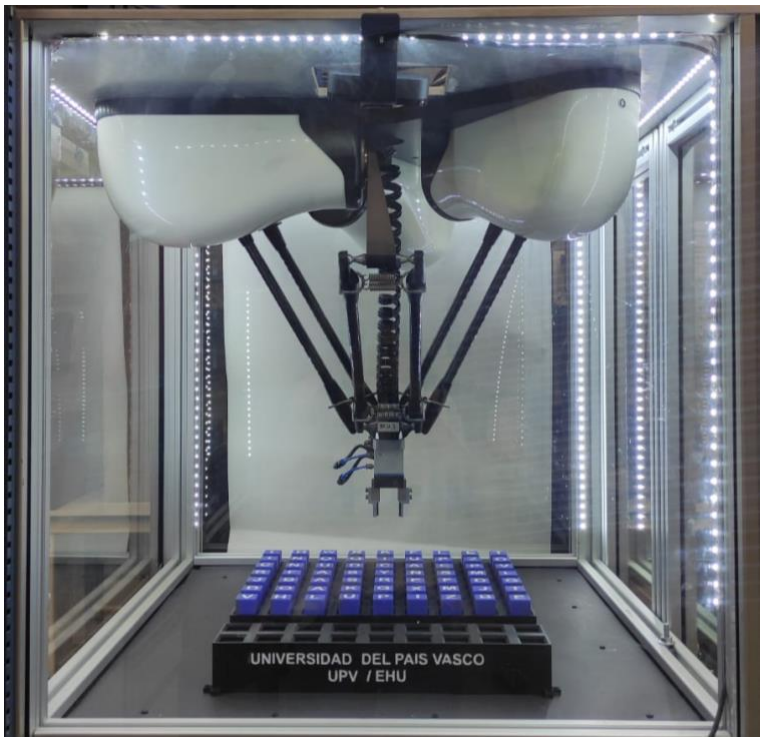
Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

MÁSTER UNIVERSITARIO EN INGENIERÍA DE CONTROL, AUTOMATIZACIÓN Y ROBÓTICA

TRABAJO FIN DE MASTER

IMPLEMENTACIÓN DE TÉCNICAS INTELIGENTES DE APRENDIZAJE PARA APLICACIÓN PICK & PLACE EN ROBOT PARALELO DELTA



Estudiante: Saratxaga Cortés, Enara

Director/Directora: Cabanes Axpe, Itziar

Curso: 2022/2023

Fecha: Bilbao, 20 de septiembre de 2023

RESUMEN

La llegada de la Industria 4.0 ha traído consigo grandes avances en cuanto a conectividad, gracias a los cuales se han creado las denominadas fábricas inteligentes, y se han logrado aumentos en la productividad. En estas fábricas es cada vez más común encontrar robots paralelos debido a su gran potencial, precisión y alta velocidad.

A su vez, la implementación de sistemas de visión artificial en el ámbito referente a la Robótica aporta gran flexibilidad a las industrias, dotándolas de inteligencia y permitiendo la toma de decisiones autónoma.

En el presente Trabajo de Fin de Máster, se ha desarrollado una solución para la implementación de Inteligencia Artificial en una célula robotizada para aplicaciones de pick & place mediante visión artificial.

La célula robotizada está compuesta por un robot paralelo tipo mini Delta del fabricante Omron, que ha sido programado a través del entorno de programación Sysmac Studio. La célula robotizada también cuenta con una webcam modelo Logitech Brio 4k UltraHD, la cual ha sido utilizada para realizar el reconocimiento de los elementos que se encuentran en el espacio de trabajo del robot.

El algoritmo de Inteligencia Artificial desarrollado ha consistido en una red neuronal convolucional, a través de la cual se realiza el reconocimiento de los diferentes caracteres contenidos en los cubos ubicados en el espacio de trabajo el robot. El usuario desde una interfaz gráfica puede seleccionar la palabra que desea formar, y, tras realizar el reconocimiento de los caracteres requeridos, el sistema de manera automática captura la ubicación de las letras necesarias para formar la palabra seleccionada. Después, se envía la información referente a la ubicación de los caracteres al robot, el cual se traslada para coger y dejar los cubos en las posiciones necesarias para escribir la palabra deseada.

El entrenamiento de la red neuronal convolucional se ha realizado a través del lenguaje de programación Python utilizando el set de datos EMNIST, sin embargo, el preprocesamiento de las imágenes y, por lo tanto, la preparación de las entradas de la red se ha realizado a través de MATLAB.

Palabras clave: Robótica paralela, Visión, Inteligencia Artificial, Red Neuronal Convolucional, Reconocimiento de caracteres, EMNIST.

LABURPENA

Industria 4.0-ren etorrerak aurrerapauso handiak ekarri ditu konektibitateari dagokionez. Aurrerapen horiek lantegi adimenduak sortzea ekarri dituzte, produktibitatea handituz. Lantegi horietan robot paraleloak aurkitzea gero eta ohikoagoa da, hauek duten potentzial, doitasun eta abiadura handiagatik.

Era berean, robotikaren esparruan ikusmen artifizialeko sistemak ezartzeak malgutasun handia ematen die industriei, adimenez hornituz eta erabakiak autonomiaz hartzeko aukera emanez.

Master-amaierako lan honetan, adimen artifiziala ikusmen artifizialaren bidez pick & place aplikazioetarako zelula robotizatu batean ezartzeko soluzioa garatu da.

Omron fabrikatzailearen Mini Delta motako robot paralelo batek osatzen du zelula robotizatua, Sysmac Studio programazio-ingurunean programatu dena. Zelula robotizatuak Logitech Brio 4k UltraHD webcam-a ere badu, eta robotaren lan-eremuan dauden elementuak errekonozitzeko erabili da.

Garaturako adimen artifizialaren algoritmoa konboluzioko sare neuronal bat izan da. Sare horren bidez errekonozituko dira robotaren lan-eremuan kokatutako kuboetako karaktereak. Interfaze grafiko baten bidez, erabiltzaileak sortu nahi duen hitza hauta dezake, eta, behar diren karaktereak errekonozitu ondoren, sistemak automatikoki atzematen du aukeratutako hitza osatzeko behar diren letren kokapena. Ondoren, karaktereen kokapenari buruzko informazioa robotari bidaltzen zaio, eta robotak eskatutako hitza idazteko behar diren lekuetan uzten ditu kuboak.

Sare neuronal konboluzionalaren entrenamendua Python programazio-lengoaiaren bidez egin da, EMNIST datu-set-a erabiliz, baina irudien aurreprozesaketa, eta beraz, sareko sarreren prestaketa MATLAB-en bidez egin da.

Hitz gakoak: Robotika paraleloa, Ikusmena, Adimen Artifiziala, Sare Neuronal Konboluzionala, Karaktereen Errekonozimendua, EMNIST.

ABSTRACT

The advent of Industry 4.0 has brought with it great advances in terms of connectivity, thanks to which so-called smart factories have been created, and productivity gains have been achieved. In these factories it is increasingly common to find parallel robots due to their great potential, precision and high speed.

At the same time, the implementation of artificial vision systems in the field of Robotics brings great flexibility to industries, providing them with intelligence and allowing autonomous decision making.

In this Master Thesis, a solution has been developed for the implementation of Artificial Intelligence in a robotic cell for pick & place applications by means of artificial vision.

The robotic cell is composed of a parallel robot type mini Delta from Omron, which has been programmed through the programming environment Sysmac Studio. The robotic cell also has a webcam model Logitech Brio 4k UltraHD, which has been used to perform the recognition of the elements that are in the robot's workspace.

The Artificial Intelligence algorithm developed has consisted of a convolutional neural network, through which the recognition of the different characters contained in the cubes located in the robot's workspace is performed. The user can select the word to be formed from a graphical interface and, after recognizing the required characters, the system automatically captures the location of the letters needed to form the selected word. Then, the information regarding the location of the characters is sent to the robot, which moves to pick up and place the cubes in the required positions to write the desired word.

The training of the convolutional neural network has been performed through the Python programming language using the EMNIST dataset, however, the preprocessing of the images and, therefore, the preparation of the network’s input has been performed through MATLAB.

Keywords: Parallel Robotics, Vision, Artificial Intelligence, Convolutional Neural Network, Character Recognition, EMNIST.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ACRÓNIMOS

TFM	Trabajo Fin de Máster
RGB	Reed-Green-Blue
BW	Black and White
3D	Tres dimensiones
Pick and Place	Cogida y dejada
PLC	Controlador lógico programable
TCP	Transmission Control Protocol
IA	Inteligencia Artificial
SVM	Support Vector Machine
MNIST	Modified National Institute of Standards and Technology
EMNIST	Extended Modified National Institute of Standards and Technology

Capítulo 1
Introducción

1.1. Introducción

El presente Trabajo de Fin de Máster se ha desarrollado en el Departamento de Ingeniería de Sistemas y Automática de la Escuela de Ingeniería de Bilbao, en la Universidad del País Vasco (UPV/EHU). En concreto, se ha llevado a cabo en calidad de cooperación educativa en el grupo de investigación ViSens del departamento.

La transformación digital se ha convertido en una estrategia muy importante para aumentar la competitividad de una empresa y enfrentar los desafíos actuales. Uno de los principales objetivos de la transformación digital es la automatización, puesto que gracias a ella se obtiene mayor productividad, mejoras en la eficiencia, reducciones de costos y mejoras en la calidad de los productos y servicios.

En los procesos industriales la automatización tiene un impacto muy considerable en la capacidad de la empresa para satisfacer las demandas de los clientes y adaptarse a los cambios del mercado.

Una de las formas que tiene la transformación digital de impulsar la automatización de los procesos productivos es mediante la integración de tecnologías avanzadas, tales como la inteligencia artificial (IA) y la visión artificial. Estas tecnologías aportan ventajas competitivas a las empresas, al automatizar trabajos que previamente requerían de intervención humana, se mejora la calidad, precisión, etc. de los productos.

Gracias a la incorporación de cámaras en la industria se puede obtener información sobre los procesos de manera precisa y rápida, incluso en tiempo real. Pero la adquisición de los datos no es suficiente, es necesario procesarlos de forma correcta para poder lograr su máximo provecho, y ahí es donde irrumpe la inteligencia artificial.

Otra de las tecnologías claves en el proceso de la transformación digital es la Robótica, ya que permite automatizar procesos de forma más rápida y precisa. En concreto, la robótica paralela aporta mayor rapidez y precisión que la robótica industrial.

Al integrar la inteligencia artificial y la visión artificial a la robótica paralela se puede conseguir la detección automática de objetos, mayor autonomía, agilidad, robustez, flexibilidad, etc., además, también se pueden ajustar las posiciones y trayectorias del robot en tiempo real.

En este trabajo se pretende integrar las tres tecnologías (Robótica, Visión artificial y IA) en una aplicación de pick&place robotizada.

1.2. Contextualización

Este trabajo se plantea como una ampliación del TFM previo "Aplicación de pick and place con robot paralelo mini delta integrando visión artificial" [1], al cual se pretende integrar la inteligencia artificial para la identificación de objetos de forma autónoma. En la Figura 1 se observa el robot utilizado, un robot Delta del fabricante Omron.

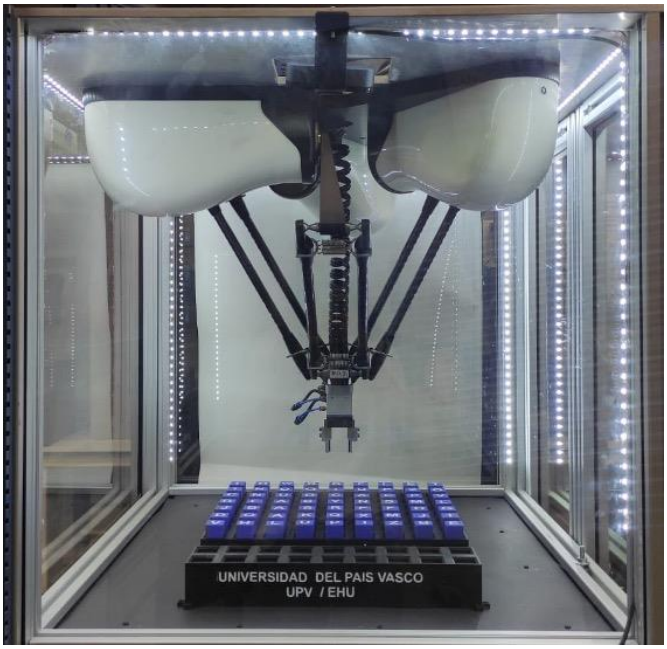
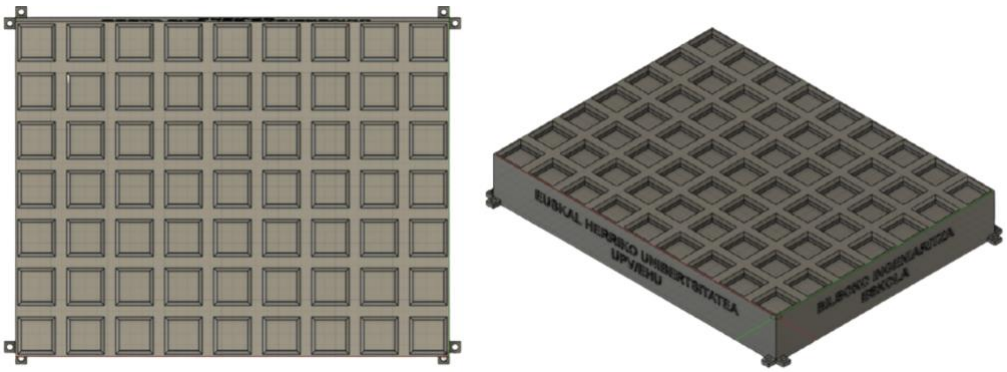


Figura 1. Robot Mini Delta.

En el trabajo nombrado se diseñó el escenario de trabajo del robot mediante la integración de dos plataformas: una grande y una pequeña, tal y como se muestra en la Figura 2. Así como los cubos que integran las letras a reconocer, Figura 3.



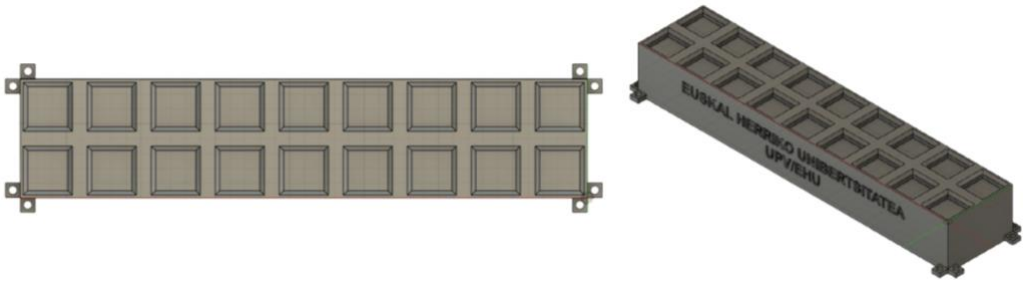


Figura 2. Plataforma grande (arriba) y plataforma pequeña (abajo).

Figura 3. Cubos a reconocer.

En ese caso, se realizó la programación y control del robot, para una aplicación de visión estática o fija, a través de la cual se puede identificar los objetos detectados y extraer sus características.

Sin embargo, la aplicación de visión desarrollada no abordaba cambios dinámicos en las plataformas que contenían las letras (carecía de inteligencia artificial). Por ello, el reconocimiento de los caracteres se realizaba a través del estudio del área y perímetro de los mismos, y esto supone varios problemas. Por un lado, la cámara no está ubicada en vertical al centro de la plataforma, por lo que las imágenes se obtienen en perspectiva. Como consecuencia, las letras de los diferentes cubos sufren una distorsión en función de la posición que adopten en la plataforma, lo que conlleva a que una misma letra pueda dar valores muy diferentes de área y perímetro en función de su posicionamiento. Por otro lado, las letras debían estar siempre en la misma posición del tablero, ya que el sistema no actualizaba cambios en las localizaciones de las letras.

Lo recientemente expuesto hace que la aplicación carezca de variabilidad, tanto en cuanto a posicionamiento, como se acaba de explicar, como en cuanto a tipografía de letras. Esto último se debe a que al modificar la tipografía de letra los valores de área y perímetro cambian para cada carácter, por lo que es imposible contemplar tantas opciones diferentes.

Por esto, se ha visto la necesidad de aplicar técnicas inteligentes de aprendizaje en la aplicación pick and place mencionada, para poder detectar todas las letras de forma automática e independientemente de la posición que adopten.

1.3. Estructura del documento

En este apartado se expone la estructura del presente Trabajo Fin de Máster que se divide en 9 capítulos y 4 anexos:

1. En el Capítulo 1, se presenta una introducción y contextualización del trabajo.
2. En el Capítulo 2, se tratan los objetivos principales y secundarios del proyecto, junto con el alcance del mismo.
3. En el Capítulo 3, se expone el estado actual de la industria 4.0 y las nuevas tecnologías.
4. En el Capítulo 4, se realiza un análisis de alternativas de las técnicas y software de inteligencia artificial a utilizar.
5. En el Capítulo 5, se explican las tareas realizadas, junto con los resultados obtenidos en ellas.
6. En el Capítulo 6, se explica la metodología seguida en el desarrollo del proyecto, explicando cada una de las tareas y el tiempo dedicado en ellas.
7. En el Capítulo 7, se realiza el cálculo del presupuesto necesario para realizar el proyecto.
8. En el Capítulo 8, se exponen las conclusiones obtenidas en el proyecto.
9. En el Capítulo 9, se documenta la bibliografía del proyecto.

I. En el Anexo I, se muestran los planos de las piezas 3D diseñadas.
II. En el Anexo II, se presenta el código de diseño de la red neuronal en Python.
III. En el Anexo III, se expone el código de implementación en MATLAB.
IV. En el Anexo IV, se muestra un manual de usuario, explicando los pasos a seguir para lograr el correcto funcionamiento de la aplicación.

Capítulo 2

Objetivos y alcance del trabajo

2. Objetivos y alcance del trabajo

En el apartado actual se realiza una descripción del objetivo principal a conseguir y se especifican los objetivos parciales para alcanzar el objetivo principal. Por último, se limita el alcance del proyecto.

2.1. Objetivos del proyecto

El objetivo principal de este proyecto es desarrollar mediante técnicas de inteligencia artificial una aplicación de visión artificial para el reconocimiento de letras, que serán manipuladas por un sistema robótico para una aplicación de pick and place.

Para lograr el objetivo del presente trabajo mencionado, se definen los siguientes objetivos parciales:

- Diseño del espacio de trabajo del robot: Selección de los elementos a utilizar en el espacio de trabajo del robot, así como escoger las herramientas (pinzas) requeridas y modelar los elementos auxiliares (dedos de las pinzas) necesarios, para su posterior impresión 3D.
- Aprendizaje de los programas MATLAB, Python, Fusion 360 y Sysmac Studio: MATLAB es frecuentemente utilizado para trabajos de visión y diseño de aplicaciones de usuario, Python para el diseño y modelado de tecnologías de Inteligencia Artificial, Fusion 360 para el modelado 3D de elementos y Sysmac Studio para la programación del robot.
- Modelado y entrenamiento del algoritmo de reconocimiento de caracteres: dado que el objetivo principal es el reconocimiento de caracteres a través de un algoritmo de inteligencia artificial será necesario modelar y entrenar una arquitectura adecuada de red neuronal a través de la herramienta Python.
- Implementación de técnicas de preprocesamiento de imágenes, como corrección de la perspectiva y segmentación de las letras de los cubos, para mejorar la calidad de las imágenes y el reconocimiento de las letras.
- Cohesión de la aplicación de usuario con el modelo que realizará las predicciones de las imágenes procesadas y con el software del robot, para lograr que la aplicación en conjunto funcione correctamente.
- Implementación del sistema y validación.

2.2. Alcance

El alcance de este Trabajo de Fin de Máster comprende el desarrollo de una aplicación de Inteligencia Artificial que permita el reconocimiento y manipulación de letras mayúsculas y minúsculas impresas en cubos utilizando técnicas de visión artificial y aprendizaje profundo. Sin embargo, no es alcance de este proyecto la identificación de caracteres numéricos, caracteres especiales u otro tipo de iconos.

Capítulo 3

Estudio de la tecnología en robótica, visión e inteligencia artificial

3. Estudio de la tecnología en robótica, visión e inteligencia artificial

En este apartado se realiza un estudio que permite conocer la situación actual en la que se encuentra la industria, poniendo el foco en las nuevas tecnologías desarrolladas, como la Inteligencia Artificial o el Internet of Things, y, en cómo se implementan estas tecnologías en la robótica.

3.1. La Industria 4.0 y las nuevas tecnologías

El concepto de Industria 4.0 se basa en conseguir un flujo de producción completamente integrado, automatizado y optimizado, logrando así mayor productividad y eficiencia en las fábricas. Para lograr este objetivo se han implementado nuevas tecnologías en la industria manufacturera. Ejemplo de estas tecnologías son la inteligencia artificial, la robótica, la computación en la nube y el Internet of Things (IoT), entre otras [2], Figura 4.

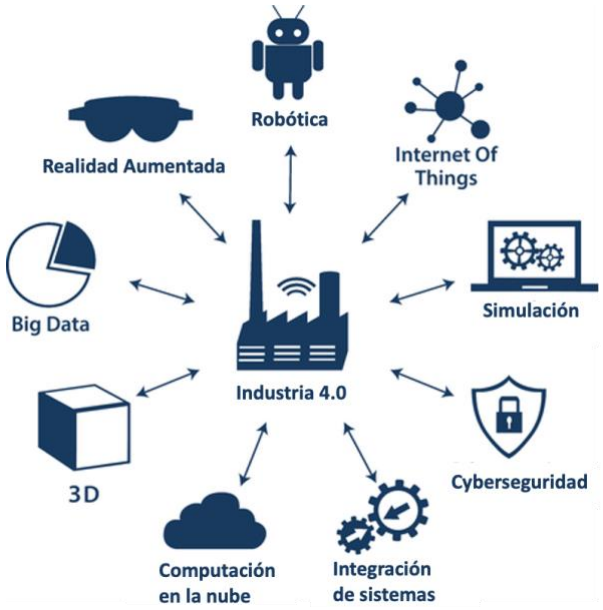


Figura 4. Nuevas tecnologías de la Industria 4.0.

