a big effect, particularly when the robot navigates near the table. In addition, the low refresh rate of AMCL causes the localisation not to be updated in some iterations of the control loop and, thus, the consequence of each action of the robot is reflected with delay in the updated localisation. This delay in the localisation could cause the agent to make sub-optimal decisions at some critical time-steps of the positioning.
Learning to stop: Although the learnt controller successfully stops the robot when it is already positioned close to the target in simulation, this is not the case in reality. Indeed, the controller learns to send near-0 velocity commands to stop the robot in simulation, but the real robot struggles to stop and oscillates. This happens due to the noisy localisation estimates, which make the controller believe that the robot is slightly moving when it is actually still. Therefore, the controller tries to correct this error sending opposite velocity commands and causes the robot to oscillate. This effect is aggravated by the sim-to-real gap. Although we add noise to the localisation to simulate this effect while training the agent, the robot fails to properly stop, and this oscillation near the goal sometimes causes the robot to brush the table.

6 Discussion and future work

In this work we propose an improved method to learn a mobile manipulation skill in simulation and show its feasibility in a real robotic system. Specifically, we learn a DRL-based reactive controller for the base which by sending velocity commands is able to position the mobile manipulator's base in zones that enable a successful picking. The rationale behind our approach is to encourage the base controller to position the robot's in zones where the arm's planner will likely succeed planning a trajectory up to the target object.

First, we develop a realistic simulation of the real environment in Unity, which also requires the development of simulated low-level drivers both for the mobile base and the robotic arm. Then, the agent that is in charge of controlling the robot's base is modelled based on the TD3 state-of-the-art DRL algorithm. We train it in simulation through the interaction with the environment and by introducing basic randomisation to make the learnt agent robust to uncertainties. The successful training of the controller requires a careful design of the reward function, as well as a correct definition of state/action spaces and neural architectures.

During the training process in simulation, we benchmark our method with the baseline approach. First we compare the training curves of TD3 with DDPG and PPO2 proposed in the baseline work, with both the proposed and the baseline setup, from the perspective of performance and reliability. Then, the same comparison is done at evaluation, running the trained models for 100 episodes. In both cases, TD3 showed to be the algorithm with the highest performance, and also the most reliable. The success rates indicate that the best performance is obtained by the combination of TD3 and the setup proposed in this work, with a success rate of 97%. Also the

average travelled distances indicate that this combination leads to shorter trajectories in average.

Finally, the learnt base controller is deployed and validated in the real system as well. Even though the performance of the controller is remarkable in simulation, it worsens in the real system. In fact, the most problematic step of the execution of the task is when the robot's base needs to stop while positioning near the target. On the one hand, the simulation-to-reality gap and the error introduced by the AMCL cause an oscillation that prevents the robot's base from successfully stopping in the grasping zone. This oscillation is the main source of the brushes with the table and prevents the robot from staying still until the grasping trial is performed. On the other hand, although the manipulation itself is out of the scope of this work, the drift introduced by AMCL in the localisation is big enough to cause unsuccessful manipulations.

Therefore, to assess the real performance of the proposed system, we measure the real position of the robot and we use it as ground truth. By doing so, we intend to effectively measure whether the target object is really reachable for the robotic arm in spite of the localisation error. The performed experiments show that the proposed system successfully learns to position the robot's base in suitable picking areas with a success rate of 75%. Despite the localisation error, the results show that in all cases the object is really within arm's reach. This means that the use of a more accurate localisation system would allow the object to be successfully grasped.

In summary, we show that the proposed system successfully positions the mobile manipulator in zones that ensure the reachability of the arm to the target object. However, due to the error introduced by current localisation systems, the manipulator must have an on-board vision system to be able to accurately estimate the relative pose between the arm's end effector and the target before executing the grasp. In addition, the main future work lines will be focused on adding more sensing capabilities to the agent to increase the safety of the navigation, and on reducing the simulation-to-reality gap. On the one hand, the use of sensors such as lasers, cameras, etc. will let the agent sense the dynamic elements in the environment to safely navigate to the target. On the other hand, more advanced domain randomisation techniques will let us reduce the simulation-to-reality gap. Due to the difficulty in properly simulating the physical properties of both the robot and the environment, domain randomisation techniques suggest to randomise these physical properties in simulation, assuming that the real world properties are a particular case of the randomised variables.