La integración de la tecnología IoT en un proceso productivo, permite convertir recursos de producción típicos en objetos de fabricación inteligentes capaces de detectar, interconectarse e interactuar entre sí para llevar a cabo lógicas de fabricación de forma automática y adaptativa, lo cual proporciona gran flexibilidad y versatilidad a las máquinas y plantas industriales en las que se implementa [3].

Un ejemplo de esto podrían ser las aplicaciones robóticas. Antes de la implantación de la Industria 4.0, el objetivo de dichas aplicaciones se basaba en sustituir los trabajos repetitivos y de grandes cargas que tenía que soportar el personal de planta, es decir, los robots no tenían poder de decisión. Sin embargo, desde la integración de las nuevas tecnologías de la Industria 4.0, entre otras, gracias a los sistemas de visión, se dota a los robots de capacidad para la toma

de decisiones en función de las necesidades del proceso. Por ello, las aplicaciones robóticas pueden ser personalizadas.

En cuanto a robótica, se distinguen dos tipos de robots según su estructura: los robots serie y los robots paralelos, Figura 5. En entornos industriales lo más común es encontrar robots tipo serie, los cuales están formados por una cadena cinemática abierta, y cuentan con mayor espacio de trabajo que los robots paralelos. En cambio, los robots paralelos están formados por una plataforma móvil conectada a una base mediante al menos dos series de cadenas cinemáticas independientes, por lo tanto, tienen una configuración de cadena cinemática cerrada. Esta última configuración concede diferentes ventajas en términos de velocidad, precisión, rigidez y repetibilidad a los robots paralelos con respecto a los robots serie [4].

Figura 5. Robot serie (izquierda) y robot paralelo (derecha).

Uno de los mecanismos paralelos de mayor éxito es el robot Delta, Figura 6, un tipo de robot paralelo de alta velocidad, posicionamiento preciso, bajo coste y alta eficiencia. Este robot es ampliamente utilizado en líneas de producción de alimentos, medicamentos y productos electrónicos para realizar tareas de clasificación rápida, agarre y ensamblaje entre otros.

Figura 6. Robot paralelo tipo Delta.

Debido a su excelente rendimiento dinámico, se utiliza comúnmente para ejecutar operaciones de Pick-and-place en líneas de producción. Se trata de una de las operaciones clave del ensamblaje programable, en la que los elementos se recogen y se colocan en sus posiciones respectivas de forma selectiva.

Este tipo de robot paralelo puede aplicarse a diversas áreas, como la paletización, el almacenamiento, la carga y descarga de máquinas, la alimentación de máquinas, la clasificación, la comprobación de placas de circuitos, la inspección y el mantenimiento remoto, y la cirugía robótica (Figura 7), en las industrias alimentaria, farmacéutica, biomédica, de embalaje, eléctrica y química [5], [6].

Figura 7. Robot Delta realizando trabajos de cirugía robótica.

3.2. Robótica y visión artificial

Para llevar a cabo correctamente sus aplicaciones, es común que estos robots contengan sistemas de visión artificial. Esto permite utilizar métodos inteligentes y procesar y analizar imágenes de los productos utilizados. La utilización de sistemas de visión proporciona al robot la posibilidad de controlar automáticamente el actuador final para agarrar con precisión los objetos a manipular. El establecimiento de la relación entre las características de la imagen y el espacio de trabajo del robot forma una cadena en bucle cerrado para enlazar las coordenadas del robot, las coordenadas de la cámara y las coordenadas de los objetos [7].

Los estudios realizados para la implementación de sistemas de visión en la robótica están enfocados simplemente en reconocer un objeto en el espacio de trabajo del robot (siendo todos los objetos iguales y teniendo que extraer todos) y en obtener las coordenadas del mismo, Figura 8, [5], [7] o, a lo sumo, en lograr el reconocimiento y clasificación de diferentes tipos de objetos que se encuentren en el espacio de trabajo del robot según su forma y color, Figura 9, [8], para

así poder seleccionar el objeto que sea necesario. En ambos casos, el objeto seleccionado será posteriormente manipulado por el robot en cuestión.

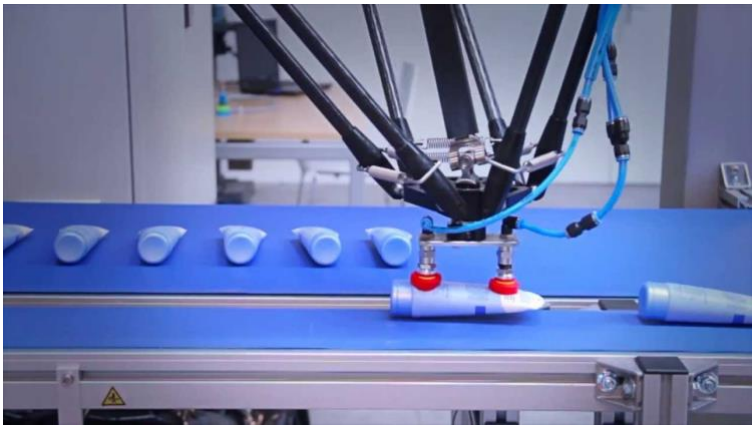


Figura 8. Reconocimiento y extracción de todos los objetos ubicados en el espacio de trabajo del robot.

Figura 9. Reconocimiento y clasificación de los objetos ubicados en el espacio de trabajo del robot.

3.3. Técnicas inteligentes de aprendizaje para el reconocimiento de caracteres

Son numerosos los estudios realizados respecto al reconocimiento de caracteres, utilizando para ello técnicas inteligentes de aprendizaje [9]. Entre otros, se han realizado estudios para el reconocimiento de caracteres chinos mediante redes neuronales convolucionales [10], y mediante aprendizaje profundo por refuerzo [11], reconocimiento de caracteres escritos en Braille [12] y de matrículas multinacionales [13] mediante redes neuronales convolucionales, en la Figura 10 se muestran un ejemplo de las predicciones obtenidas con la red neuronal al estudiar matriculas de diferentes países.

Figura 10. Resultados de la predicción de matrículas multinacionales.

También se han realizado un gran número de estudios para realizar el reconocimiento de los gestos de las manos, para desarrollar aplicaciones que permitan detectar los signos de las manos de las personas sordas y de las personas que utilizan el lenguaje de signos para comunicarse. El funcionamiento de estas aplicaciones se basa en detectar los signos realizados con las manos y posteriormente predecir la siguiente palabra o recomendar la palabra más adecuada. En concreto, en [14] se ha utilizado una CNN con varias capas de convolución, capturando características de las imágenes capa a capa, y utilizando dichas características para el entrenamiento del modelo. La solución propuesta en este artículo ha alcanzado una precisión de predicción del 91,07%.

Para trabajar con el reconocimiento de caracteres existe un set de datos llamado EMNIST (Extended MNIST) que consiste en 814.255 caracteres y dígitos manuscritos por 3.700 escritores diferentes, de los cuales 697,932 se utilizan para entrenamiento y 116.323 para test [15]. Este dataset se ha utilizado para diferentes técnicas inteligentes de aprendizaje, como por ejemplo redes neuronales profundas convolucionales (DCNN) [16] y técnicas de aprendizaje no supervisado [17].

Capítulo 4

Análisis de alternativas

4. Análisis de alternativas

En este apartado se realiza un estudio que permite comparar las alternativas existentes en cuanto a tecnologías y softwares de IA, así como definir los criterios para seleccionar la opción más adecuada para esta aplicación.

4.1. Tecnología de IA seleccionada

En este apartado se analizan dos tipos de algoritmos de Inteligencia Artificial que pueden ser utilizados para el reconocimiento de caracteres según la revisión de la tecnología realizada en el capítulo anterior: Redes Neuronales y *Support Vector Machines* (SVM).

Ambos algoritmos se basan en el aprendizaje supervisado, un tipo de aprendizaje en el que se trabaja con conjuntos de datos etiquetados. En este tipo de aprendizaje el modelo debe aprender a relacionar las entradas a las salidas correctas, realizando para ello un ajuste de sus parámetros internos a través de técnicas de optimización y minimización de error entre las predicciones y las respuestas esperadas. Cuando el modelo está entrenado, éste puede realizar predicciones precisas en nuevos datos no etiquetados.

Por un lado, las redes neuronales, Figura 11,están inspiradas en el funcionamiento del cerebro humano y se basan en un conjunto de algoritmos que imitan la forma en que las neuronas procesan y comunican la información. Estas redes consisten en capas de nodos interconectados llamados neuronas artificiales o unidades que trabajan juntas para aprender y extraer patrones complejos de los datos de entrada.

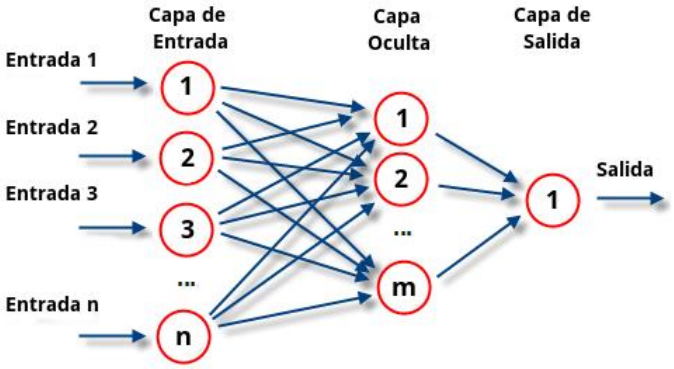


Figura 11. Ejemplo de Red Neuronal [18].

Las SVM (Figura 12), por otro lado, trabajan con el objetivo de encontrar un hiperplano o conjunto de hiperplanos óptimos que mejor separen las muestras de diferentes clases en un espacio de características. Tratan de maximizar el margen entre las muestras más cercanas a la frontera de decisión, lo que garantiza una mejor generalización en datos no vistos.

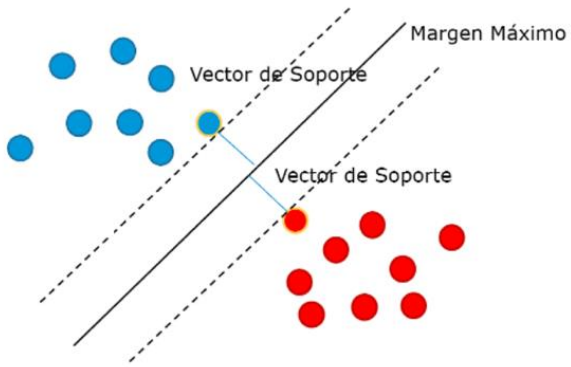


Figura 12. Ejemplo de SVM [19].

En la Tabla 1 se recogen las características extraídas de la literatura relativas a cada tipo de algoritmo:

Tabla 1. Comparativa entre Red Neuronal y Supper Vector Machines.

	Complejidad	Flexibilidad	Coste computacional
Red Neuronal	Media	Alta	Alto
SVM	Alta	Baja	Alto

Tras la comparación de ambos algoritmos se ha optado por utilizar Redes Neuronales para el reconocimiento de los caracteres de la aplicación.

4.2. Software de IA

En este apartado se han evaluado las principales diferencias entre el lenguaje de programación Python y el entorno de programación MATLAB, dos de los más populares a la hora de trabajar con Inteligencia Artificial, esta comparación se recoge en la Tabla 2.

Tabla 2. Comparativa entre MATLAB y Python.

	Complejidad	Bibliotecas y código abierto	Rendimiento
MATLAB	Alta	Pocas	Alto
Python	Media	Muchas	Alto

Tras comparar ambos softwares, se ha optado por trabajar con el lenguaje de programación Python.

Capítulo 5

Diseño y desarrollo de solución robótica para el reconocimiento de caracteres

5. Diseño y desarrollo de solución robótica para el reconocimiento de caracteres

En primer lugar es necesario comenzar actualizando el espacio de trabajo del robot, para una mejor adaptación a la aplicación. Se continúa diseñando el sistema inteligente de reconocimiento, siendo necesario para ello la selección del conjunto de datos a utilizar, y el diseño, entrenamiento y validación de la red neuronal. Después, se implementa el sistema inteligente a la aplicación robotizada, realizando para ello primero la captura y el preprocesamiento de la imagen, y después la predicción de las letras gestionando la comunicación de la herramienta del robot con MATLAB y Python. También es necesario verificar que la palabra pueda crearse y obtener las coordenadas de las letras a manipular. Por último se implementa una nueva funcionalidad en la aplicación de usuario y se analizan los resultados obtenidos.

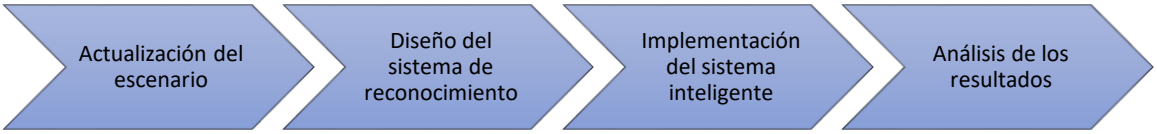


Figura 13. Esquema de los pasos realizados para el desarrollo de la solución.

En los siguientes apartados se explica detalladamente cada uno de los bloques expuestos en la Figura 13.

5.1. Actualización del escenario

Como se ha comentado anteriormente, dos de los mayores beneficios de la robótica paralela son la rapidez y precisión que pueden alcanzar los robots con esta estructura. Pero, para lograr dichos beneficios, es necesario que el escenario del robot esté en buenas condiciones.

Al comenzar a trabajar con el robot, se ha observado que la pinza utilizada sufre deformaciones, fruto del deterioro, tras realizar varias repeticiones. Como consecuencia de estas deformaciones, no se consigue un agarre correcto de las piezas a manipular, llegando en ciertos casos a desprenderse la pieza en movimiento, y, perdiendo la precisión debido a la holgura que presenta la pinza. Además, para tratar de prevenir lo recientemente explicado, se recomienda trabajar a velocidades reducidas. Por lo tanto, con esta pinza no se dispone de los beneficios que aporta la robótica paralela.

Tras analizar los problemas expuestos, se ha decidido reemplazar la pinza del robot. Se ha optado por la pinza paralela DHPS [21] del fabricante FESTO para reemplazar la garra del robot. Esta elección se basa en varias razones, que incluyen su elevada fuerza de sujeción, su precisión de repetición máxima, su capacidad de centrado de mordazas y su aseguramiento de fuerza de sujeción. Además, es importante destacar que esta pinza cuenta con una guía en T, ya que el mayor deterioro que ha sufrido la pinza reemplazada ha sido en la guía, la cual era de bolas. En la Figura 14 se muestra la pinza seleccionada.

Figura 14. Pinzas paralelas DHPS.

Una vez seleccionada la pinza hay que elegir el modelo, y, tras estudiar las características de cada uno, se ha optado por el modelo DHPS-16-A-NO [22]. Los aspectos a destacar de este modelo son el recorrido entre mordazas (10 mm) y la fuerza máxima admisible (150 N). Los modelos inferiores tienen estos parámetros más reducidos, perdiendo por lo tanto versatilidad, y, los superiores, en cambio, ofrecen mayor recorrido y fuerza máxima, pero para la aplicación en cuestión no son necesarios valores tan altos.

Para adecuar la nueva pinza al robot y a la aplicación se han realizado dos diseños a través del software de diseño Fusion 360. El primero, denominado "Adaptador Nuevo", Figura 15, se utiliza para enganchar la pinza de forma firme a la plataforma móvil del robot, asegurando así su correcta colocación. En el Anexo I se puede observar el plano de la pieza diseñada.

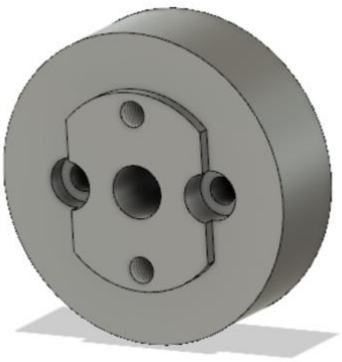


Figura 15. Adaptador Nuevo.

El segundo diseño, denominado "Dedos Pinza Nueva" consta de dos piezas o dedos, cada dedo se unirá a una mordaza de la pinza. Tras varios diseños preliminares, en la Figura 16 se muestra la pieza final diseñada. El diseño de estas piezas se ha realizado teniendo en cuenta el rango de apertura necesario para que el agarre de los cubos a manipular sea correcto,

concretamente, al colocar los dedos y cerrar la pinza se obtiene una anchura de 26,6 mm, pero, en la superficie que contacta con los cubos se coloca una membrana de neopreno de aproximadamente 1 mm de anchura, por lo que se disminuye la apertura a 24,6 mm. La colocación de la membrana aporta ventajas en cuanto a agarre (funciona como antideslizante) y durabilidad de la pieza, ya que absorbe parte de la fuerza ejercida en el agarre de los cubos, así como posibles impactos. Los planos de los dedos también pueden observarse en el Anexo I.

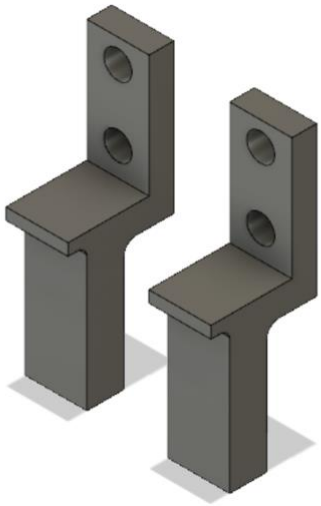


Figura 16. Dedos Pinza Nueva.

Al integrar en el robot la pinza DHPS y las piezas diseñadas, se obtiene un escenario robótico adecuado para abordar el reconocimiento y manipulación de los caracteres. En la Figura 17 se muestra el resultado de la modificación del escenario.

Figura 17. Escenario robótico tras la integración de la pinza DHPS y las piezas diseñadas.

5.2. Diseño del sistema inteligente de reconocimiento

Para llevar a cabo el reconocimiento de caracteres, de la revisión de la literatura realizada sobre técnicas de IA, se extrae que las redes neuronales convolucionales son las más adecuadas para la aplicación actual, debido fundamentalmente a su amplia flexibilidad. Para lograr un correcto funcionamiento de la red es necesario escoger un conjunto de datos correcto con el que se realizará el entrenamiento de la red, además de diseñar la red (escoger los hiperparámetros a utilizar), entrenarla con el conjunto de datos previamente escogido y finalmente validarla. En la Figura 18 se resumen los pasos a realizar para obtener una red neuronal apta para lograr el reconocimiento de los caracteres.

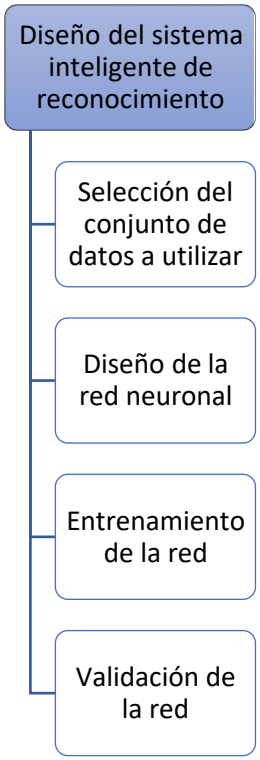


Figura 18. Pasos a realizar para el diseño y desarrollo del sistema inteligente de reconocimiento.

5.2.1. Selección del conjunto de datos a utilizar

Los conjuntos de datos desempeñan un papel muy importante en el desarrollo y entrenamiento de modelos de redes neuronales en el campo de aprendizaje automático y la visión por computadora. Uno de los conjuntos de datos ampliamente utilizados es EMNIST.

El conjunto de datos EMNIST (Extended Modified National Institute of Standards and Technology) es una ampliación del conjunto de datos MNIST. La diferencia entre los dos conjuntos de datos nombrados es que MNIST solamente contiene imágenes de dígitos escritos a mano, mientras que EMNIST recoge también las letras del alfabeto en mayúsculas y minúsculas, Figura 19.

Figura 19. Ejemplos de letras y dígitos del conjunto de datos EMNIST [15].

El conjunto de datos EMNIST está compuesto por un total de 814,255 caracteres y dígitos manuscritos por 3.700 escritores diferentes, de los cuales 697.932 se utilizan para entrenamiento y 116.323 para test. El conjunto de datos EMNIST está a su vez formado por 6 conjuntos de datos diferentes, cada uno con diferente número de muestras y de clases de muestra. En la Figura 20 se muestran los parámetros recientemente nombrados para cada conjunto de datos EMNIST, además, también pueden observarse las divisiones de entrenamiento y prueba de cada conjunto.

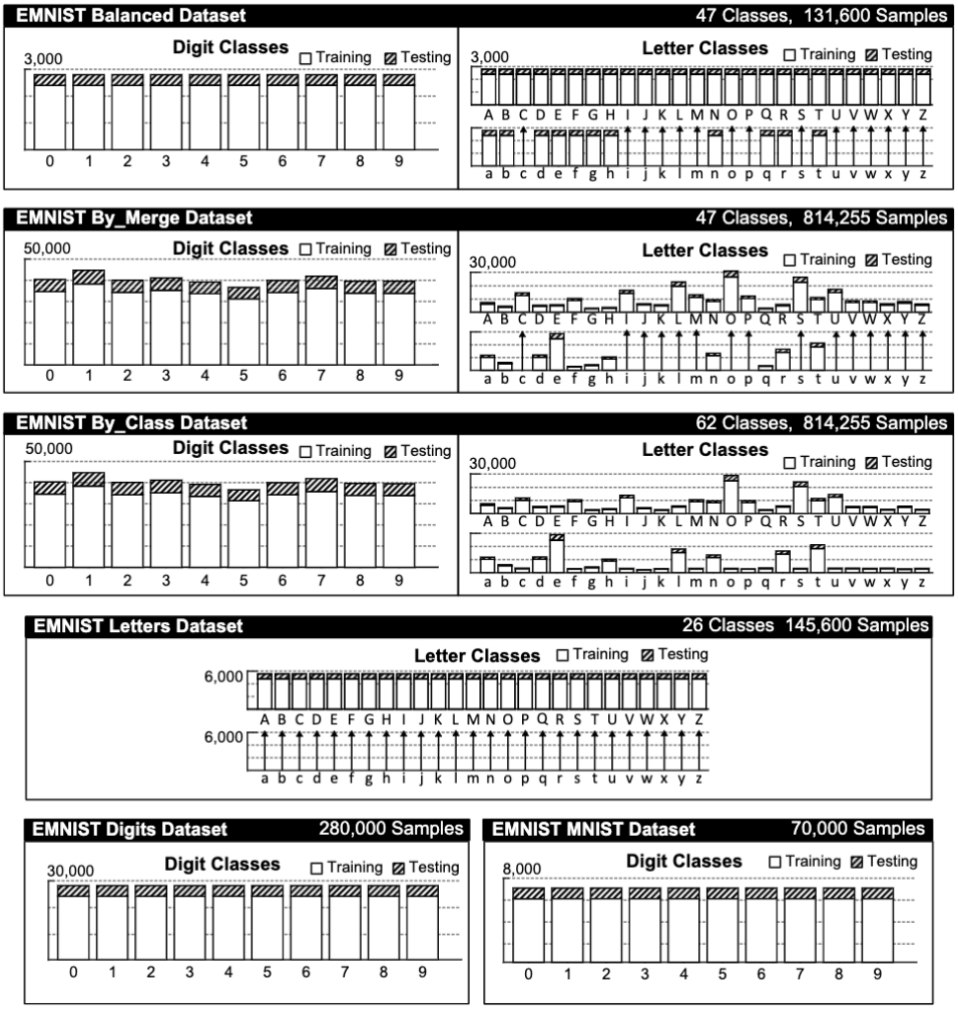


Figura 20. Desglose de los conjuntos de datos EMNIST [23].

Tras analizar los diferentes conjuntos de datos EMNIST disponibles, se escogió el que mejor se adaptaba a la aplicación: el conjunto de datos EMNIST Letters. Este conjunto de datos está formado por 145.600 imágenes en escala de grises, distribuidas en 26 clases de letras mayúsculas y minúsculas del alfabeto. Cada imagen tiene un tamaño de 28x28 píxeles, lo que hace un total de 784 características.

En el Anexo II.I – Código para la preparación de las variables de la red se muestra el código de Python utilizado para la preparación del conjunto de datos.

5.2.2. Diseño de la red neuronal

Como se ha comentado anteriormente, se ha optado por la implementación de una Red Neuronal Convolucional para conseguir realizar el reconocimiento de los caracteres impresos en los cubos ubicados en el espacio de trabajo del robot.

El modelo de Red Neuronal Convolucional se ha creado a través de la biblioteca de Redes Neuronales de Código Abierto Keras, la cual está escrita en Python. En este caso, se ha utilizado la clase secuencial.

El modelo creado está formado por 8 capas, una capa de entrada, una capa de salida y 6 capas intermedias: la capa de entrada recoge la entrada y la utiliza en el proceso de cálculo, las capas ocultas recogen la entrada de la capa anterior y generan la salida para la siguiente capa, y la capa de salida predice el resultado.

La primera capa, la de entrada, es de convolución ("Conv2D()"), y, al ser la primera, se debe identificar el formato de entrada, el cuál es de 28x28x1 debido a que las imágenes obtenidas para la realización de la predicción son imágenes de 28x28 píxeles y de un único canal (imágenes en escala de grises). Esta capa cuenta con un espacio de salida ("filters") de 128, una ventana de convolución ("kernel_size") de 5x5 y la función de activación ("activation") "ReLU", esta es una función que genera una salida igual a cero cuando la entrada es negativa, y una salida igual a la entrada cuando esta es positiva.

La segunda capa de la Red Neuronal, es una capa de pooling ("MaxPooling2D()"), estas capas permiten reducir el peso de la representación para así agilizar el proceso de aprendizaje de la red neuronal. Esta capa asegura que los patrones detectados en la capa convolucional se mantenga en la siguiente capa de la red, además, mediante este tipo de capas también se reduce el sobreajuste y se consigue aumentar la abstracción sobre los datos de entrada. El tamaño de ventana ("pool_size") utilizado para obtener el número máximo es de 2x2. El valor de zancada ("stride"), valor que especifica como de lejos se mueve la ventana de agrupación para cada paso de agrupación también es de 2x2.

La tercera capa vuelve a ser de convolución, en este caso, con un espacio de salida de 64 y una ventana de convolución de 3x3, la función de activación utilizada sigue siendo "ReLU".

A continuación, en la cuarta capa, se vuelve a aplicar una ventana de max pooling de 2x2. El valor de zancada también es de 2x2.

La quinta capa, es una capa utilizada para aplanar las salidas de la fase convolucional ("flatten()"). Las imágenes son de dos dimensiones, por lo que se utiliza esta capa para convertir los datos en un vector para preparar la entrada para las capas densas.

La sexta capa es utilizada para conectar las capas densamente ("Dense()"), esto significa que todas las neuronas de la capa están conectadas a todas las neuronas de la siguiente capa. En esta función se puede elegir el número de neuronas a utilizar mediante el parámetro "units" en el caso de esta capa se ha decidido utilizar 128 neuronas, con la función de activación "ReLU".

La séptima capa (“Dropout()”) es denominada capa de abandono, ya que establece aleatoriamente las unidades de entrada en 0 con una frecuencia de “rate” en cada paso durante el tiempo de entrenamiento, lo que ayuda a evitar el sobreajuste. El parámetro “rate” establece la fracción de las unidades de entrada que hay que dejar caer, y puede tener valores de entre 0 y 1. En este caso el valor asignado para este parámetro es de 0.5, por lo que elimina aleatoriamente el 50% de las neuronas de la capa anterior.

Por último, la capa de salida vuelve a ser una capa de conexión densa de neuronas, en este caso, el número de neuronas se ha establecido igual al número de letras a predecir (26), y la función de activación utilizada en este caso es la denominada “softmax”, esta función es comúnmente utilizada en la capa de salida de las redes neuronales utilizadas para realizar la clasificación multiclase. La función se utiliza para convertir las salidas de la capa anterior en probabilidades que suman uno, de esta forma se puede medir la confianza del modelo en la pertenencia a cada clase.

En la Figura 21 se puede observar de forma esquemática la estructura de la Red Neuronal Convolucional recientemente explicada.

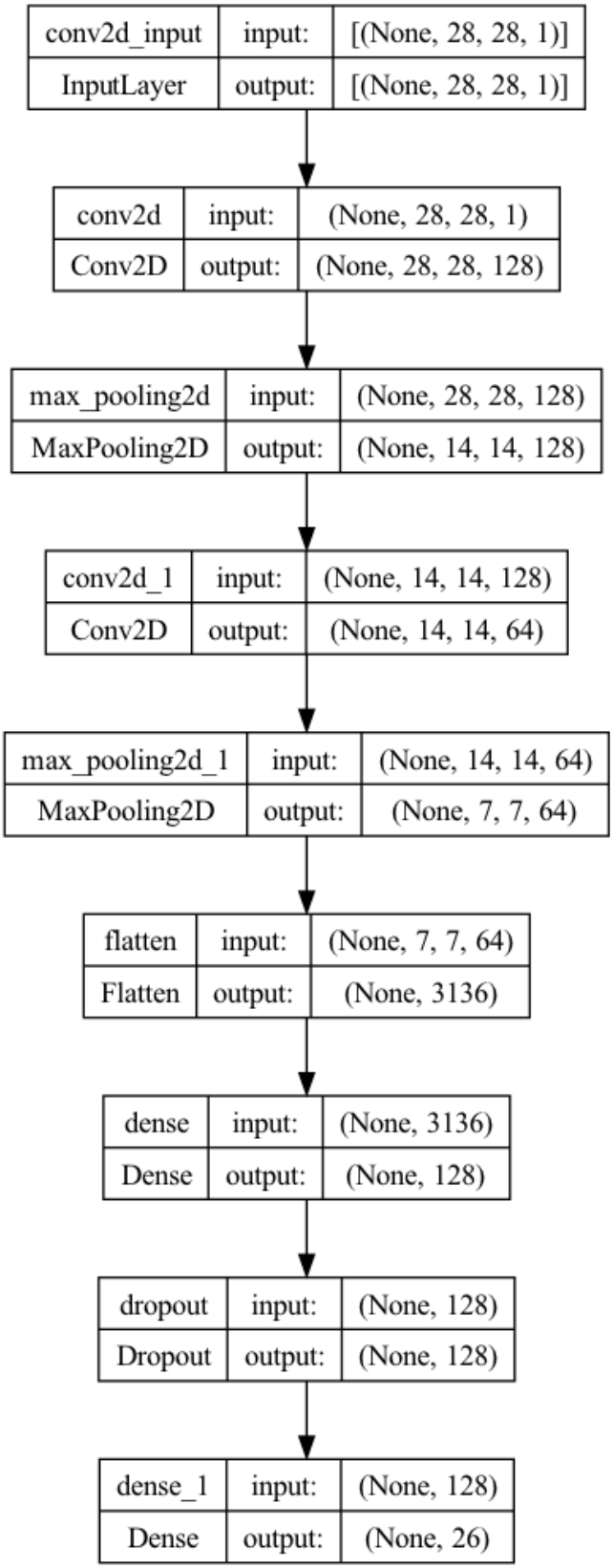


Figura 21. Estructura de la Red Neuronal Convolucional utilizada.

5.2.3. Entrenamiento de la Red

Una vez definido el modelo a utilizar, se ha de especificar cómo será su proceso de aprendizaje, esto se realiza mediante la función "compile()", en donde se deben seleccionar los argumentos de función de pérdida, optimizador y métrica:

- Función de pérdida: esta función se utiliza para medir la diferencia entre la salida real y la salida predicha por la red, y la reduce a un solo valor que se utiliza para actualizar los pesos de la red. En este caso, se ha utilizado la función de pérdida entropía cruzada categórica, utilizada comúnmente en problemas de clasificación multiclase.
- Optimizador: es el algoritmo optimizador que se utiliza para actualizar los pesos de la red durante el entrenamiento. En este caso se utiliza el optimizador Adam.
- Métrica: se utiliza para evaluar el rendimiento de la red neuronal durante el entrenamiento. En este caso se ha elegido la métrica de precisión.

Una vez especificado el proceso de aprendizaje se procede a entrenar el modelo, para ello se ha decidido entrenar durante 50 épocas con un tamaño de lote de 512. Esto puede parecer exagerado para la aplicación en cuestión, pero, se han definido tres objetos "callbacks" para el entrenamiento:

- ModelCheckPoint: guarda los mejores puntos del modelo durante el entrenamiento.
- EarlyStopping: detiene el entrenamiento si no se observa una mejora en la precisión de validación después de cierto número de épocas.
- ReduceLROnPlateau: reduce la tasa de aprendizaje si no se observa una mejora en la pérdida de validación después de cierto número de épocas.

En el Anexo II.II – Código para el diseño, entrenamiento y generación del modelo se muestra el código de Python utilizado para realizar lo recientemente explicado en los dos últimos subapartados: el diseño, entrenamiento y generación del modelo.

5.2.4. Validación de la red

Como consecuencia de estos "callbacks" el entrenamiento ha finalizado tras 17 épocas. En la Figura 22, en la imagen de la izquierda, se muestra la comparación entre las curvas de precisión de validación y de entrenamiento del modelo obtenido. En esa imagen se puede observar que a partir de la época 15 se supera el 90% de precisión de validación, obteniendo un valor final en la época 17 de 92,87%. Y, en la imagen de la derecha, se realiza una comparación del error de validación y de entrenamiento, obteniendo un error de validación final de 2,39%.

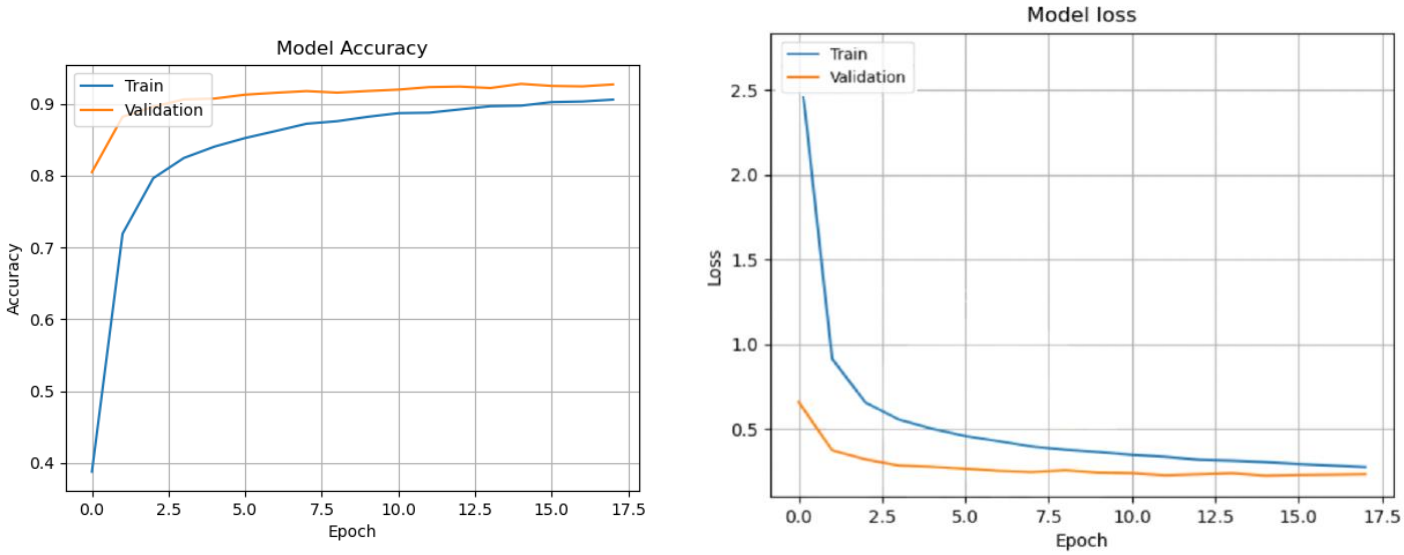


Figura 22. a) Precisión de validación y entrenamiento del modelo. b) Error de validación y entrenamiento del modelo.

5.3. Implementación del sistema inteligente en la aplicación robotizada

Una vez obtenido un buen modelo de red neuronal para efectuar la tarea de reconocimiento de caracteres, se comienza con la implementación del mismo a la aplicación robotizada. Para ello es necesario realizar una correcta captura de imagen, procesar la misma adecuadamente, preparar la entrada de la red neuronal, y conseguir comunicar las tres herramientas: Sysmac Studio, MATLAB y Python. En la Figura 23 se recogen los pasos a realizar para lograr la implementación del sistema inteligente en la aplicación robotizada.

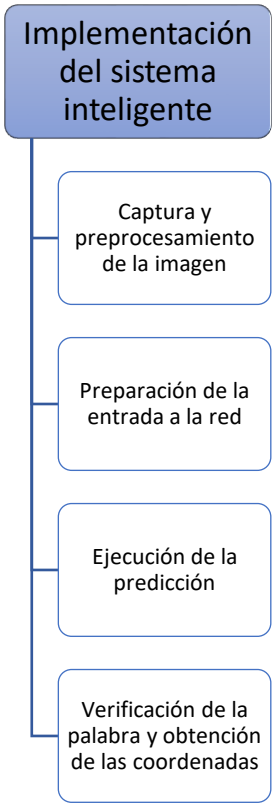


Figura 23. Pasos a realizar para la implementación del sistema inteligente.

En los siguientes apartados se complementan las explicaciones con secciones del código de MATLAB empleado para lograr la implementación.

5.3.1. Captura y preprocesamiento de la imagen

La captura de la imagen se realiza a través de la webcam Logitech Brio 4k UltraHD ubicada en el escenario de la célula robotizada, y la orden de captura se envía a través de MATLAB, mediante el comando *snapshot*. En la Figura 24 se muestra un ejemplo de cómo es la imagen capturada a través de MATLAB.

Figura 24. Imagen capturada a través de la webcam.

Como se puede ver en la anterior figura, la imagen se obtiene en perspectiva, debido a que por la disposición del escenario del robot no es posible obtener una imagen en vertical al centro de la plataforma. Al obtener la imagen de forma oblicua ésta sufre una distorsión, la cual se va a compensar enderezando la imagen a través de la aplicación de una transformada:

```
input_points=[2.349999999999999e+02,68;559,68;591,313;192,297];
base_points= [0 0; 500 0; 500 250;0 250];
t_entorno=cp2tform(input_points, base_points, 'projective');
recta = imtransform(fotopanel,t_entorno);
```

Primero se definen las coordenadas de entrada (input_points) y se establecen las dimensiones predefinidas de escalado (base_points), a continuación, a través del comando *cp2tform* se crea la estructura de la transformación (en este caso observando la perspectiva de la imagen se ha optado por proyectiva). Por último, se aplica la transformada para conseguir enderezar la imagen, obteniendo el resultado mostrado en la Figura 25.

Figura 25. Imagen obtenida tras aplicar la transformada.

Una vez enderezada la imagen, se recorta para eliminar todo lo que no tenga que ver con los cubos, es decir, se eliminan las zonas de la imagen donde se observa el fondo, obteniendo el resultado mostrado en la Figura 26.

Figura 26. Imagen recortada.

Se continua aplicando diferentes técnicas de preprocesamiento para lograr que las letras se observen lo mejor posible, lo cual mejorará la futura predicción del modelo. Primero, se transforma la imagen de RGB a escala de grises, y, después, se binariza con un umbral de 0.79. En la Figura 27 se puede observar la evolución de la imagen tras los cambios realizados.

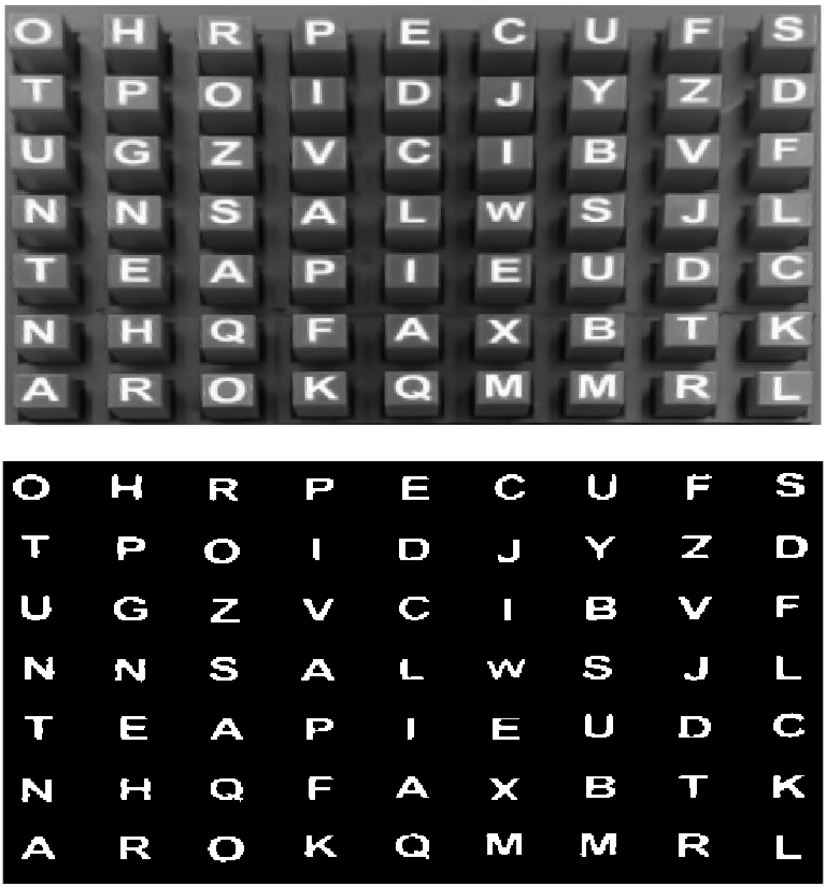


Figura 27. a) Imagen en escala de grises. b) Imagen en BW.

5.3.2. Preparación de la entrada a la red

Como se ha explicado anteriormente, la red neuronal se entrena utilizando imágenes de 28x28 píxeles de letras manuscritas, es decir, en cada imagen solo aparece una letra. Concretamente, cada imagen representa una fila de un archivo ".csv", dicha fila tiene una longitud de 784 (cantidad total de píxeles, 28x28 = 784). Por esto, es necesario segmentar la imagen de la Figura 27, la imagen en blanco y negro, para obtener 63 imágenes independientes, una para cada letra. Y, después, convertir cada imagen, que será una matriz de 28x28, en un vector de 1x784. Cabe destacar también que en las imágenes utilizadas las letras se encuentran centradas. A continuación se explica detalladamente el procedimiento seguido para obtener unos datos adecuados que pasarle a la red como entrada.

Para realizar la segmentación de las letras se valoran dos alternativas:

Alternativa 1: Crear un recuadrado de dimensiones fijas en una esquina de la imagen e ir moviéndolo una distancia predefinida 9 veces por fila, es decir, pasar por todas las columnas de una misma fila, y, una vez llegado a la última columna, pasar a la siguiente fila.

Esta forma de segmentar la imagen en 63 imágenes más pequeñas parece sencilla, pero cuenta con una complicación, las letras no están perfectamente alineadas, ni vertical ni horizontalmente, ver Figura 28, por lo que en ciertos casos las letras no quedan centradas en la imagen.