Acknowledgments. This publication has been funded by the Basque Government - Department of Economic Development, Sustainability and Environment - Aid program for collaborative research in strategic areas - ELKARTEK 2021 Program (File KK-2021/00033 TREBEZIA), and the project "5R- Red Cervera de Tecnologías robóticas en fabricación inteligente", contract number CER-20211007, under "Centros Tecnológicos de Excelencia Cervera" programme funded by "The Centre for the Development of Industrial Technology (CDTI)".

Competing interests. The authors have no competing interests to declare that are relevant to the content of this article.

Data Availability. Data sharing not applicable to this article as no datasets were generated or analysed during the current study.

References

[1] Sandakalum T, Ang Jr MH (2022) Motion planning for mobile manipulators—a systematic review. Machines 10(2):97. https://doi.org/10.3390/machines10020097

[2] Stulp F, Fedrizzi A, Mösenlechner L, et al (2012) Learning and reasoning with action-related places for robust mobile manipulation. Journal of Artificial Intelligence Research 43:1–42. https://doi.org/10.1613/jair.3451

[3] Kappler D, Pastor P, Kalakrishnan M, et al (2015) Data-driven online decision making for autonomous manipulation. In: Robotics: science and systems, https://doi.org/10.15607/RSS.2015.XI.044

[4] Arulkumaran K, Deisenroth MP, Brundage M, et al (2017) Deep reinforcement learning: A brief survey. IEEE Signal Processing Magazine 34(6):26–38. https://doi.org/10.1109/MSP.2017.2743240

[5] Yang X, Xu Y, Kuang L, et al (2021) An information fusion approach to intelligent traffic signal control using the joint methods of multiagent reinforcement learning and artificial intelligence of things. IEEE Transactions on Intelligent Transportation Systems https://doi.org/10.1109/TITS.2021.3105426

[6] Iriondo A, Lazkano E, Susperregi L, et al (2019) Pick and place operations in logistics using a mobile manipulator controlled with deep reinforcement learning. Applied Sciences 9(2):348. https://doi.org/10.3390/app9020348

[7] Fujimoto S, Hoof H, Meger D (2018) Addressing function approximation error in actor-critic methods. In: International Conference on Machine Learning, PMLR, pp 1587–1596, URL https://proceedings.mlr.press/v80/fujimoto18a.html

[8] Juliani A, Berges VP, Teng E, et al (2018) Unity: A general platform for intelligent agents. arXiv preprint arXiv:180902627 URL https://arxiv.org/pdf/1809.02627.pdf

[9] Quigley M, Conley K, Gerkey B, et al (2009) Ros: an open-source robot operating system. In: ICRA workshop on open source software, Kobe, Japan, p 5, URL http://robotics.stanford.edu/~ang/papers/icraoss09-ROS.pdf

[10] Brockman G, Cheung V, Pettersson L, et al (2016) Openai gym. arXiv preprint arXiv:160601540 URL https://arxiv.org/pdf/1606.01540.pdf

[11] Siciliano B, Khatib O (2016) Springer handbook of robotics. Springer, URL https://link.springer.com/content/pdf/10.1007%2F978-3-319-32552-1.pdf

[12] Marder-Eppstein E, Berger E, Foote T, et al (2010) The office marathon: Robust navigation in an indoor office environment. In: IEEE international conference on robotics and automation, IEEE, pp 300–307, https://doi.org/10.1109/ROBOT.2010.5509725

[13] Coleman D, Sucan I, Chitta S, et al (2014) Reducing the barrier to entry of complex robotic software: a moveit! case study. arXiv preprint arXiv:14043785

https://doi.org/10.6092/JOSER_2014_05_01_p3

[14] Dömel A, Kriegel S, Kaßecker M, et al (2017) Toward fully autonomous mobile manipulation for industrial environments. International Journal of Advanced Robotic Systems 14(4):1729881417718,588. https://doi.org/10.1177/1729881417718588

[15] Xu J, Harada K, Wan W, et al (2020) Planning an efficient and robust base sequence for a mobile manipulator performing multiple pick-and-place tasks. In: IEEE International Conference on Robotics and Automation (ICRA), IEEE, pp 11,018–11,024, https://doi.org/10.1109/ICRA40945.2020.9196999

[16] Padois V, Fourquet JY, Chiron P (2006) From robotic arms to mobile manipulation: On coordinated motion schemes. In: Intelligent Production Machines and Systems. Elsevier, p 572–577, URL https://hal.archives-ouvertes.fr/hal-00624374/file/2006ACTI1475.pdf

[17] Tan J, Xi N, Wang Y (2003) Integrated task planning and control for mobile manipulators. The International Journal of Robotics Research 22(5):337–354. https://doi.org/10.1177/0278364903022005004

[18] Berntorp K, Arzén KE, Robertsson A (2012) Mobile manipulation with a kinematically redundant manipulator for a pick-and-place scenario. In: Control Applications (CCA), 2012 IEEE International Conference on, IEEE, pp 1596–1602, https://doi.org/10.1109/CCA.2012.6402361

[19] Meeussen W, Wise M, Glaser S, et al (2010) Autonomous door opening and plugging in with a personal robot. In: Robotics and Automation (ICRA), IEEE International Conference on, IEEE, pp 729–736, https://doi.org/10.1109/ROBOT.2010.5509556

[20] Ibarguren A, Daelman P (2021) Path driven dual arm mobile co-manipulation architecture for large part manipulation in industrial environments. Sensors 21(19):6620. https://doi.org/10.3390/s21196620

[21] Lin S, Goldenberg AA (2001) Neural-network control of mobile manipulators. IEEE Transactions on Neural Networks 12(5):1121–1133. https://doi.org/10.1109/72.950141

[22] Konidaris G, Kuindersma S, Grupen R, et al (2011) Autonomous skill acquisition on a mobile manipulator. In: Twenty-Fifth AAAI Conference on Artificial Intelligence, https://doi.org/10.1609/aaai.v25i1.7982

[23] Ibarz J, Tan J, Finn C, et al (2021) How to train your robot with deep reinforcement learning: lessons we have learned. The International Journal of Robotics Research 40(4-5):698–721. https://doi.org/10.1177/0278364920987859

[24] Mohammed MQ, Chung KL, Chyi CS (2020) Review of deep reinforcement learning-based object grasping: Techniques, open challenges and recommendations. IEEE Access https://doi.org/10.1109/ACCESS.2020.3027923

[25] Hansen J, Hogan F, Rivkin D, et al (2022) Visuotactile-rl: Learning multimodal manipulation policies with deep reinforcement learning. In: 2022 International Conference on Robotics and Automation (ICRA), IEEE, pp 8298–8304, https://doi.org/10.1109/ICRA46639.2022.9812019

[26] Zhu K, Zhang T (2021) Deep reinforcement learning based mobile robot navigation: A review. Tsinghua Science and Technology 26(5):674–691. https://doi.org/10.26599/TST.2021.9010012

[27] Haarnoja T, Ha S, Zhou A, et al (2018) Learning to walk via deep reinforcement learning. arXiv preprint arXiv:181211103 URL https://arxiv.org/pdf/1812.11103.pdf

[28] Peng XB, Berseth G, Yin K, et al (2017) Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. ACM Transactions on Graphics (TOG) 36(4):1–13. https://doi.org/10.1145/3072959.3073602

[29] Kalashnikov D, Irpan A, Pastor P, et al (2018) Scalable deep reinforcement learning for vision-based robotic manipulation. In: Conference on Robot Learning, PMLR, pp 651–673, URL https://proceedings.mlr.press/v87/kalashnikov18a.html

[30] Jangir R, Alenyà G, Torras C (2020) Dynamic cloth manipulation with deep reinforcement learning. In: IEEE International Conference on Robotics and Automation (ICRA), IEEE, pp 4630–4636, https://doi.org/10.1109/ICRA40945.2020.9196659

[31] Lillicrap TP, Hunt JJ, Pritzel A, et al (2016) CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING. arXiv preprint arXiv:150902971 URL https://arxiv.org/pdf/1509.02971.pdf

[32] Kim M, Han DK, Park JH, et al (2020) Motion planning of robot manipulators for a smoother path using a twin delayed deep deterministic policy gradient with hindsight experience replay. Applied Sciences 10(2):575. https://doi.org/10.3390/app10020575

[33] Hsu D, Latombe JC, Kurniawati H (2006) On the probabilistic foundations of probabilistic roadmap planning. The International Journal of Robotics Research 25(7):627–643. https://doi.org/10.1177/0278364906067174