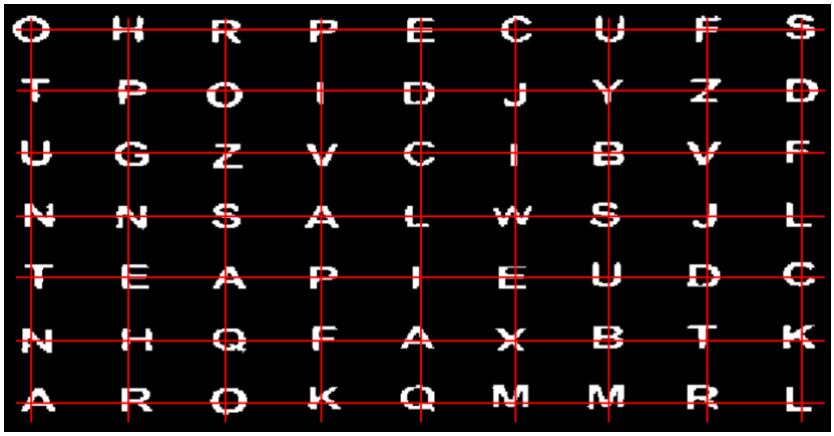


Figura 28. Demostración de que las letras no están perfectamente alineadas entre sí.

Alternativa 2: Valernos de la función *regionprops* para encontrar el centroide de cada letra y a partir de ahí realizar el recorte de las mismas.

Tras probar ambas alternativas se ha observado que se obtienen mejores resultados con la segunda, dado que con la primera algunas letras quedan seccionadas, dificultando la predicción de la red. A continuación se explica detalladamente el procedimiento seguido para recortas las letras utilizando la técnica de los centroides:

Se comienza midiendo las propiedades de centroide, cuadro delimitador y área de cada uno de los objetos de la imagen mediante la función *regionprops*.

Con los centroides se obtienen las coordenadas horizontales (coordenada x) y verticales (coordenada y) de cada centro de masas de la región, Figura 29.

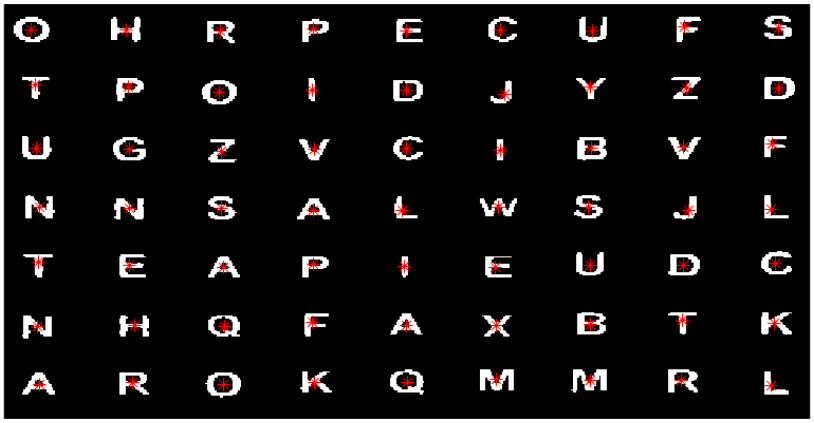


Figura 29. Representación de los centroides de cada región.

Los cuadros delimitadores especifican la posición y tamaño del cuadro más pequeño que contiene la región, Figura 30, y el área devuelve un escalar con el número real de píxeles en la región.

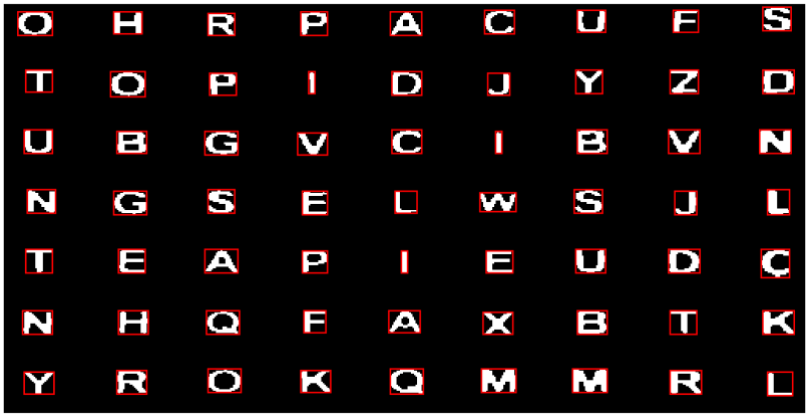
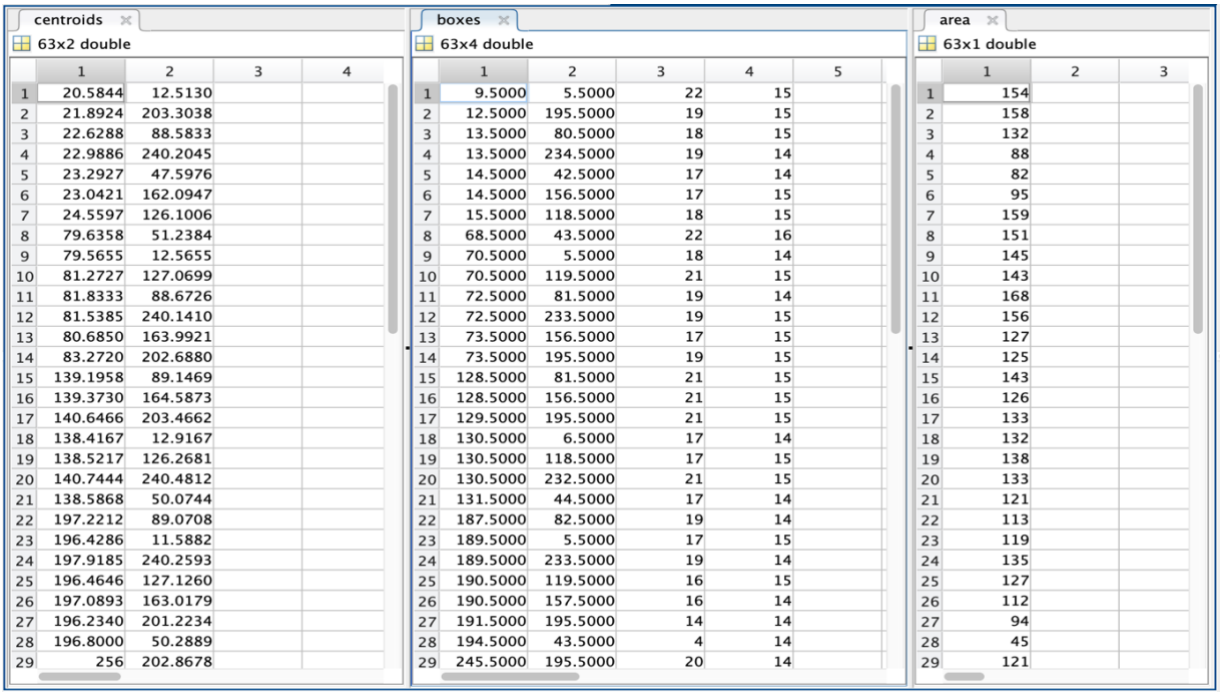


Figura 30. Representación de los cuadros delimitadores de cada región.

En la Figura 31 se muestran los valores de las tres propiedades estudiadas para los 29 primeros objetos encontrados.



centroids — 63x2 double

	1	2	3	4
1	20.5844	12.5130		
2	21.8924	203.3038		
3	22.6288	88.5833		
4	22.9886	240.2045		
5	23.2927	47.5976		
6	23.0421	162.0947		
7	24.5597	126.1006		
8	79.6358	51.2384		
9	79.5655	12.5655		
10	81.2727	127.0699		
11	81.8333	88.6726		
12	81.5385	240.1410		
13	80.6850	163.9921		
14	83.2720	202.6880		
15	139.1958	89.1469		
16	139.3730	164.5873		
17	140.6466	203.4662		
18	138.4167	12.9167		
19	138.5217	126.2681		
20	140.7444	240.4812		
21	138.5868	50.0744		
22	197.2212	89.0708		
23	196.4286	11.5882		
24	197.9185	240.2593		
25	196.4646	127.1260		
26	197.0893	163.0179		
27	196.2340	201.2234		
28	196.8000	50.2889		
29	256	202.8678		

boxes — 63x4 double

	1	2	3	4	5
1	9.5000	5.5000	22	15	
2	12.5000	195.5000	19	15	
3	13.5000	80.5000	18	15	
4	13.5000	234.5000	19	14	
5	14.5000	42.5000	17	14	
6	14.5000	156.5000	17	15	
7	15.5000	118.5000	18	15	
8	68.5000	43.5000	22	16	
9	70.5000	5.5000	18	14	
10	70.5000	119.5000	21	15	
11	72.5000	81.5000	19	14	
12	72.5000	233.5000	19	15	
13	73.5000	156.5000	17	15	
14	73.5000	195.5000	19	15	
15	128.5000	81.5000	21	15	
16	128.5000	156.5000	21	15	
17	129.5000	195.5000	21	15	
18	130.5000	6.5000	17	14	
19	130.5000	118.5000	17	15	
20	130.5000	232.5000	21	15	
21	131.5000	44.5000	17	14	
22	187.5000	82.5000	19	14	
23	189.5000	5.5000	17	15	
24	189.5000	233.5000	19	14	
25	190.5000	119.5000	16	15	
26	190.5000	157.5000	16	14	
27	191.5000	195.5000	14	14	
28	194.5000	43.5000	4	14	
29	245.5000	195.5000	20	14	

area — 63x1 double

	1	2	3
1	154		
2	158		
3	132		
4	88		
5	82		
6	95		
7	159		
8	151		
9	145		
10	143		
11	168		
12	156		
13	127		
14	125		
15	143		
16	126		
17	133		
18	132		
19	138		
20	133		
21	121		
22	113		
23	119		
24	135		
25	127		
26	112		
27	94		
28	45		
29	121		

Figura 31. Datos de las propiedades de centroide, cuadro delimitador y área de los objetos.

Una vez obtenidas las tres propiedades, se comienzan a estudiar todos los centroides uno a uno, y se evalúan en función del área que tengan:

- Si el área es menor a 20 el centroide será rechazado, se considera ruido.
- Si el área es entre 20 y 60 se recorta un cuadrado cuyas dimensiones serán las del cuadro delimitador + [-10 -10 20 20].
- Si el área es mayor a 60 se recorta un cuadrado cuyas dimensiones serán las del cuadro delimitador + [-5 -5 10 10].

Esta evaluación se debe a que las letras como la I o la J suelen tener un área de entre 20 y 60 píxeles, y su cuadrado delimitador es muy pequeño, por ello, en estos casos se recorta un recuadro más grande. Por otro lado, tras realizar diferentes pruebas, se ha comprobado que todas las regiones con área inferior a 20 son ruido, por lo que son rechazadas. Las imágenes recortadas no son todas de las mismas dimensiones, depende del tamaño del cuadrado delimitador de cada una, por ello, tras recortarlas se les aplica un reescalado, convirtiendo todas las imágenes en matrices de 28x28 píxeles.

Las imágenes recortadas se guardan nombrándolas con el número correspondiente a su posición, para así saber en todo momento la ubicación que tiene la letra que se está estudiando en la plataforma grande. Siendo la numeración elegida la siguiente:

Tabla 3. Numeración de las letras de la plataforma grande.

1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18
19	20	21	22	23	24	25	26	27
28	29	30	31	32	33	34	35	36
37	38	39	40	41	42	43	44	45
46	47	48	49	50	51	52	53	54
55	56	57	58	59	60	61	62	63

Un problema encontrado al trabajar con el corte a través de centroides es que estos no se detectan en el orden especificado en la Tabla 3, por lo tanto, las imágenes de las letras se recortan en un orden desconocido, y se pierde la información de qué ubicación tiene cada letra en la plataforma grande. Se ha creado un algoritmo para solventar este problema:

Se analizan los valores de los centroides en coordenadas x e y, y sacando un valor aproximado para cada columna y fila:

Tabla 4. Coordenadas aproximadas de cada columna de la plataforma grande.

Columna	1	2	3	4	5	6	7	8	9
Valor	22	80	139	196	255	314	373	433	490

Tabla 5. Coordenadas aproximadas de cada fila de la plataforma grande.

Fila	Valor
1	12
2	50
3	88
4	126
5	164
6	202
7	240

Se crean dos vectores, uno, *vector_coordenadasx* con la información de la Tabla 4, y *vector_coordenadasy* con la información de la Tabla 5. Y una matriz *matriz_posiciones* con la información de la Tabla 3.

```
handles.vector_coordenadasx = [22 80 139 196 255 314 373 433 490];
handles.vector_coordenadasy = [12; 50; 88; 126; 164; 202; 240];

handles.matriz_posiciones = [1 2 3 4 5 6 7 8 9;
                             10 11 12 13 14 15 16 17 18;
                             19 20 21 22 23 24 25 26 27;
                             28 29 30 31 32 33 34 35 36;
                             37 38 39 40 41 42 43 44 45;
                             46 47 48 49 50 51 52 53 54;
                             55 56 57 58 59 60 61 62 63;]
```

Para obtener la posición del centroide, se crean dos bucles, un primero para conocer la columna a la que pertenece, y el segundo para conocer la fila. El modo de funcionamiento es el siguiente, para obtener la columna a la que pertenece cada centroide, se compra su coordenada x con las del *vector_coordenadax*, se realiza una resta entre ellas, y, en caso de que el resultado de la resta sea inferior a *valor* (en la primera comparación valor = 100) se guarda el número de la iteración en la variable *posx* y se actualiza la variable *valor*. Después se repite el mismo procedimiento con la coordenada y del centroide y con el vector *vector_coordenaday*, obteniendo así la fila del vector.

```
% Ordenar los centroides
valor=100;
for v1=1:length(handles.vector_coordenadasx) % para obtener la columna a la que pertenece el
centroide
 valornuevox=abs(coordenadax-handles.vector_coordenadasx(1,v1));
 if valornuevox<valor
   posx=v1;
   valor=valornuevox;
 end
end
valor=100;
for v2=1:length(handles.vector_coordenadasy) % para obtener la fila a la que pertenece el centroide
 valornuevoy=abs(coordenaday-handles.vector_coordenadasy(v2,1));
 if valornuevoy<valor
   posy=v2;
   valor=valornuevoy;
 end
```

```
end
```

Por lo tanto, tras ejecutar estos dos bucles, se tiene la información de la fila y columna a la que pertenece el centroide en las variables *posx* y *posy*. Con estas dos variables se conoce la ubicación del centroide en la plataforma grande, por ello, es posible obtener la numeración que le corresponde:

```
numcrop=handles.matriz_posiciones(posy,posx); % Se obtiene el numero correspondiente a la
posición del centroide
```

Como se ha comentado anteriormente, la entrada de la red neuronal es un archivo csv que contiene la información de las imágenes de todas las letras a predecir. Este archivo está compuesto por 63 filas, una por cada imagen, y 784 columnas, una por cada píxel de la imagen. El archivo csv se crea tras completar una matriz de dimensiones 63x784, la cual se va rellenando de forma iterativa una vez obtenida la posición de cada imagen en la plataforma grande, pero para ello, primero es necesario reescalar la imagen para convertirla en un vector de dimensiones 1x784.

```
let=reshape(croppedImage,[1,784]); % Convierto la matriz de 28x28 a un vector de 1x784
for columna=1:784
  matriz_csv(numcrop,columna)=let(columna);
end
```

Con la creación del archivo csv se obtiene la entrada para el modelo de la red neuronal. En el Anexo III se puede observar el código completo de lo recientemente explicado.

5.3.3. Ejecución de la predicción

Para realizar la predicción de las letras ubicadas en la plataforma grande, se llama desde MATLAB a la función *recognition* programada en el script *makeprediction*, Anexo II.III – Código para realizar las predicciones, de Python:

```
pyOut = py.makeprediction.recognition;
```

La función *recognition* comienza con la carga del modelo previamente generado, así como del subconjunto de datos desconocidos sobre los que se van a realizar las predicciones (archivo csv generado en MATLAB).

Al subconjunto de datos cargado se le aplica un preprocesamiento que consiste en reescalar la imagen, así como en rotar y voltearla para su correcta visualización.

Una vez preprocesado el subconjunto de datos se realiza la predicción, los valores predichos que devuelve el modelo son números enteros del 1 al 26, siendo 1=A, 2=B … 26=Z. Por ello,

se utiliza una lista creada a modo de diccionario, *label_dict*, que mapea los números con su correspondiente letra ASCII.

La letra ASCII correspondiente se guarda en la lista *predicted*, la cual una vez completada con las 63 predicciones es enviada a MATLAB, donde se convierte en string y se guarda en la variable *cadena_letras*.

5.3.4. Verificación de la palabra y obtención de las coordenadas

Es necesario corroborar que la palabra seleccionada por el usuario pueda ser escrita, es decir, que se cuente con las letras necesarias para su escritura. Para ello, se estudia la cantidad de cada tipo de letra predicha, y se compara con las letras necesarias para escribir la palabra. En caso de que no haya letras suficientes para su escritura (por ejemplo, que la palabra a escribir sea manzana y solo se hayan predicho 2 letras "a") se devuelve un mensaje en pantalla informando de que no es posible realizar la escritura de la palabra. Por el contrario, en caso de que se cuente con las letras necesarias para escribirla, se comienza con la obtención de las coordenadas de cada letra.

Se estudia por separado cada una de las letras que forman la palabra a escribir. Se compara cada letra con las del string *cadena_letras*, en caso de encontrar una coincidencia se sobreescribe un 0 en la posición del string donde esté ubicada la letra (para eliminar la letra seleccionada del string, así se evita que se trate de coger la misma letra más de una vez).

Gracias al trabajo previo realizado para ordenar las letras a predecir, conociendo el índice del string de la letra seleccionada es posible conocer la posición donde se ubica dicha letra en la plataforma grande. Por ejemplo, si el índice en el string es el 32, la posición en la que se ubica dicha letra será la 4º fila y 5º columna (Tabla 3).

Se han definido también dos matrices, tal y como se muestran a continuación, que cuentan con las coordenadas reales de cada posición de la plataforma grande:

```
%Coordenadas robot.
%Se genera la matriz de coordenadas x.
handles.coordenadas_robot_x=[158, 160, 161, 163, 164, 165, 167, 168, 169;
                             117, 118, 119, 121, 122, 123, 124, 125, 126;
                             73, 75, 77, 78, 79, 80, 81, 83, 84;
                             31, 33, 35, 36, 37, 38, 39, 41, 42;
                             -11, -10, -9, -8, -6, -5, -3.5, -2, -1;
                             -56, -54.5, -53, -52, -51, -50, -49, -48, -47;
                             -99, -97.5, -96, -95, -94, -93, -91.5, -89.5, -88.5];

%Se genera matriz de coordenadas y.
handles.coordenadas_robot_y=[175, 133, 90, 47, 4 -39, -82, -125, -169;
```

```
175, 132, 89, 46, 2 -41, -83, -126, -170;
173, 130, 87, 44, 1, -42, -84, -127, -170;
172, 130, 87, 44, 0, -43, -86, -129, -172;
171, 128, 85, 42, -1, -44, -87, -130, -173;
170, 127, 84, 41, -2, -45, -88, -131, -174;
168.5, 125, 82, 39, -4, -47, -89.5, -132, -175];
```

Al haber seguido el mismo orden que en la *matriz_posiciones*, conociendo la fila y columna en la que se encuentra la letra, podemos obtener las coordenadas x e y reales del cubo a manipular. Dichas coordenadas se guardan en el vector *vector_TCP*, una vez rellenado el vector, se realiza una comunicación a través de TCP/IP con el PLC que gobierna el robot, logrando así enviarle las coordenadas de los cubos a manipular.

5.3.5. Aplicación de usuario

En cuanto a la aplicación de usuario a utilizar, es la misma que la implementada en [1], pero se le añade una nueva funcionalidad: la posibilidad de abrir un panel, en el que se observan las imágenes obtenidas y preprocesadas por la cámara, cada una de ellas con su correspondiente predicción.

Dicho panel se denomina *Predicciones* y se abre al pulsar sobre el botón *Panel reconocimiento – IA*. En la Figura 32 se muestra un ejemplo de la aplicación con la nueva funcionalidad.

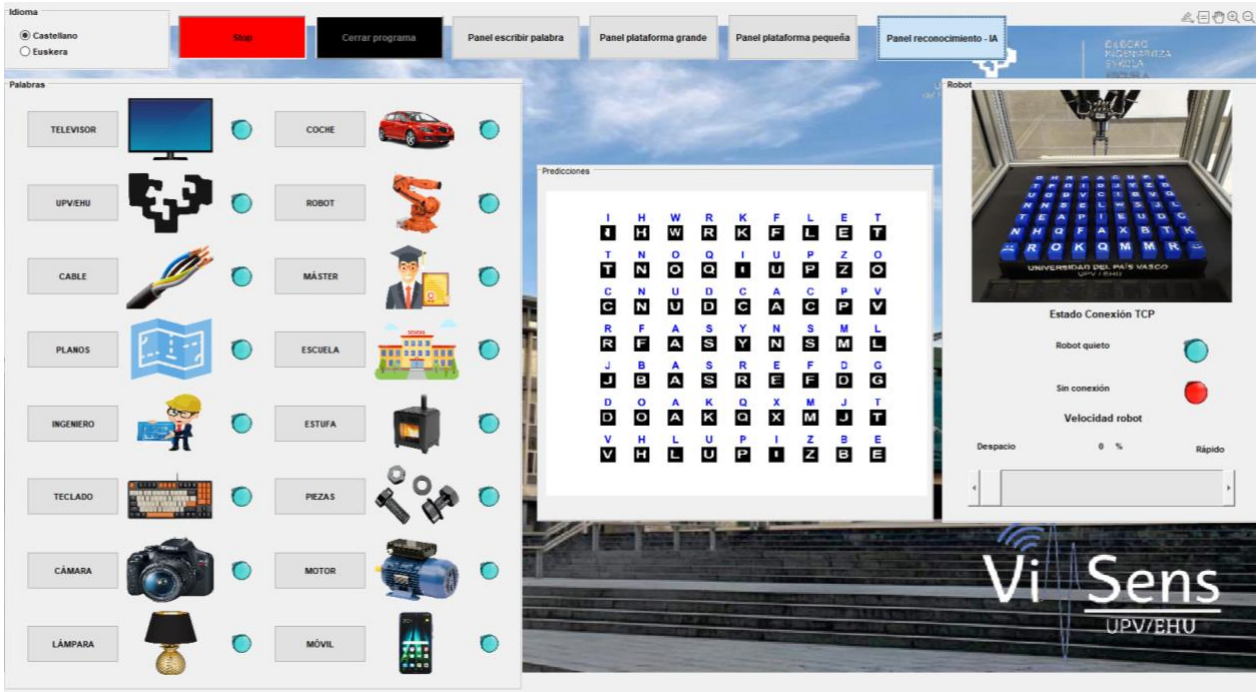


Figura 32. Aplicación de usuario con la nueva funcionalidad.

5.4. Análisis de los resultados

Para validar la correcta implementación de la solución, se realizan diferentes ensayos con objeto de estudiar la precisión y robustez de la aplicación.

Se realizan 6 pruebas diferentes: tres primeras con alta iluminación artificial (encendiendo los LEDs ubicados en el chasis del robot), en las que se modificará la ubicación de las letras en la plataforma grande. En estas tres primeras pruebas se quiere estudiar el porcentaje de acierto de la aplicación ante diferentes escenarios.

Después se realizan otras tres pruebas, esta vez con los LEDs apagados. Se realizan manteniendo la misma ubicación de caracteres que la utilizada en la 3º prueba, ya que el objetivo de estas tres pruebas es estudiar la afección de la incidencia de luz, por ello, las pruebas se han realizado en tres momentos diferentes del día: a las 17:00, a las 18:00 y a las 19:00, para estudiar si el cambio de iluminación afecta a la aplicación. En la Tabla 6 se recoge el resumen de las pruebas realizadas.

Tabla 6. Resumen de las pruebas realizadas.

Iluminación	N.º de prueba	Detalle de la prueba
Iluminación alta y constante (LEDs encendidos)	1º Prueba	Posicionamiento de letras 1
	2º Prueba	Posicionamiento de letras 2
	3º Prueba	Posicionamiento de letras 3
Iluminación variable (lámparas del techo del laboratorio y la luz natural entrante por las ventanas)	4º Prueba	Posicionamiento de letras 3 a las **17:00**
	5º Prueba	Posicionamiento de letras 3 a las **18:00**
	6º Prueba	Posicionamiento de letras 3 a las **19:00**

A continuación, se exponen los resultados obtenidos en cada una de las pruebas.

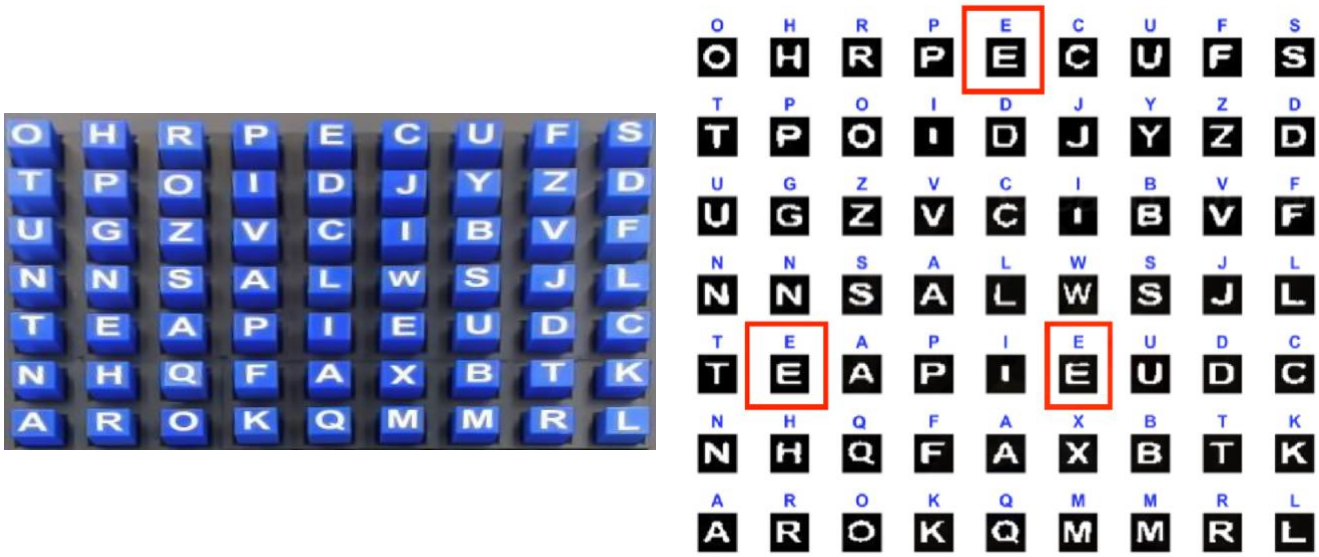


Figura 33. Análisis de los resultados - 1º prueba iluminación con LEDs.

En la primera prueba realizada, Figura 33, se observa que se ha logrado un 100% de acierto en las predicciones. En la figura se han marcado en rojo las diferentes letras "E" encontradas, a pesar de ser la misma letra y tipografía, se observa que hay diferencias entre ellas, y, a pesar de esas diferencias, la red realiza la predicción correctamente en todas ellas.

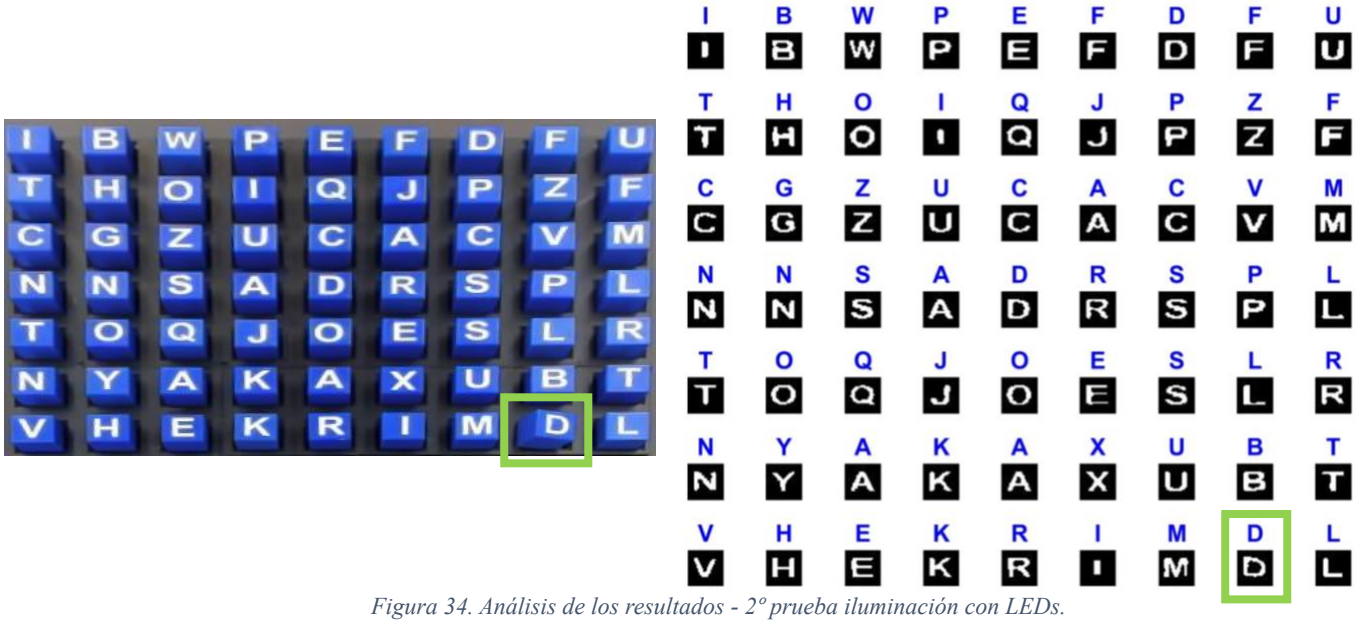


Figura 34. Análisis de los resultados - 2º prueba iluminación con LEDs.

En la segunda prueba, Figura 34, se vuelve a obtener un 100% de acierto. Además, en este caso, se observa que la letra "D" marcada en verde no está perfectamente colocada, se encuentra parcialmente girada en la plataforma, lo cual deforma levemente la letra. Y, aun así, la predicción realizada por la red ha sido correcta.

Por último, se realiza una tercera prueba, Figura 35, en la que también se consigue un 100% de acierto.

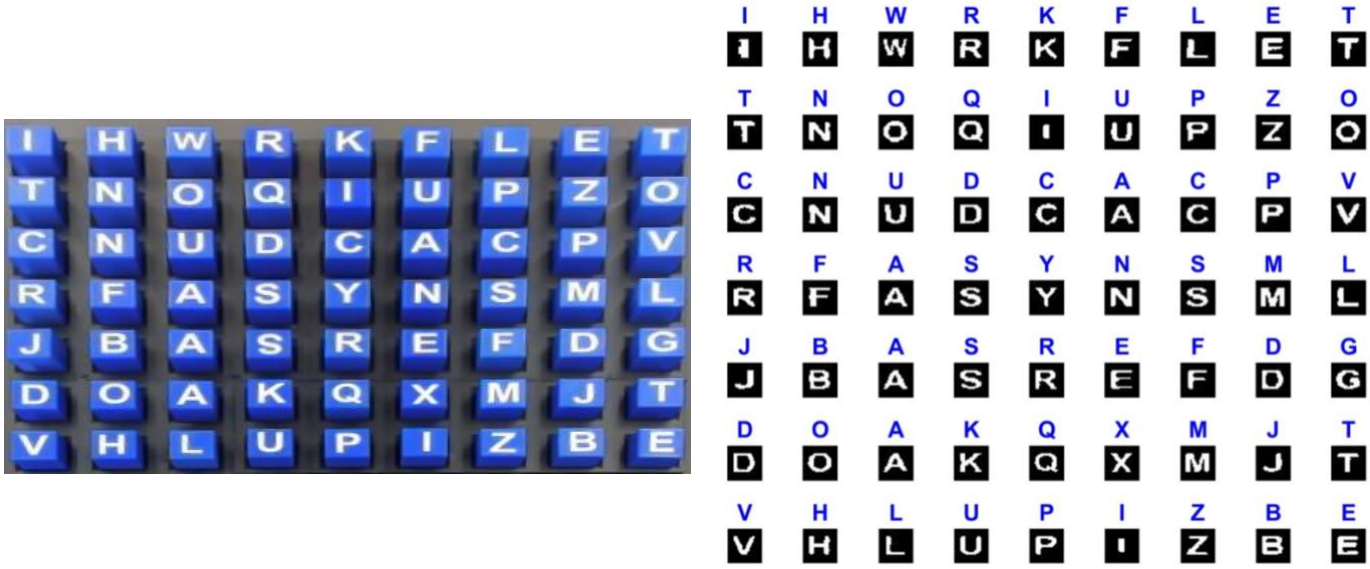


Figura 35. Análisis de los resultados - 3º prueba iluminación con LEDs.

Tras analizar los resultados obtenidos en las tres primeras pruebas realizadas, se corrobora que la aplicación es altamente robusta cuando se cuenta con buena iluminación, a continuación se realizan las siguientes tres pruebas sin iluminación artificial directa, es decir, apagando los LEDs ubicados en el chasis del robot.

En estas pruebas se ha realizado la captura de imagen en tres momentos diferentes del día (mismo día a las 17:00, 18:00 y 19:00), siendo la iluminación utilizada para ellas la de las lámparas del techo del laboratorio y la luz natural entrante por las ventanas. Lo que se quiere estudiar es cómo afecta la iluminación a la predicción de las letras, por ello, las tres pruebas se realizan con la misma disposición de bloques en la plataforma, la utilizada en la 3º prueba, asegurando así que el único factor cambiante sea la iluminación.

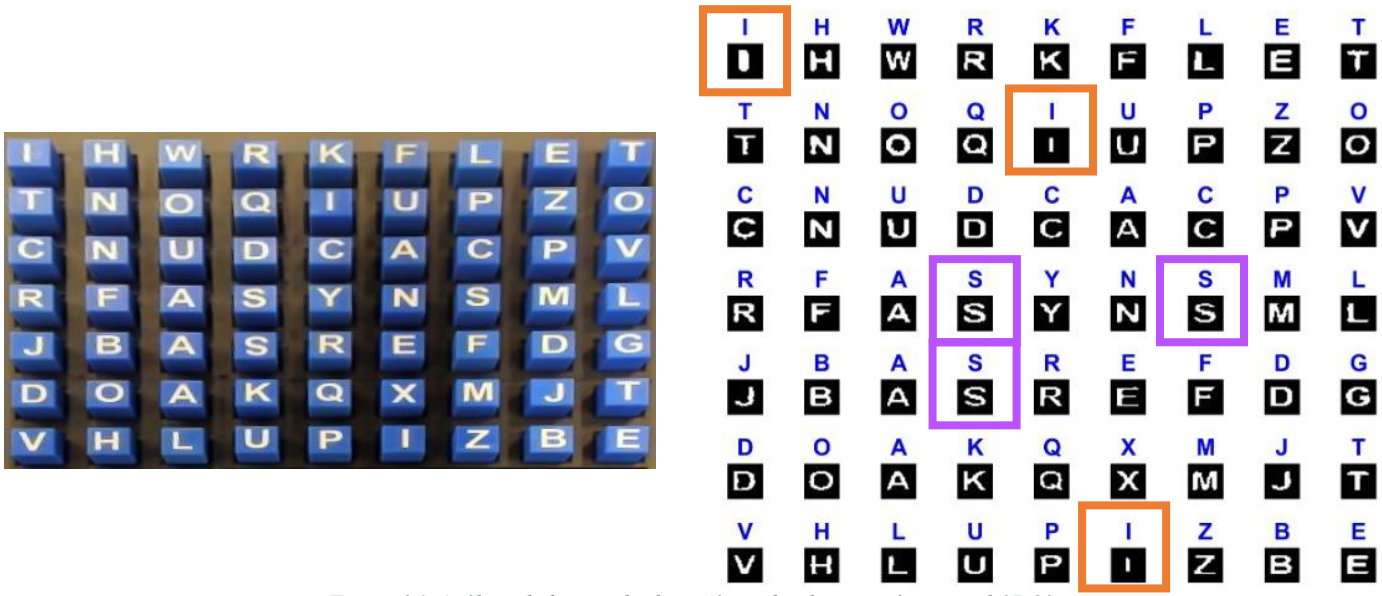


Figura 36. Análisis de los resultados - 4º prueba iluminación natural 17:00

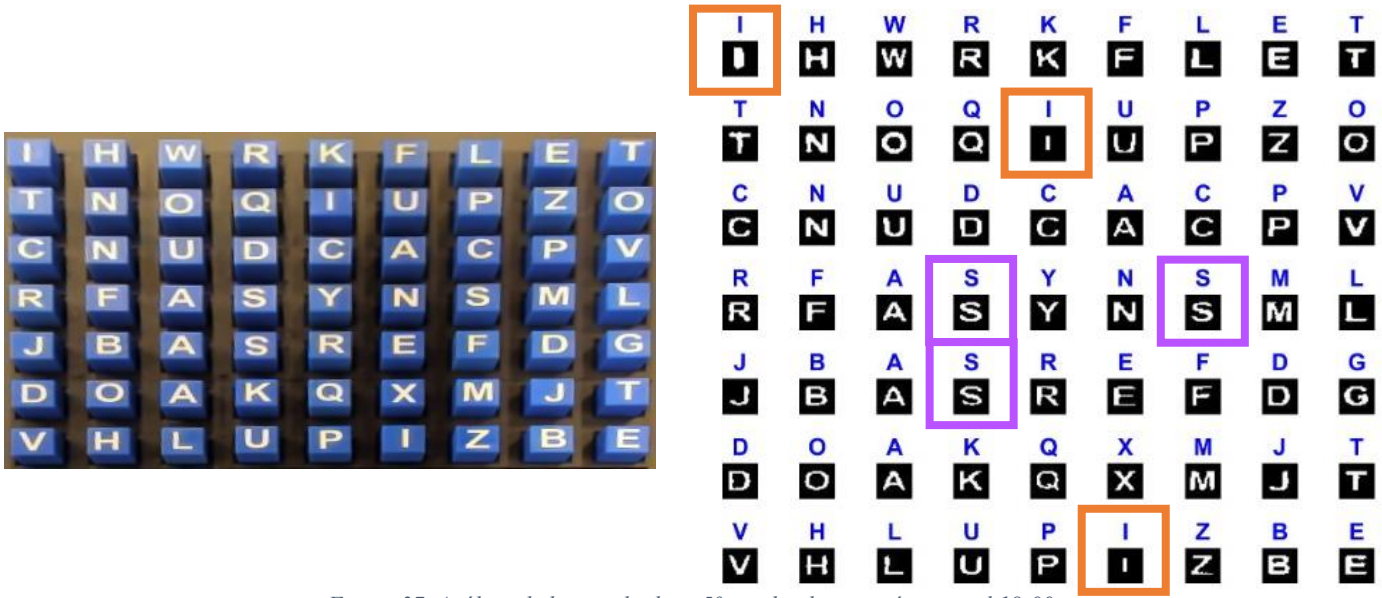


Figura 37. Análisis de los resultados - 5º prueba iluminación natural 18:00

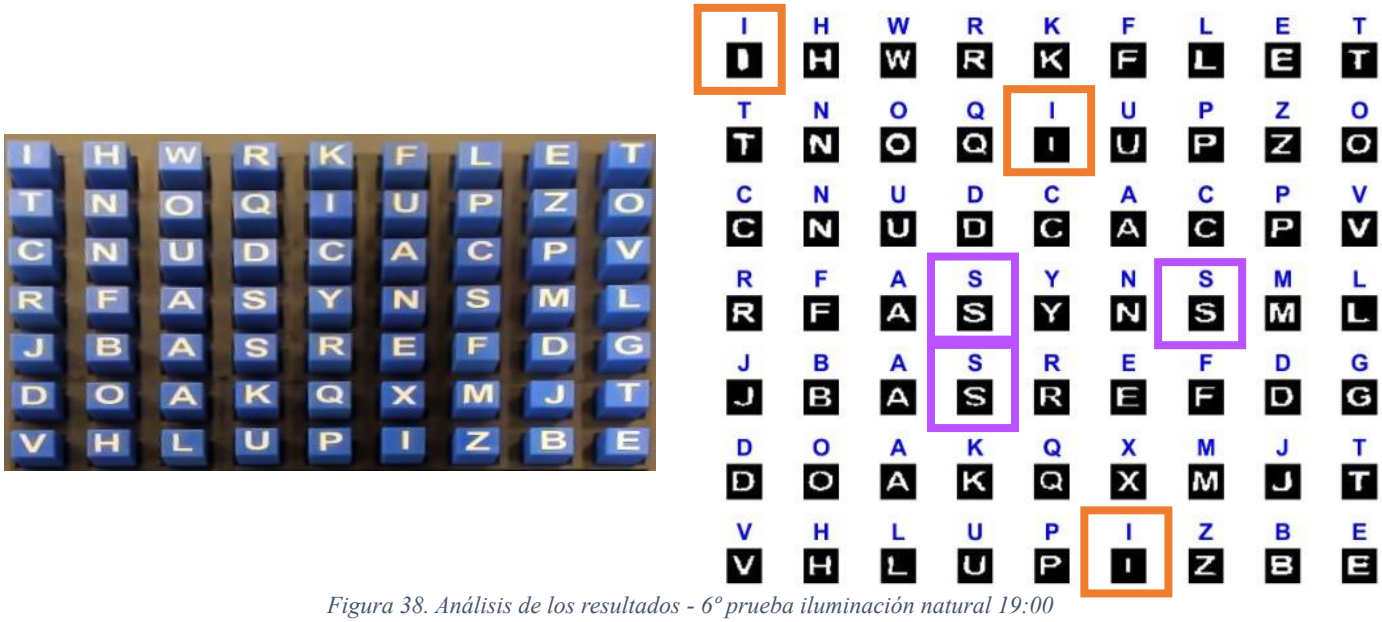


Figura 38. Análisis de los resultados - 6º prueba iluminación natural 19:00

En las tres pruebas realizadas con iluminación natural, Figura 36, Figura 37 y Figura 38, también se ha obtenido un porcentaje de acierto en las predicciones del 100%, por lo tanto, queda demostrada la alta robustez de la red diseñada, incluso ante cambios de iluminación.

Además, en dichas figuras, se han marcado las letras "I" en naranja y las letras "S" en morado, al estudiar las letras marcadas se aprecia la gran diferencia que hay en cuanto a tamaño o grosor entre ellas, pero a pesar de esas diferencias, la red logra acertar todas las letras.

En la Tabla 7 se recogen los resultados conseguidos en las pruebas realizadas, las seis pruebas se ha logrado un porcentaje de acierto del 100%, por lo que queda demostrada la variabilidad en cuanto a posicionamiento de los cubos, así como que la aplicación implementada es extremadamente robusta, incluso ante cambios de iluminación.

Tabla 7. Resultados de las pruebas realizadas.

Iluminación	N.º de prueba	Porcentaje de acierto
Iluminación alta y constante (LEDs encendidos)	1º Prueba	100%
	2º Prueba	100%
	3º Prueba	100%
Iluminación variable (lámparas del techo del laboratorio y la luz natural entrante por las ventanas)	4º Prueba	100%
	5º Prueba	100%
	6º Prueba	100%

Capítulo 6

Metodología seguida en el desarrollo del trabajo

6. Metodología seguida en el desarrollo del trabajo

En este apartado se describen las tareas realizadas durante el transcurso del trabajo. Se incluyen también los medios materiales y humanos utilizados en cada una de las tareas, que servirán como base para el desarrollo del presupuesto. Las tareas realizadas son las siguientes:

1. Toma de contacto con la aplicación existente.
2. Selección de la instrumentación utilizada.
3. Diseño de las piezas 3D.
4. Análisis de las diferentes soluciones.
5. Estudio del entorno Python.
6. Impresión de piezas 3D.
7. Programación del sistema de visión artificial.
8. Obtención de imagen y preprocesado.
9. Aprendizaje de uso del software Sysmac Studio.
10. Calibración y programación del robot.
11. Pruebas y validación.
12. Elaboración del informe.