[34] Tai L, Paolo G, Liu M (2017) Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, pp 31–36, https://doi.org/10.1109/IROS.2017.8202134

[35] Wang C, Wang J, Shen Y, et al (2019) Autonomous navigation of uavs in large-scale complex environments: A deep reinforcement learning approach. IEEE Transactions on Vehicular Technology 68(3):2124–2136. https://doi.org/10.1109/TVT.2018.2890773

[36] Dankwa S, Zheng W (2019) Modeling a continuous locomotion behavior of an intelligent agent using deep reinforcement technique. In: IEEE 2nd International Conference on Computer and Communication Engineering Technology (CCET), pp 172–175, https://doi.org/10.1109/CCET48361.2019.8989177

[37] Khoi P, Giang N, Tan H (2021) Control and simulation of a 6-dof biped robot based on twin delayed deep deterministic policy gradient algorithm. Indian Journal of Science and Technology 14(30):2460–2471. https://doi.org/10.17485/IJST/v14i30.1030

[38] Kindle J, Furrer F, Novkovic T, et al (2020) Whole-body control of a mobile manipulator using end-to-end reinforcement learning. arXiv preprint arXiv:200302637 URL https://arxiv.org/pdf/2003.02637.pdf

[39] Wang C, Zhang Q, Tian Q, et al (2020) Learning mobile manipulation through deep reinforcement learning. Sensors 20(3):939. https://doi.org/10.3390/s20030939

[40] Schulman J, Wolski F, Dhariwal P, et al (2017) Proximal Policy Optimization Algorithms. arXiv preprint arXiv:170706347 pp 1–12. https://arxiv.org/abs/arXiv:1707.06347

[41] Bischof M (2018) ROS-SHARP. https://github.com/siemens/ros-sharp, [Accessed: 16-01-2023]

[42] Qian W, Xia Z, Xiong J, et al (2014) Manipulation task simulation using ros and gazebo. In: IEEE International Conference on Robotics and Biomimetics (ROBIO 2014), IEEE, pp 2594–2598, https://doi.org/10.1109/ROBIO.2014.7090732

[43] Chitta S, Marder-Eppstein E, Meeussen W, et al (2017) ros_control: A generic and simple control framework for ros. The Journal of Open Source Software https://doi.org/10.21105/joss.00456

[44] Sutton RS, Barto AG (1998) Reinforcement learning: An introduction, vol 1. MIT press Cambridge, URL https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf

[45] Foote T (2013) tf: The transform library. In: Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on, Open-Source Software workshop, pp 1–6, https://doi.org/10.1109/TePRA.2013.6556373

[46] Silver D, Lever G, Heess N, et al (2014) Deterministic policy gradient algorithms. In: Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32, ICML'14, p I–387–I–395, URL http://proceedings.mlr.press/v32/silver14.pdf

[47] Mnih V, Kavukcuoglu K, Silver D, et al (2015) Human-level control through deep reinforcement learning. nature 518(7540):529–533. https://doi.org/10.1038/nature14236

[48] Sutton RS (1988) Learning to predict by the methods of temporal differences. Machine learning 3(1):9–44. https://doi.org/10.1007/BF00115009

[49] Hill A, Raffin A, Ernestus M, et al (2018) Stable baselines. URL https://github.com/hill-a/stable-baselines

[50] Murtagh F (1991) Multilayer perceptrons for classification and regression. Neurocomputing 2(5-6):183–197. https://doi.org/10.1016/0925-2312(91)90023-5

[51] Ng AY, Harada D, Russell S (1999) Policy invariance under reward transformations: Theory and application to reward shaping. In: Icml, pp 278–287

[52] Chan SC, Fishman S, Canny J, et al (2020) Measuring the reliability of reinforcement learning algorithms. In: International Conference on Learning Representations, Addis Ababa, Ethiopia, URL https://openreview.net/pdf?id=SJlpYJBKvH

[53] Riaz M (2015) On enhanced interquartile range charting for process dispersion. Quality and Reliability Engineering International 31(3):389–398. https://doi.org/10.1002/qre.1598

[54] Acerbi C, Tasche D (2002) Expected shortfall: a natural coherent alternative to value at risk. Economic notes 31(2):379–388. https://doi.org/10.1111/1468-0300.00091