En el siguiente apartado se procede a describir cada una de las tareas con mayor detalle.

6.1. Descripción de las tareas

En este apartado se procede a describir cada una de las tareas con mayor detalle:

6.1.1. Toma de contacto con la aplicación existente

En esta primera tarea se ha realizado una detallada investigación sobre lo realizado hasta el momento. Se ha investigado sobre la forma de funcionamiento del robot y de la aplicación de usuario, observando y entendiendo todo el código de programación utilizado para su funcionamiento.

- Duración: 3 semanas.
- Medios humanos: Ingeniera Junior.
- Medios materiales: Ordenador, webcam, robot paralelo.

6.1.2. Selección de la instrumentación utilizada

Una vez conocido el funcionamiento de la aplicación existente, se procede a la selección de la pinza que mejor se ajuste a los requerimientos de la aplicación.

- Duración: 1 semana.
- Medios humanos: Ingeniera Junior.
- Medios materiales: Ordenador.

6.1.3. Diseño de las piezas 3D

En esta tarea se ha realizado el diseño detallado de las piezas mostradas en el apartado 5.1.

- Duración: 3 semanas.

- Medios humanos: Ingeniera Junior.
- Medios materiales: Ordenador, SW Fusion 360

6.1.4. Análisis de las diferentes soluciones

En esta tarea se ha realizado una detallada búsqueda de información de los trabajos existentes relacionados con el reconocimiento de caracteres y soluciones existentes que integren robótica y visión artificial.

Además también se han estudiado las diferentes alternativas existentes en cuanto a tecnología y software de inteligencia artificial.

- Duración: 3 semanas.
- Medios humanos: Ingeniera Junior.
- Medios materiales: Ordenador.

6.1.5. Estudio del entorno Python

Dado se ha seleccionado Python como software de IA a utilizar, ha sido necesario realizar un proceso de aprendizaje en del entorno apoyándose en los recursos existentes en la red.

- Duración: 2 semanas.
- Medios humanos: Ingeniera Junior.
- Medios materiales: Ordenador.

6.1.6. Impresión de piezas 3D

Esta tarea incluye el tiempo de espera transcurrido en el proceso de impresión de las piezas mediante impresoras 3D.

- Duración: 3 semanas.
- Medios humanos: Ingeniera Junior.
- Medios materiales: Ordenador, set de herramientas.

6.1.7. Programación del sistema de visión artificial

En esta tarea se ha realizado la programación del sistema de visión artificial, es decir, el diseño y entrenamiento de la red neuronal convolucional.

- Duración: 6 semanas.
- Medios humanos: Ingeniera Junior.
- Medios materiales: Ordenador.

6.1.8. Obtención de imagen y preprocesado

En esta tarea se ha realizado la captura y preprocesamiento de la imagen capturada, así como la preparación de la entrada que se le pasa a la red neuronal.

- Duración: 6 semanas.
- Medios humanos: Ingeniera Junior.
- Medios materiales: Ordenador, webcam.

6.1.9. Aprendizaje de uso del software Sysmac Studio

Dado que el fabricante de robots dispone de su propio software y lenguaje de programación, ha sido necesario realizar un proceso de aprendizaje en la herramienta de Sysmac Studio apoyándose en los recursos existentes en la red.

- Duración: 2 semanas.
- Medios humanos: Ingeniera Junior.
- Medios materiales: Ordenador.

6.1.10. Calibración del robot

En esta tarea se ha realizado la calibración del robot.

- Duración: 2 semana.
- Medios humanos: Ingeniera Junior.
- Medios materiales: Ordenador, set de herramientas.

6.1.11. Pruebas y validación

Una vez comprobado el funcionamiento individual de cada uno de los elementos, se han realizado las pruebas de integración, comprobando el funcionamiento conjunto y solucionando los errores encontrados.

- Duración: 3 semanas.
- Medios humanos: Ingeniera Junior.
- Medios materiales: Ordenador.

6.1.12. Elaboración del informe

Esta tarea incluye el tiempo necesario para realizar un informe elaborado que exponga cada una de las tareas realizadas.

- Duración: 5 semanas.
- Medios humanos: Ingeniera Junior.
- Medios materiales: Ordenador.

6.2. Diagrama de Gantt

En este apartado, se muestra el diagrama de Gantt, Figura 39, de las tareas desarrolladas en el apartado anterior, obteniendo una duración total del proyecto de 36 semanas, comenzando el 12 de diciembre de 2022 y finalizando el 4 de septiembre de 2023.

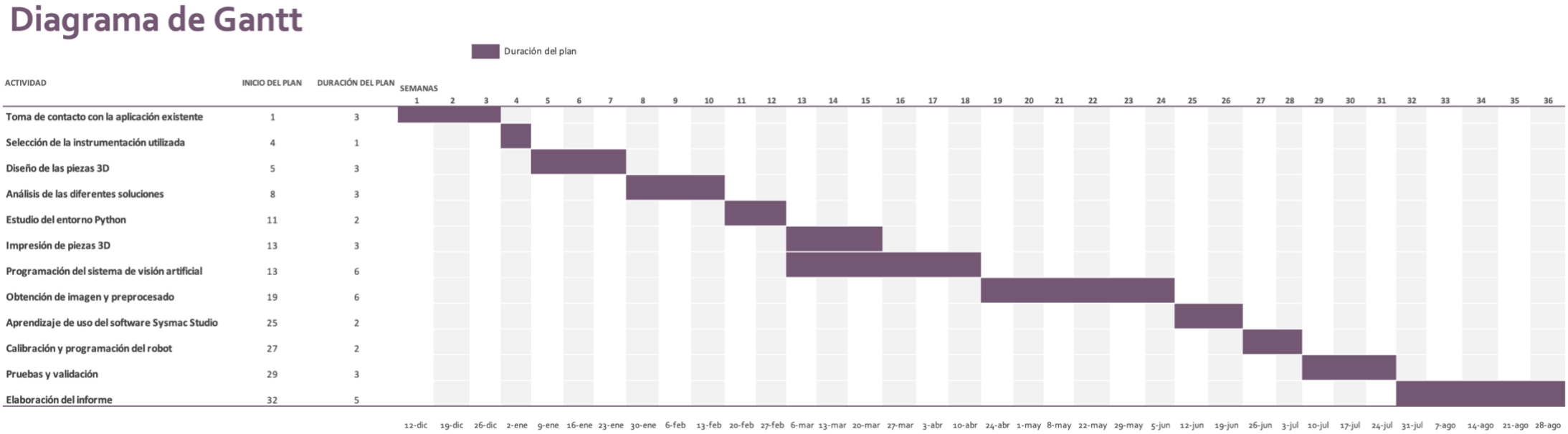


Figura 39. Diagrama de Gantt

Capítulo 7

Aspectos económicos

7. Aspectos económicos

En este apartado se realiza un estudio económico del proyecto, analizando por separado los costes materiales, los costes de mano de obra y los costes de software.

7.1. Costes materiales

En este apartado se muestra la Tabla 8, que contiene la identificación de cada producto, la cantidad, el precio por unidad y total de los componentes utilizados para realizar el proyecto.

Cabe destacar que en el caso del robot, de la controladora y del PC es necesario calcular la amortización correspondiente a la duración del proyecto:

- Robot → $\frac{23.890€}{15\ años}\ x\ \frac{1\ año}{52\ semanas} = 30{,}628\frac{€}{semana} x\ 36\ semanas = 1.102{,}615€$
- Controladora → $\frac{8.000€}{7\ años}\ x\ \frac{1\ año}{52\ semanas} = 21{,}978\frac{€}{semana} x\ 36\ semanas = 791{,}209\ €$
- PC → $\frac{700\ €}{4\ años}\ x\ \frac{1\ año}{52\ semanas} = 3{,}365\frac{€}{semana}\ x\ 36\ semanas = 121{,}15€$

Tabla 8. Costes materiales

REFERENCIA	CANTIDAD	DESCRIPCIÓN	PVP
CR-UGD4MINI-NR	1	Robot Delta	1.102,62 €
R88D-KN04H-ECT	3	Servodrivers Accurax G5	2.880,00 €
R88A-CAKA005SR-E	3	Cables de potencia XYZ	540,00 €
R88A-CRKA005CR-E	3	Cables de encoder	360,00 €
R88A-CAKA005BR-E	3	Cables de freno	110,00 €
R88A-CRGD0R3C-BS	3	Batería y cables encoder	540,00 €
R88A-CSK003S-E	3	Cables de seguridad	119,00 €
R88A-CNW01C	3	Conectores de E/S (26 pines)	105,00 €
XS6W6LSZH8SS50CMY	3	Cables EtherCAT 0,5m	18,00 €
NJ501-4500	1	Controlador NJ	791,21 €
NJ-PA3001	1	Fuente controlador NJ	320,00 €
CP1W-CN221	1	Cable programación NJ	12,00 €
NX-ECC201	1	Cabecera NX	220,00 €
XS6W6LSZH8SS30CMY	1	Cable EtherCAT a cabecera	6,00 €
NX-EC0142	2	Tarjeta encoder NX	668,00 €
E6B2-CWZ1X2000PR05MOMS	2	Encoder	478,00 €
Logitech Brio Webcam 4k	1	Webcam	163,30 €
LG 17MB15T-B-	1	Monitor táctil de 17''	252,89 €
Tiras Led	1	Tiras led de 7 metros	21,99 €
PC	1	Ordenador	121,15 €
Adaptador HDMI a VGA	1	Adaptador PC a monitor	9,34 €
Sparmax TC-610H Plus Air	1	Compresor	269,00 €
R88A-BAT01G 3.6V 2700mAh	3	Batería Litio servodrivers	47,85 €
DHPS-16-A	1	Pinza paralela DHPS	486,64 €
		SUBTOTAL	**9.641,99 €**

7.2. Costes de mano de obra

En este apartado se recogen los costes de personal relacionados con las diferentes partes del proyecto. Siendo el coste de la hora trabajada para una Ingeniera Junior de 30€/h. El desglose de los costes se muestra en la Tabla 9.

Tabla 9. Costes mano de obra

PUESTO	€/HORA	Nº HORAS	SUBTOTAL
Estudio del estado del arte	**30**	**40**	**1.200,00 €**
Diseño de piezas 3D	**30**	**45**	**1.350,00 €**
Desarrollo del programa	**30**	**400**	**12.000,00 €**
Validación de resultados	**30**	**30**	**900,00 €**
Elaboración del informe	**30**	**100**	**3.000,00 €**
Horas totales	**615**		
SUBTOTAL	**18.450,00 €**		

7.3. Costes de software

En la Tabla 10 se recogen los costes que conllevan las licencias necesarias para utilizar los programas necesarios para el desarrollo del proyecto.

Tabla 10. Costes de software

CONCEPTO	SUBTOTAL
Licencia Fusion 360	530,00 €
Licencia MATLAB	860,00 €
Licencia Sysmac Studio	2.200,00 €
SUBTOTAL	**3.590,00 €**

7.4. Costes totales

Los costes se calculan realizando la suma de los costes materiales, los costes de mano de obra y los costes de software, Tabla 11.

Tabla 11. Costes totales.

CONCEPTO	SUBTOTAL
Costes materiales totales	9.641,99 €
Coste mano de obra directa	18.450,00 €
Costes de equipo y software	3.590,00 €
SUBTOTAL	**31.681,99 €**

Además, se estima un 13% (4.118,66€) para gastos generales y un 6% (1.900,92€) para el beneficio industrial, haciendo con todo esto un **presupuesto total de 37.701,57€.**

Capítulo 8
Conclusiones

8. Conclusiones

En el transcurso de este proyecto de investigación y desarrollo, se ha logrado alcanzar de manera exitosa cada uno de los objetivos planteados inicialmente. Este proyecto, centrado en la implementación de inteligencia artificial y visión para el reconocimiento de caracteres impresos en cubos, ha finalizado con resultados muy satisfactorios que demuestran tanto su viabilidad como su eficacia.

En primer lugar, se ha diseñado un espacio de trabajo para el robot, seleccionando los componentes necesarios y modelando piezas auxiliares mediante tecnología de impresión 3D. Esta etapa del proyecto ha sido fundamental para garantizar la correcta manipulación de los cubos.

Además, se ha abordado el aprendizaje de diferentes herramientas software, como MATLAB, Python, Fusion 360 y Sysmac Studio, que permiten la manipulación de datos, el diseño 3D y la programación del robot. La adquisición de estas competencias ha sido fundamental para ofrecer un enfoque interdisciplinario y versátil en el desarrollo del proyecto.

El objetivo principal del proyecto era lograr el reconocimiento de caracteres a través de técnicas inteligentes de aprendizaje. La red neuronal convolucional entrenada ha conseguido un rendimiento excelente, con una precisión de validación de 92,87%, y un error de validación de 2,39%.

Adicionalmente, se han aplicado técnicas de preprocesamiento de imágenes para mejorar la calidad de las imágenes capturadas y adecuarlas a la entrada de la red. La mejora realizada en las imágenes, corrección de perspectiva, transformación de escala de colores y segmentación de letras han permitido aumentar la precisión de las predicciones, logrando un porcentaje de acierto del 100%.

El funcionamiento completo del sistema ha sido posible gracias a la integración de la aplicación de usuario con el modelo de reconocimiento de caracteres y con el software de control del robot.

En conclusión, este proyecto demuestra la factibilidad y efectividad de la implementación de inteligencia artificial y visión en una aplicación robótica industrial para el reconocimiento y manipulación de caracteres impresos en cubos.

Capítulo 9
Bibliografía

9. Bibliografía

[1] R. Alonso Mijangos, «APLICACIÓN DE PICK AND PLACE CON ROBOT PARALELO MINI DELTA INTEGRANDO VISIÓN ARTIFICIAL».

[2] G. Beier, A. Ullrich, S. Niehoff, M. Reißig, y M. Habich, «Industry 4.0: How it is defined from a sociotechnical perspective and how much sustainability it includes – A literature review», *J Clean Prod*, vol. 259, jun. 2020, doi: 10.1016/J.JCLEPRO.2020.120856.

[3] R. Y. Zhong, X. Xu, E. Klotz, y S. T. Newman, «Intelligent Manufacturing in the Context of Industry 4.0: A Review», *Engineering*, vol. 3, n.º 5, pp. 616-630, 2017, doi: 10.1016/J.ENG.2017.05.015.

[4] R. M. A. Nzue, J. F. Brethé, E. Vasselin, y D. Lefebvre, «Comparison of serial and parallel robot repeatability based on different performance criteria», *Mech Mach Theory*, vol. 61, pp. 136-155, 2013, doi: 10.1016/J.MECHMACHTHEORY.2012.10.004.

[5] T. Su, L. Cheng, Y. Wang, X. Liang, J. Zheng, y H. Zhang, «Time-Optimal Trajectory Planning for Delta Robot Based on Quintic Pythagorean-Hodograph Curves», *IEEE Access*, vol. 6, pp. 28530-28539, abr. 2018, doi: 10.1109/ACCESS.2018.2831663.

[6] Q. Meng, F. Xie, y X. J. Liu, «Conceptual design and kinematic analysis of a novel parallel robot for high-speed pick-and-place operations», *Frontiers of Mechanical Engineering*, vol. 13, n.º 2, pp. 211-224, jun. 2018, doi: 10.1007/S11465-018-0471-4.

[7] W. Ding, J. Gu, S. Tang, Z. Shang, E. A. Duodu, y C. Zheng, «Development of a calibrating algorithm for Delta Robot's visual positioning based on artificial neural network», *Optik (Stuttg)*, vol. 127, n.º 20, pp. 9095-9104, oct. 2016, doi: 10.1016/J.IJLEO.2016.06.126.

[8] K. Harada, T. Tsuji, K. Nagata, N. Yamanobe, y H. Onda, «Validating an object placement planner for robotic pick-and-place tasks», *Rob Auton Syst*, vol. 62, n.º 10, pp. 1463-1477, 2014, doi: 10.1016/J.ROBOT.2014.05.014.

[9] Y. Xue, Y. Tong, Z. Yuan, S. Su, A. Slowik, y S. Toglaw, «Handwritten character recognition based on improved convolutional neural network», *Intelligent Automation and Soft Computing*, vol. 29, n.º 2, pp. 497-509, 2021, doi: 10.32604/IASC.2021.016884.

[10] Z. Li, Q. Wu, Y. Xiao, M. Jin, y H. Lu, «Deep Matching Network for Handwritten Chinese Character Recognition», *Pattern Recognit*, vol. 107, p. 107471, nov. 2020, doi: 10.1016/J.PATCOG.2020.107471.

[11] W. Sihang, W. Jiapeng, M. Weihong, y J. Lianwen, «Precise detection of Chinese characters in historical documents with deep reinforcement learning», *Pattern Recognit*, vol. 107, nov. 2020, doi: 10.1016/J.PATCOG.2020.107503.

[12] T. Kausar, S. Manzoor, A. Kausar, Y. Lu, M. Wasif, y M. Adnan Ashraf, «Deep Learning Strategy for Braille Character Recognition», *IEEE Access*, vol. 9, pp. 169357-169371, 2021, doi: 10.1109/ACCESS.2021.3138240.

[13] C. Henry, S. Y. Ahn, y S. W. Lee, «Multinational License Plate Recognition Using Generalized Character Sequence Detection», *IEEE Access*, vol. 8, pp. 35185-35199, 2020, doi: 10.1109/ACCESS.2020.2974973.

[14] S. Juneja, A. Juneja, G. Dhiman, S. Jain, A. Dhankhar, y S. Kautish, «Computer Vision-Enabled Character Recognition of Hand Gestures for Patients with Hearing and Speaking Disability», 2021, doi: 10.1155/2021/4912486.

[15] A. Baldominos, Y. Saez, y P. Isasi, «A survey of handwritten character recognition with MNIST and EMNIST», *Applied Sciences (Switzerland)*, vol. 9, n.º 15, ago. 2019, doi: 10.3390/APP9153169.

[16] B. Ma, X. Li, Y. Xia, y Y. Zhang, «Autonomous deep learning: A genetic DCNN designer for image classification», *Neurocomputing*, vol. 379, pp. 152-161, feb. 2020, doi: 10.1016/J.NEUCOM.2019.10.007.

[17] R. Vaila, J. Chiasson, y V. Saxena, «A Deep Unsupervised Feature Learning Spiking Neural Network With Binarized Classification Layers for the EMNIST Classification», *IEEE Trans Emerg Top Comput Intell*, vol. 6, n.º 1, pp. 124-135, feb. 2022, doi: 10.1109/TETCI.2020.3035164.

[18] «Qué son las redes neuronales y sus funciones | ATRIA Innovation». https://www.atriainnovation.com/que-son-las-redes-neuronales-y-sus-funciones/ (accedido 24 de julio de 2023).

[19] «Máquinas de Vectores de Soporte (SVM) - IArtificial.net». https://www.iartificial.net/maquinas-de-vectores-de-soporte-svm/ (accedido 24 de julio de 2023).

[20] «Pinza paralela DHPC», Accedido: 1 de agosto de 2023. [En línea]. Disponible en: www.festo.com/catalogue/...

[21] «Pinza paralela DHPS», Accedido: 1 de agosto de 2023. [En línea]. Disponible en: www.festo.com

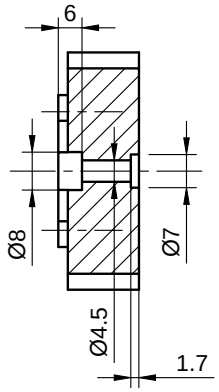
[22] «Pinza paralela DHPS-16-A-NO | Festo ES». https://www.festo.com/es/es/a/1254044/?q=~:sortByCoreRangeAndSp2020 (accedido 1 de agosto de 2023).

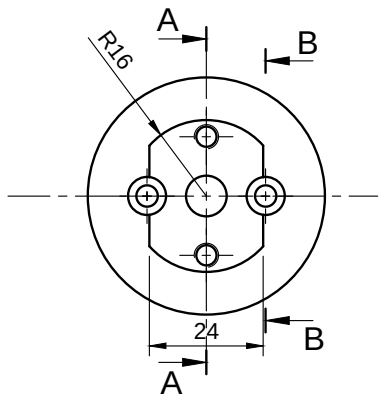
[23] G. K. Cohen, S. Afshar, J. Tapson, y A. van Schaik, «EMNIST: an extension of MNIST to handwritten letters», *Arxiv preprint*, feb. 2017, doi: 10.48550/arxiv.1702.05373.

Anexo I
Planos

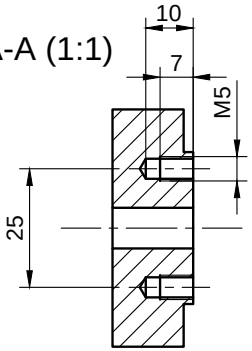
Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

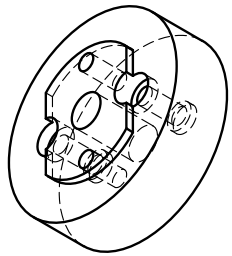
B-B (1:1)



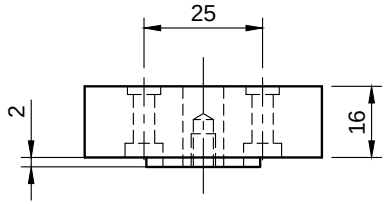


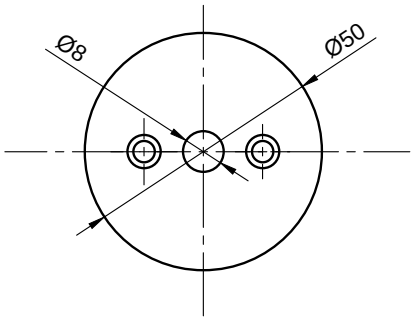
A-A (1:1)





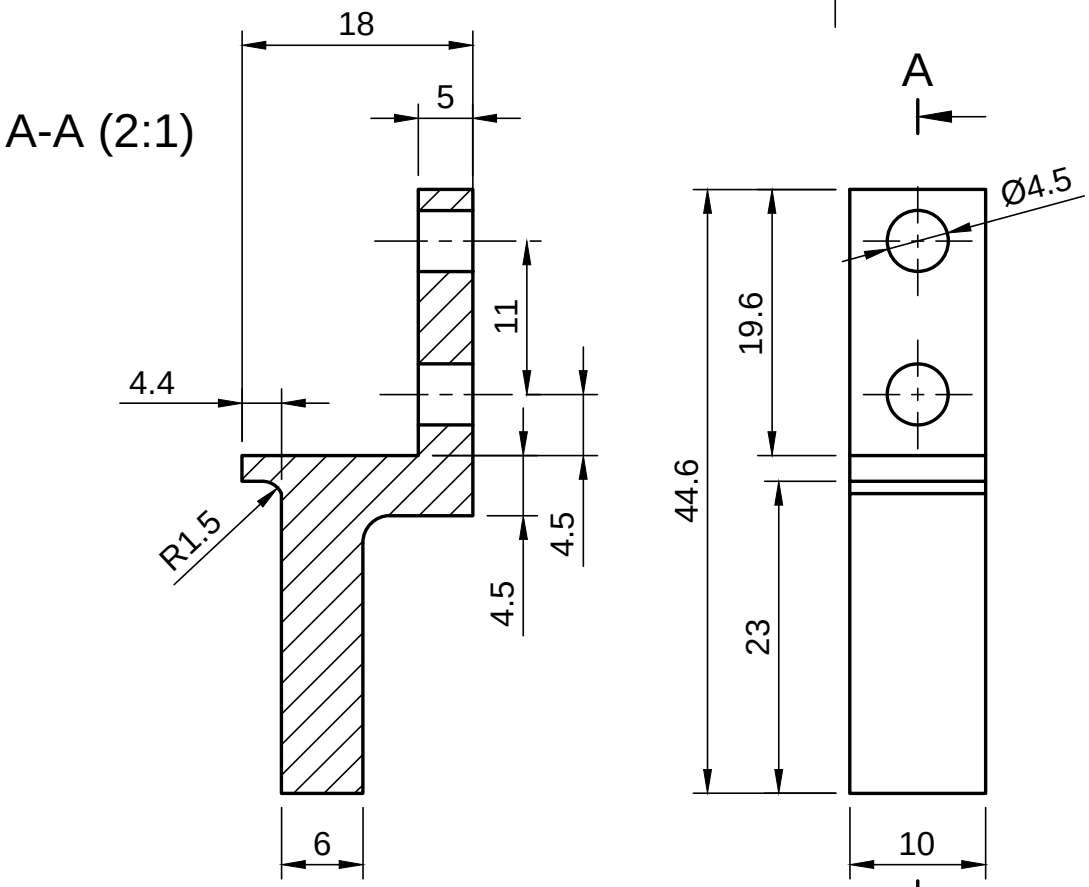
Isometric view
Scale: 1:1

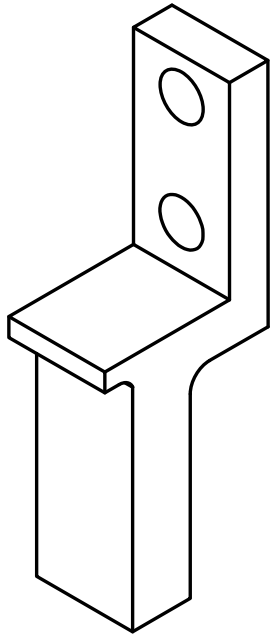




Created by: Enara Sarachaga Cortés		Title: Adaptador Nuevo Robot MiniDelta
Date: 30/11/22	Quantity to be produced: 1	
Scale: 1:1	Size: A3	

Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO
ViSens UPV/EHU





Isometric view
Scale: 2:1

Created by: Enara Sarachaga Cortés		Title: Dedos Pinza Nueva Robot MiniDelta
Date: 07/02/2023	Quantity to be produced: 2	
	Scale: 2:1	Size: A4

Anexo II

Código de diseño de la red neuronal

Anexo II.I – Código para la preparación de las variables de la red

```
"""
DESCRIPCIÓN:
    Se trata de un código que a partir de la lectura de un fichero *.csv,
realiza la separación entre variables
    independientes y dependiente, y, tras realizar un normalizado y
reescalado de imagen las almacena en un fichero
    *.pickle.

    Enara Sarachaga Cortés

TRAZABILIDAD DE VERSIONES:
============================================================================
==========================================
    V1r0_280123_ESC: Versión Inicial, características:
        1.- Carga datos del *.csv referentes al entrenamiento
        2.- Separa los datos en las variables independientes (X) y
dependientes (y)
        3.- Normaliza y reescala las variables independientes
        4.- Exporta en un fichero pickle ambas variables
============================================================================
==========================================
"""

import numpy as np
import pandas as pd
import os
import datetime as dt
import time
import pickle

# Data load
data_loadPath = r'/Users/enara/Desktop/TFM/codigo_python/data'
data_loadName = r'emnist-letters-train.csv'
data_path = os.path.join(data_loadPath, data_loadName)

# Data save
data_savePath = r'/Users/enara/Desktop/TFM/codigo_python/results/var'
data_saveName = r'var.pickle'
save_path = os.path.join(data_savePath, data_saveName)

#============================================================================
==========================================
def MAIN(data_path, save_path, HEIGHT, WIDTH):
    print('INICIO de la EJECUCIÓN', dt.datetime.now(), '\n')

    print('CARGANDO datos del fichero...'); tic = time.time()
    dfDAT = read_data(data_path)
    print('// Fin:', str(time.time() - tic), '[s]')

    print('SEPARANDO datos en variables X e y...'); tic = time.time()
    X, y = var_split(dfDAT)
    print('// Fin:', str(time.time() - tic), '[s]')

    print('NORMALIZANDO y realizando el REESCALADO de imagen...'); tic =
time.time()
    X = data_preprocessing(X, HEIGHT, WIDTH)
```

```
    print('// Fin:', str(time.time() - tic), '[s]')

    print('ALMACENANDO las variables en un fichero *.pickle...'); tic =
time.time()
    var_save(save_path, X, y)
    print('// Fin:', str(time.time() - tic), '[s]')

    print('FINAL de la EJECUCIÓN', dt.datetime.now(), '\n')
#================================================================================
==========================================
def read_data(data_path):
    """
    Se trata de una función que realiza la lectura del fichero *.csv y lo
almacena en un dataframe.

        :param
            data_path: Ruta del fichero *.csv que contiene el dataset de
partida.

        :return
            dfDAT: Dataframe que contiene los datos leidos del fichero
*.csv.
    """
    dfDAT = pd.read_csv(data_path)

    return dfDAT
#================================================================================
==========================================
def var_split(dfDAT):
    """
    Se trata de una función que realiza el split entre las variables
independientes (X) y la variable dependiente (y)

        :param
            dfDAT: Dataframe que contiene los datos leidos del fichero
*.csv.

        :return
            X: numpy array que contiene los datos de las variables
independientes
            y: Serie que contiene los valores de la variable dependiente
    """
    # Separamos los datos de las imagenes del label
    X = dfDAT.iloc[:, 1:].values
    y = dfDAT.iloc[:, 0]

    return X, y
#================================================================================
==========================================
def data_preprocessing(X, HEIGHT, WIDTH):
    """
    Se realiza el preprocesamiento de los valores de las variables
independientes. Para ello se realiza un normalizado
    de datos, así como un reescalado de imagen.

        :param
            X[n de imagenes, n de pixeles]: numpy array 2D que contiene los
datos de las variables independientes.

        :return
            X_rotate[n de imagenes, HEIGHT, WIDTH]: numpy array 3D que
```

```
    contiene los datos de las variables independientes, tras su normalización y
    reescalado.
        """
        X_rotate = []
        for image in X:
            tmp = rotate(image, HEIGHT, WIDTH)
            X_rotate.append(tmp)

        X_rotate = np.array(X_rotate)

        return X_rotate
#==============================================================================
==============================================
def rotate(image, HEIGHT, WIDTH):
        """
        Se trata de una función que reescala, rota (en el sentido de las agujas
    del reloj 90°) y voltea (horizontalmente)
        la imagen para su correcta visualización.

            :param
                image: numpy array que contiene los valores de las variables
    independientes de una unica imagen

            :return
                image: numpy array que contiene los valores de las variables
    independientes de una unica imagen, tras su
                reescalado, rotación y volteado
        """
        image = image.reshape([HEIGHT, WIDTH])
        image = np.fliplr(image)
        image = np.rot90(image)

        return image
#==============================================================================
==============================================
def var_save(save_path, X, y):
        """
        Se trata de una función que guarda las variables X e y en un archivo
    pickle

            :param
                save_path: Ruta y nombre del fichero *.pickle donde se
    almacenarán las variables generadas
                X: numpy array que contiene los datos de las variables
    independientes
                y: Serie que contiene los valores de la variable dependiente

            :return
                None
        """
        # Open a file and use dump()
        with open(save_path, 'wb') as file:
            # A new file will be created
            pickle.dump((X, y), file)
#==============================================================================
==============================================
if __name__ == '__main__':
    MAIN(data_path, save_path, HEIGHT=28, WIDTH=28)
```

Anexo II.II – Código para el diseño, entrenamiento y generación del modelo

```
"""
DESCRIPCIÓN:
    Se trata de un código que a partir de la carga de variables X e y
almacenadas en el script "varGeneration.py",
    realiza un split de los datos, dividiendo estos en los subconjuntos de
entrenamiento y validación, para
    posteriormente realizar el entrenamiento de una red neuronal
convolucional y almacenar el modelo generado.

    Enara Sarachaga Cortés

TRAZABILIDAD DE VERSIONES:
============================================================================
==========================================
    V1r0_280123_ESC: Versión Inicial, características:
        1.- Carga variables X e y del fichero *.pickle generado en el
script "varGeneration.py"
        2.- Prepara los datos para el entrenamiento
        3.- Realiza el split diviendo los datos en los subconjuntos de
train y test
        4.- Realiza un entrenamiento de un modelo basado en una red
neuronal convolucional
        4.- Exporta dicho modelo entrenado
============================================================================
==========================================
"""
import numpy as np
import os
import datetime as dt
import time
import pickle
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.utils import np_utils
from keras.callbacks import EarlyStopping,
ReduceLROnPlateau,ModelCheckpoint

# Var load
var_loadPath = r'/Users/enara/Desktop/TFM/codigo_python/results/var'
var_loadName = r'var.pickle'
var_path = os.path.join(var_loadPath, var_loadName)

# Model save
model_savePath = r'/Users/enara/Desktop/TFM/codigo_python/results/model'
model_saveName = r'model_bw.pickle'
model_path = os.path.join(model_savePath, model_saveName)

# Plot save
plot_savePath =
r'/Users/enara/Desktop/TFM/codigo_python/results/model/plot'

#============================================================================
==========================================
```

```
def MAIN(var_path, model_path, evaluate, plot_savePath, HEIGHT, WIDTH):
    print('INICIO de la EJECUCIÓN', dt.datetime.now(), '\n')

    print('CARGANDO datos del fichero...'); tic = time.time()
    X, y = load_var(var_path)
    print('// Fin:', str(time.time() - tic), '[s]')

    print('PREPARANDO datos para la ingesta en el modelo...'); tic =
time.time()
    X, y, num_classes = data_preprocessing(X, y, HEIGHT, WIDTH)
    print('// Fin:', str(time.time() - tic), '[s]')

    print('GENERANDO y ENTRENANDO modelo...'); tic = time.time()
    model, X_test, y_test = model_training(X, y, num_classes, HEIGHT,
WIDTH)
    print('// Fin:', str(time.time() - tic), '[s]')

    if evaluate == 'TRUE':
        print('EVALUANDO el modelo...'); tic = time.time()
        model_evaluation(model, plot_savePath, X_test, y_test)
        print('// Fin:', str(time.time() - tic), '[s]')
    else:
        pass

    print('ALMACENANDO el modelo en un fichero *.pickle...'); tic =
time.time()
    model_save(model, model_path)
    print('// Fin:', str(time.time() - tic), '[s]')

    print('FINAL de la EJECUCIÓN', dt.datetime.now(), '\n')
#=============================================================================
==========================================
def load_var(var_path):
    """
    Se trata de una función que carga las variables X e y almacenadas
previamente en un fichero *.pickle

        :param
            var_path: Ruta y nombre del fichero *.pickle donde previamente
se han almacenado las variables generadas

        :return
            X: numpy array que contiene los datos de las variables
independientes
            y: Serie que contiene los valores de la variable dependiente
    """
    # Open the file in binary mode
    with open(var_path, 'rb') as file:
        # Call load method to deserialze
        X, y = pickle.load(file)

    return X, y
#=============================================================================
==========================================
def data_preprocessing(X, y, HEIGHT, WIDTH):
    """
    Se trata de una función que adapta las variables X e y para su
posterior ingesta en el modelo

        :param
            X [num de imagenes, HEIGTH, WIDTH]: numpy array 3D que contiene
```

```
los datos de las variables independientes
            y: Serie que contiene los valores de la variable dependiente

        :return
            X [num de imagenes, HEIGTH, WIDTH, 1]: numpy array 4D que
contiene los datos de las variables independientes, preparadas para su
ingesta en el
            modelo
            y [num de imagenes, num de clases]: numpy array 2D que contiene
los valores de la variable dependiente, preparada para su ingesta en el
modelo.
            num_classes: int con el numero de clases que existen
    """
    # Determina el numero de clases
    num_classes = y.nunique()

    # One hot encoding
    y = np_utils.to_categorical(y,num_classes + 1)  # La clasificación es
de 1 a 26 por lo que hay que sumar uno, los arrays empiezan en 0
    y = np.delete(y, 0, 1)  # Eliminamos la columna 0,  no tiene
información

    # Reshape image for CNN
    X = X.reshape(-1, HEIGHT, WIDTH, 1)

    return X, y, num_classes
#=============================================================================
=========================================
def model_training(X, y, num_classes, HEIGHT, WIDTH):
    """
    Se trata de una función que genera y realiza el entrenamiento del
modelo

        :param
            X [num de imagenes, HEIGTH, WIDTH, 1]: numpy array 4D que
contiene los datos de las variables independientes, preparadas para su
ingesta en el
            modelo
            y [num de imagenes, num de clases]: numpy array 2D que contiene
los valores de la variable dependiente, preparada para su ingesta en el
modelo.
            num_classes: int con el numero de clases que existen

        :return
            history: Modelo entrenado
            X_test [num de imagenes de test, HEIGTH, WIDTH, 1]: numpy array
4D que contiene los datos de las variables
            independientes del subconjunto test, preparadas para su ingesta
en el modelo
            y_test [num de imagenes de test, num de clases]: numpy array 2D
que contiene los valores de la variable
            dependiente del subconjunto test, preparada para su ingesta en
el modelo.
            num_classes: int con el numero de clases que existen
    """
    # partition to train and val
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.30, random_state=7)

    # Building model
    model = Sequential()
```

```
    model.add(Conv2D(filters=128, kernel_size=(5, 5), padding='same',
activation='relu', \
                     input_shape=(HEIGHT, WIDTH, 1)))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
    model.add(Conv2D(filters=64, kernel_size=(3, 3), padding='same',
activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Flatten())
    model.add(Dense(units=128, activation='relu'))
    model.add(Dropout(.5))
    model.add(Dense(units=num_classes, activation='softmax'))

    model.summary()

    model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

    MCP = ModelCheckpoint('Best_points.h5', verbose=1, save_best_only=True,
monitor='val_accuracy', mode='max')
    ES = EarlyStopping(monitor='val_accuracy', min_delta=0, verbose=0,
patience=3, mode='max')
    RLP = ReduceLROnPlateau(monitor='val_loss', patience=3, factor=0.2,
min_lr=0.0001)

    history = model.fit(X_train, y_train, epochs=50, batch_size=512,
verbose=1, \
                        validation_data=(X_test, y_test), callbacks=[MCP,
ES, RLP])

    return history, X_test, y_test
#================================================================================
==========================================
def model_evaluation(model, plot_savePath, X_test, y_test):
    """
    Se trata de una función OPCIONAL que realiza la evaluación del modelo
entrenado mediante la visualización y
    almacenado de representaciones gráficas.

        :param
            model: Modelo entrenado
            X_test [num de imagenes de test, HEIGTH, WIDTH, 1]: numpy array
4D que contiene los datos de las variables
            independientes del subconjunto test, preparadas para su ingesta
en el modelo
            y_test [num de imagenes de test, num de clases]: numpy array 2D
que contiene los valores de la variable
            dependiente del subconjunto test, preparada para su ingesta en
el modelo.
            plot_savePath: Ruta donde se almacenan los ploteos realizados

        :return
            None
    """
    # Evaluating model on validation data.
    scores = model.model.evaluate(X_test, y_test, verbose=0)
    print("Accuracy: %.2f%%" % (scores[1] * 100))

    print(model.history.keys())
```

```
    # summarize history for accuracy
    plt.plot(model.history['accuracy'])
    plt.plot(model.history['val_accuracy'])
    plt.title('Model Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper left')
    plt.grid()
    plt.show()
    plot_name0 = 'ModelAccuracy.jpg'
    plot0_savePath = os.path.join(plot_savePath, plot_name0)
    plt.savefig(plot0_savePath)

    # summarize history for loss
    plt.plot(model.history['loss'])
    plt.plot(model.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.grid()
    plt.show()
    plot_name1 = 'ModelLoss.jpg'
    plot1_savePath = os.path.join(plot_savePath, plot_name1)
    plt.savefig(plot1_savePath)
#=============================================================================
==========================================
def model_save(model, model_path):
    """
    Se trata de una función que almacena el modelo entrenado en un fichero
*.pickle

        :param
            model: Modelo entrenado
            model_path: Ruta y nombre donde se almacenará el fichero
*.pickle con el modelo entrenado

        :return
            None
    """
    # Open a file and use dump()
    with open(model_path, 'wb') as file:
        # A new file will be created
        pickle.dump(model, file)
#=============================================================================
==========================================
if __name__ == '__main__':
    MAIN(var_path, model_path, evaluate='TRUE',
plot_savePath=plot_savePath, HEIGHT=28, WIDTH=28)
```

Anexo II.III – Código para realizar las predicciones

```
import pickle
import os
import pandas as pd
import numpy as np
#import matplotlib.pyplot as plt

def recognition():
    # Set label map
    label_map_path =
r'C:\Users\870113\Desktop\TFM_EnaraSaratxaga\Codigo_matlab\emnist-letters-
mapping.txt'
    label_map = pd.read_csv(label_map_path, delimiter=' ', index_col=0,
header=None, squeeze=True)

    # Model load
    archivo_csv= "./letters_to_predict.csv"
    model_loadPath =
r'C:\Users\870113\Desktop\TFM_EnaraSaratxaga\Codigo_matlab'
    model_loadName = r'model_bw.pickle'
    model_path = os.path.join(model_loadPath, model_loadName)

    HEIGHT =28
    WIDTH=28


#=============================================================================
=========================================

    label_dict = {}
    for index, label in enumerate(label_map):
        label_dict[index] = chr(label)


#=============================================================================
============================================================
    """
      Se trata de una función que realiza la carga del modelo previamente
generado y entrenado, además de cargar el
      subconjunto de datos desconocidos para el modelo, con el cual se van
a realizar las predicciones.

          :param
              model_path: Ruta y nombre del fichero *.pickle que contiene
el modelo entrenado
              data_path: Ruta y nombre del fichero *.csv que contiene los
datos desconocios para el modelo, a partir
              del cual se van a realizar las predicciones

          :return
              model: Modelo entrenado
              dfDAT: Dataframe a partir del cual se van a realizar las
predicciones
      """
    # Open the file in binary mode
    with open(model_path, 'rb') as file:
        # Call load method to deserialze
        model = pickle.load(file)
    model = model.model
```

```
    dfDAT = pd.read_csv(archivo_csv, header=None)
    #dfDAT=archivo_csv

#=====================================================================
=========================================================

    """
        Se trata de una función que convierte el Dataframe en un numpy
array

            :param
                dfDAT: Dataframe que contiene los datos leidos del fichero
*.csv.

            :return
                X: numpy array que contiene los datos de las variables
independientes
        """

    X_test = dfDAT.iloc[:,:].values

    #
=====================================================================
=========================================

    def rotate(image, HEIGHT, WIDTH):
        """
        Se trata de una función que reescala, rota (en el sentido de las
agujas del reloj 90°) y voltea (horizontalmente)
        la imagen para su correcta visualización.

            :param
                image: numpy array que contiene los valores de las
variables independientes de una unica imagen

            :return
                image: numpy array que contiene los valores de las
variables independientes de una unica imagen, tras su
                reescalado, rotación y volteado
        """
        image = image.reshape([HEIGHT, WIDTH])
        image = np.fliplr(image)
        image = np.rot90(image)

        return image


#=====================================================================
=========================================================

    """
        Se realiza el preprocesamiento de los valores de las variables
independientes. Para ello se realiza un reescalado de imagen.

            :param
                X[n de imagenes, n de pixeles]: numpy array 2D que contiene
los datos de las variables independientes.

            :return
                X_rotate[n de imagenes, HEIGHT, WIDTH]: numpy array 3D que
contiene los datos de las variables independientes, tras su reescalado.
```

```
        """
    X_rotate = []
    for image in X_test:
        tmp = rotate(image, HEIGHT, WIDTH)
        X_rotate.append(tmp)

    X_rotate = np.array(X_rotate)


#============================================================================
============================================================
    """
        Se trata de una función que realiza la prediccion de cada imagen, y
devuelve una lista con el valor de las predicciones

            :param
                X [num de imagenes, HEIGTH, WIDTH, 1]: numpy array 4D que
contiene los datos de las variables independientes, preparadas para su
ingesta en el
                modelo
                label_dict: lista creada a modo diccionario para poder
mapear los números con su correspondiente letra ASCII
                model: Modelo entrenado

            :return
                predicted: lista que contiene el valor de las predicciones
    """

    # fig=plt.figure()
    # fig.subplots_adjust(hspace=1,wspace=0.4)
    predicted = []

    for i in range (1,64):
        prediction = model.predict(X_rotate[i-1].reshape(-1, HEIGHT, WIDTH,
1))  # X_rotate empieza en 0, la foto tiene que empezar en 1, por lo que
utilizo [i-1]
        predicted_label = np.argmax(prediction)
        prediction = label_dict[predicted_label]
        predicted.append(prediction)
    #     plt.subplot(7,9,i)
    #     plt.axis('off')
    #     plt.imshow(X_rotate[i-1])
    #     plt.title(prediction)
    # plt.show()
    # print(predicted)
    return predicted

# if __name__ == '__main__':
#     recognition(archivo_csv)
```