[55] Chekhlov A, Uryasev S, Zabarankin M (2005) Drawdown measure in portfolio optimization. International Journal of Theoretical and Applied Finance 8(01):13–58. https://doi.org/10.1142/S0219024905002767

[56] Fox D, Burgard W, Dellaert F, et al (1999) Monte carlo localization: Efficient position estimation for mobile robots. AAAI/IAAI (343-349):2–2. URL http://robots.stanford.edu/papers/fox.aaai99.pdf

[57] Grisetti G, Stachniss C, Burgard W (2007) Improved techniques for grid mapping with rao-blackwellized particle filters. IEEE transactions on Robotics 23(1):34–46. https://doi.org/10.1109/TRO.2006.889486

Appendix A Agent training and evaluation algorithms

Algorithm 1 Training of TD3-based agent

Initialise critic networks Q_{w_1} and Q_{w_2}, and actor network π_θ with random parameters w_1, w_2, θ.
Initialise target networks $w_1' \leftarrow w_1$,$w_2' \leftarrow w_2$, $\theta' \leftarrow \theta$
Initialise replay buffer $\mathbb{B}$
$env \leftarrow gym.environment()$
$i \leftarrow 0$
while $i < train\_steps$ **do**
    randomiseScene()
    $s_t \leftarrow env.reset()$
    for $t = 0$ .. $T - 1$ **do**
        Select action with exploration noise $a_t \approx \pi_\theta(s_t) + \epsilon$, $\epsilon \approx \mathcal{N}(0, \sigma)$
        $s_{t+1}, r_t, done \leftarrow env.step(a_t)$
        if $t = T - 1$ **then**
            $plan? \leftarrow doPickingTrial()$
            if $plan?$ **then**
                $r_t \leftarrow r_t + G$
            end if
        end if
        Store transition tuple (s_t, a_t, r_t, s_{t+1}) in $\mathbb{B}$
        if done **then**
            break
        end if
    end for
    if i mod $train\_freq$ **then**
        Sample minibatch of N transitions (s_t, a_t, r_t, s_{t+1}) from $\mathbb{B}$
        $\tilde{a} \leftarrow \pi_{\theta'}(s) + \epsilon', \epsilon' \approx clip(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
        $y \leftarrow r + \gamma min_{i=1,2} Q'_{w_i'}(s_{t+1}, \tilde{a})$
        Update the critics $w_i \leftarrow argmin_{w_i} \frac{1}{N} \sum (y - Q_{w_i}(s, a))^2$
        if i mod d **then**
            Update θ by the deterministic policy gradient
            $\nabla_\theta J(\theta) = \frac{1}{N} \sum \nabla_a Q_{w_1}(s, a)|_{a=\pi_\theta(s)} \nabla_\theta \pi_\theta(s)$
            Update the target networks:
            $w_i' \leftarrow \tau w_i + (1 + \tau) w_i'$
            $\theta_i' \leftarrow \tau \theta_i + (1 + \tau) \theta_i'$
        end if
    end if
    $i \leftarrow i + t$
    $s_t \leftarrow s_{t+1}$
end while

Algorithm 2 Evaluation of TD3-based agent

Initialise actor network π_θ with trained parameters θ.
$accumulated\_rewards \leftarrow []$
$successes \leftarrow 0$
$env \leftarrow gym.environment()$
for $i = 0$ .. $eval\_episodes$ **do**
    $accumulated\_reward \leftarrow 0$
    randomiseScene()
    $s_t \leftarrow env.reset()$
    for $t = 0$ .. $T - 1$ **do**
        Select optimal action $a_t = \pi_\theta(s_t)$
        $s_{t+1}, r_t, done \leftarrow env.step(a_t)$
        if $t = T - 1$ **then**
            $plan? \leftarrow doPickingTrial()$
            if $plan?$ **then**
                $r_t \leftarrow r_t + G$
                $successes \leftarrow successes + 1$
            end if
        end if
        $accumulated\_reward \leftarrow accumulated\_reward + r_t$
        if done **then**
            break
        end if
    end for
    $accumulated\_rewards.append(accumulated\_reward)$
end for
$success\_rate \leftarrow \frac{successes}{eval\_episodes}$
$average\_accumulated\_reward \leftarrow average(accumulated\_rewards)$