Anexo III

Código de implementación

Anexo III. Código de implementación

```
function [handles]=Inicio_Vision(handles)

%% Preprocesamiento de la imagen
% %Sacar foto, transformar, recortar y guardar para panel.
cam=webcam;
fotopanel=snapshot(cam);
%figure(),imshow(fotopanel);
%[x,y]=ginput(4); input_points= [x y]; % Obtener las coordenadas de entrada manualmente
input_points=[2.349999999999999e+02,68;559,68;591,313;192,297]; %P2 Definicion de las coordenadas de entrada
base_points= [0 0; 500 0; 500 250;0 250]; % Establecimiento de dimensiones predefinidas de escalado
t_entorno=cp2tform(input_points, base_points, 'projective'); % Creacion de la estructura de la transformación proyectiva
recta = imtransform(fotopanel,t_entorno); % Se aplica la transformada para enderezar la imagen
%figure, imshow(recta);
%I=imcrop(recta); % Obtener las coordenadas del cuadrado que se va a recortar
fotopanel=imcrop(recta,[469.5 100.5 488 251]); % Recorta la imagen para quedarnos solo con los bloques
%figure(),imshow(fotopanel);
imwrite(fotopanel,'result.jpg');


Fondo robot.
axes(handles.axes298);
background=imread('result.jpg');
imshow(background);
axis off;


% Cambiar de RGB a escala de grises.
fotopanel_gray=rgb2gray(fotopanel);
%figure(),imshow(fotopanel_gray);

bw=imbinarize(fotopanel_gray,0.79);
```

```
%figure(),imshow(bw);

%% Recortar mediante centroides --- Segmentar la imagen completa para obtener cada bloque por separado

numcrop = 0; % Se inicializa la variale a 0, se irá incrementando a medida que se vaya realizando recortes
matriz_csv=zeros(63,784);

s=regionprops(bw,'centroid');
centroids=cat(1,s.Centroid);
s=regionprops(bw,'BoundingBox');
boxes=cat(1,s.BoundingBox);
s=regionprops(bw,'Area');
area=cat(1,s.Area);

handles.vector_coordenadasx = [22 80 139 196 255 314 373 433 490];
handles.vector_coordenadasy = [12; 50; 88; 126; 164; 202; 240];

handles.matriz_posiciones = [1 2 3 4 5 6 7 8 9; % se utiliza para nombrar las imagenes recortadas
                             10 11 12 13 14 15 16 17 18;
                             19 20 21 22 23 24 25 26 27;
                             28 29 30 31 32 33 34 35 36;
                             37 38 39 40 41 42 43 44 45;
                             46 47 48 49 50 51 52 53 54;
                             55 56 57 58 59 60 61 62 63;]


%imshow(bw)

for i=1:length(centroids)
   %hold on
   %plot(centroids(i,1),centroids(i,2),'r*')
   %rectangle('Position',[boxes(i,1),boxes(i,2),boxes(i,3),boxes(i,4)],'EdgeColor','r','LineWidth',2 )
   coordenadax=round(centroids(i,1));
   coordenaday=round(centroids(i,2));
```

```
    %hold off

    % En caso de que el area sea inferior a 20 se considera ruido, esos centroides no serán estudiados

    if  (area(i)>20) && (area(i) < 60)
        croppedImage = imcrop(bw, boxes(i,:)+[-10 -10 20 20]); % Si tiene un área entre 20 y 70
recortamos un cuadrado más grande para disminuir la deformación al hacer el resize
        croppedImage=imresize(croppedImage,[28,28]);

        % Ordenar los centroides
        valor=100;
        for v1=1:length(handles.vector_coordenadasx) % para obtener la columna a la que pertenece el
centroide
            valornuevox=abs(coordenadax-handles.vector_coordenadasx(1,v1));
            if valornuevox<valor
                posx=v1;
            end
            valor=valornuevox;
        end
        valor=100;
        for v2=1:length(handles.vector_coordenadasy) % para obtener la fila a la que pertenece el
centroide
            valornuevoy=abs(coordenaday-handles.vector_coordenadasy(v2,1));
            if valornuevoy<valor
                posy=v2;
            end
            valor=valornuevoy;
        end

        numcrop=handles.matriz_posiciones(posy,posx); % Se obtiene el numero correspondiente a la
posición del centroide

        val = num2str(numcrop);
        formato = '.jpg';
        nombre_guardar = strcat(val,formato);
        ruta='centroid/';
```

```
        ruta_guardar=strcat(ruta,nombre_guardar); % Se crea la ruta donde se va a guardar la imagen
        imwrite(croppedImage,ruta_guardar);
        let=reshape(croppedImage,[1,784]); % Convierto la matriz de 28x28 a un vector de 1x784
        for columna=1:784
            matriz_csv(numcrop,columna)=let(columna);
        end
    elseif (area(i) >= 60)
        croppedImage = imcrop(bw, boxes(i,:)+[-5 -5 10 10]);
        croppedImage=imresize(croppedImage,[28,28]);

         % Ordenar los centroides
        valor=100;
        for v1=1:length(handles.vector_coordenadasx) % para obtener la columna a la que pertenece el
centroide
            valornuevox=abs(coordenadax-handles.vector_coordenadasx(1,v1));
            if valornuevox<valor
                posx=v1;
            end
            valor=valornuevox;
        end
        valor=100;
        for v2=1:length(handles.vector_coordenadasy) % para obtener la fila a la que pertenece el
centroide
            valornuevoy=abs(coordenaday-handles.vector_coordenadasy(v2,1));
            if valornuevoy<valor
                posy=v2;
            end
            valor=valornuevoy;
        end

        numcrop=handles.matriz_posiciones(posy,posx); % Se obtiene el numero correspondiente a la
posición del centroide

        val = num2str(numcrop);
        formato = '.jpg';
        nombre_guardar = strcat(val,formato);
```

```
        ruta='centroid/';
        ruta_guardar=strcat(ruta,nombre_guardar); % Se crea la ruta donde se va a guardar la imagen
        imwrite(croppedImage,ruta_guardar);
        let=reshape(croppedImage,[1,784]); % Convierto la matriz de 28x28 a un vector de 1x784
        for columna=1:784
            matriz_csv(numcrop,columna)=let(columna);
        end
    end
end
csvwrite('letters_to_predict.csv',matriz_csv);


%% Ejecución función de predicción de python
pathToPredict = fileparts(which('predict.py'));        % Obtención del path donde se encuentra el
archivo python
if count(py.sys.path,pathToPredict) == 0
    insert(py.sys.path,int32(0),pathToPredict);
end

pyOut = py.makeprediction.recognition(); % Ejecución de la función de python, la cual realiza la
predicción y nos devuelve una
                        % lista de python con el valor de las letras predichas

cadena_letras = string(pyOut); % convertir la lista de python en string

%% Obtención de imagen con predicciones

% Ruta de la carpeta con las 63 imágenes
carpeta_imagenes = 'centroid/';

% Dimensiones de la matriz (7x9)
filas = 7;
columnas = 9;

% Tamaño de cada imagen
ancho_imagen = 150;
```

```
alto_imagen = 150;

% Crear una figura
figure;

for i = 1:63 % Iterar sobre las imágenes y sus títulos
    val = num2str(i);
    formato = '.jpg';
    nombre_abrir = strcat(val,formato);
    nombre_abrir = strcat(carpeta_imagenes,nombre_abrir);
    imagen = imread(nombre_abrir);
    imagen = imresize(imagen, [alto_imagen ancho_imagen]); % Redimensionar la imagen al tamaño
deseado
    titulo = cadena_letras{i}; % Agregar el título (letra)
    subplot(filas, columnas, i);% Crear una subimagen en la posición correspondiente
    imshow(imagen);
    title(titulo, 'FontSize', 20, "Color", 'b');  % Agregar el título (letra) a la subimagen
end

% Ajustar la disposición de las subtramas
set(gcf, 'Position', [18, 25, 1000, 800]);
% Guardar la imagen resultante (opcional)
saveas(gcf, 'imagen_predicciones.jpg');
%% Contabilizar la cantidad de letras que hay
% % Para cuando no haya letras suficientes
a=0; b=0; c=0; d=0; e=0; f=0; g=0; h=0; i=0; j=0; k=0; l=0; m=0;
n=0; o=0; p=0; q=0; r=0; s=0; t=0; u=0; v=0; w=0; x=0; y=0; z=0;
handles.nosepuede=0;

% contar la frecuencia de cada letra de la cadena de letras
a = sum(count(cadena_letras, 'A'));
b = sum(count(cadena_letras, 'B'));
c = sum(count(cadena_letras, 'C'));
d = sum(count(cadena_letras, 'D'));
e = sum(count(cadena_letras, 'E'));
f = sum(count(cadena_letras, 'F'));
```

```
g = sum(count(cadena_letras, 'G'));
h = sum(count(cadena_letras, 'H'));
i = sum(count(cadena_letras, 'I'));
j = sum(count(cadena_letras, 'J'));
k = sum(count(cadena_letras, 'K'));
l = sum(count(cadena_letras, 'L'));
m = sum(count(cadena_letras, 'M'));
n = sum(count(cadena_letras, 'N'));
o = sum(count(cadena_letras, 'O'));
p = sum(count(cadena_letras, 'P'));
q = sum(count(cadena_letras, 'Q'));
r = sum(count(cadena_letras, 'R'));
s = sum(count(cadena_letras, 'S'));
t = sum(count(cadena_letras, 'T'));
u = sum(count(cadena_letras, 'U'));
v = sum(count(cadena_letras, 'V'));
w = sum(count(cadena_letras, 'W'));
x = sum(count(cadena_letras, 'X'));
y = sum(count(cadena_letras, 'Y'));
z = sum(count(cadena_letras, 'Z'));

%% 7. Extracción de las coordenadas de los cubos de la handles.palabra.
handles.palabra= upper(handles.palabra);
handles.caracteres=convertStringsToChars(handles.palabra);

% Para cuando no haya letras suficientes
a1=0; b1=0; c1=0; d1=0; e1=0; f1=0; g1=0; h1=0; i1=0; j1=0; k1=0; l1=0; m1=0;
n1=0; o1=0; p1=0; q1=0; r1=0; s1=0; t1=0; u1=0; v1=0; w1=0; x1=0; y1=0; z1=0;

%Extraemos la frecuencia de cada letra de la handles.palabra.
a1 = sum(count(handles.palabra, 'A'));
b1 = sum(count(handles.palabra, 'B'));
c1 = sum(count(handles.palabra, 'C'));
d1 = sum(count(handles.palabra, 'D'));
e1 = sum(count(handles.palabra, 'E'));
f1 = sum(count(handles.palabra, 'F'));
```

```
g1 = sum(count(handles.palabra, 'G'));
h1 = sum(count(handles.palabra, 'H'));
i1 = sum(count(handles.palabra, 'I'));
j1 = sum(count(handles.palabra, 'J'));
k1 = sum(count(handles.palabra, 'K'));
l1 = sum(count(handles.palabra, 'L'));
m1 = sum(count(handles.palabra, 'M'));
n1 = sum(count(handles.palabra, 'N'));
o1 = sum(count(handles.palabra, 'O'));
p1 = sum(count(handles.palabra, 'P'));
q1 = sum(count(handles.palabra, 'Q'));
r1 = sum(count(handles.palabra, 'R'));
s1 = sum(count(handles.palabra, 'S'));
t1 = sum(count(handles.palabra, 'T'));
u1 = sum(count(handles.palabra, 'U'));
v1 = sum(count(handles.palabra, 'V'));
w1 = sum(count(handles.palabra, 'W'));
x1 = sum(count(handles.palabra, 'X'));
y1 = sum(count(handles.palabra, 'Y'));
z1 = sum(count(handles.palabra, 'Z'));

%Comprobar si es posible escribir la palabra (si hay letras suficientes)

if (a1>a || b1>b || c1>c || d1>d || e1>e || f1>f || g1>g || h1>h || i1>i || j1>j || k1>k ||l1>l || m1>m || n1>n ||
o1>o || p1>p || q1>q || r1>r || s1>s || t1>t || u1>u ||v1>v || w1>w || x1>x || y1>y || z1>z)
   handles.nosepuede=1;
end

[ty,tx]=size(handles.matriz_posiciones);

%Buscar letras.
for i = 1:length(handles.palabra)
   detectada=0;
   for l = length(cadena_letras):-1:1 % Empiezo por el final de la cadena de letras, ya que esas letras
están más cerca de la plataforma pequeña, se reduce el tiempo
      if handles.palabra(i) == cadena_letras(l); % se compara la letra de la palabra con las de la lista
```

```
            cadena_letras(l) = 0; % Para eliminar la letra seleccionada de la lista
            for fil = 1:ty
                for col = 1:tx
                    if l == handles.matriz_posiciones(fil,col) % l es el índice de la letra coincidente de la lista,
se busca ese índice en la matriz para obtener su posición
                        handles.Posicion_x(i)=col;
                        handles.Posicion_y(i)=fil;
                        detectada = 1;
                        break
                    end
                end
                if detectada == 1
                    break;
                end
            end
        end
        if detectada == 1
            break; % Ya se ha encontrado la letra buscada, fuerzo la salida del for interno para buscar la
siguiente
        end
    end
end
```

Anexo IV

Manual de usuario

Anexo IV.I – Carga del proyecto de Sysmac Studio al PLC

En este anexo se explica cómo realizar la carga del proyecto de Sysmac Studio al PLC de Omron que gobierna el robot.

1. El primer paso a realizar es abrir la aplicación de Sysmac Studio.

Figura 40. Logo Sysmac Studio.

2. Escoger la opción de abrir proyecto:

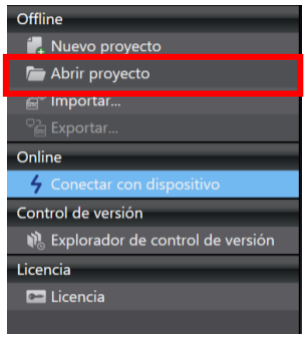


Figura 41. Opción abrir proyecto.

3. Escoger el proyecto “Control_Final_funcional.smc2”
4. Poner en modo “Online”

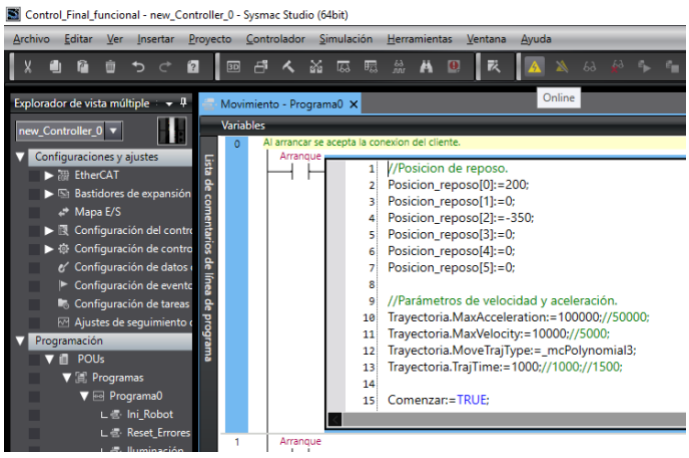


Figura 42. Modo "Online".

5. Transferir al controlador

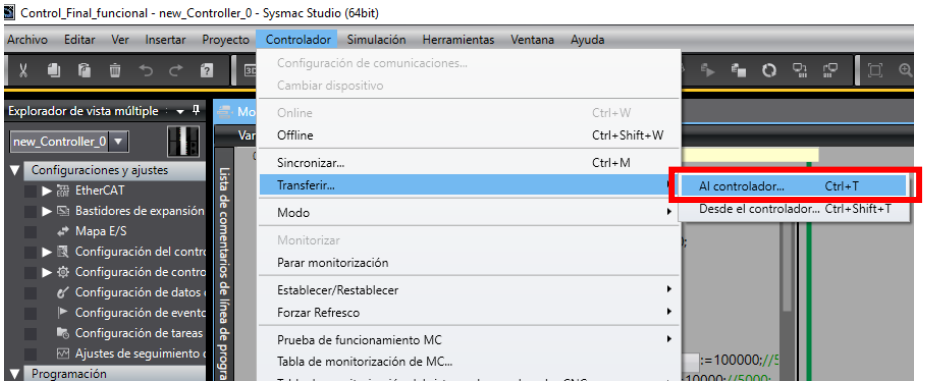


Figura 43. Transferir al controlador.

NOTA: En caso de que tras la transferencia del programa al controlador se vea que el LED rojo del PLC se queda parpadeando, volver a transferir el programa. En ocasiones falla la transferencia.

Anexo IV.II – Gestión de las rutas utilizadas en *makeprediction*

En el archivo *makeprediction.py* se especifican las rutas (path) donde se encuentran el modelo entrenado "model_bw.pickle" y el archivo "emnist-letters-mapping.txt". Por ello, en caso de cambiar la carpeta "Codigo_matlab" de ubicación, será necesario actualizar dichas rutas.

Las nuevas rutas deben reemplazarse por las especificadas en las líneas de código 9 y 14:

```
label_map_path=r'C:\Users\870113\Desktop\TFM_EnaraSaratxaga\Codigo_matlab\emnist-letters-mapping.txt'


model_loadPath=r'C:\Users\870113\Desktop\TFM_EnaraSaratxaga\Codigo_matlab'
```

Anexo IV.III – Calibración de la cámara para la captura de imagen

En caso de que la cámara se mueva y, como consecuencia, no se consiga capturar correctamente la imagen, será necesario volver a calibrar la cámara.

1- Mover manualmente la webcam de forma que se obtenga una imagen en la que la plataforma grande esté lo más centrada y recta posible.
2- Obtener los puntos de entrada: Descomentar las líneas 7 y 8 del código de la función *Inicio_Vision* mostrado en la Figura 44, y en la imagen que se abre al ejecutar el script pulsar con el ratón las esquinas de la plataforma grande en el orden especificado en la Figura 45, logrando así las coordenadas de los puntos de entrada.

```
Inicio_Vision.m
1  function [handles]=Inicio_Vision(handles)
2
3  %% Preprocesamiento de la imagen
4  % %Sacar foto, transformar, recortar y guardar para panel.
5  cam=webcam;
6  fotopanel=snapshot(cam);
7  %figure(),imshow(fotopanel);
8  %[x,y]=ginput(4); input_points= [x y]; % Obtener las coordenadas de entrada manualmente
9  input_points=[2.349999999999999e+02,68;559,68;591,313;192,297]; %P2 Definicion de las coordenadas de entrada
10 base_points= [0 0; 500 0; 500 250;0 250]; % Establecimiento de dimensiones predefinidas de escalado
11 t_entorno=cp2tform(input_points, base_points, 'projective'); % Creacion de la estructura de la transformación proyectiva
12 recta = imtransform(fotopanel,t_entorno); % Se aplica la transformada para enderezar la imagen
13 %figure, imshow(recta);
14 %I=imcrop(recta); % Obtener las coordenadas del cuadrado que se va a recortar
15 fotopanel=imcrop(recta,[469.5 100.5 488 251]); % Recorta la imagen para quedarnos solo con los bloques
16 %figure(),imshow(fotopanel);
17 imwrite(fotopanel,'result.jpg');
18
```

Figura 44. Código a modificar para calibrar la cámara.

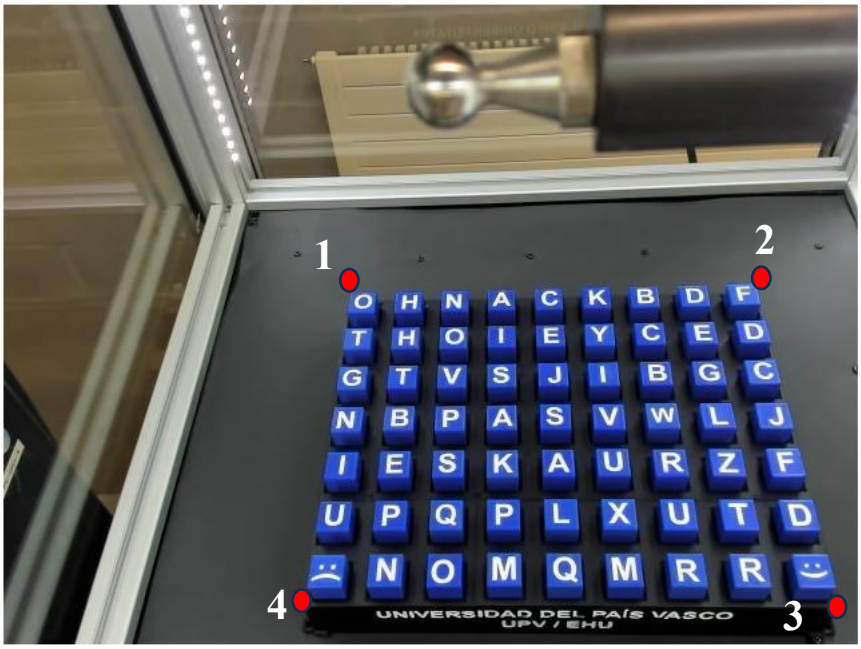


Figura 45. Orden de obtención de los puntos de entrada.

3- Reemplazar los valores de las coordenadas de entrada de la línea de código 9 (input_points), por los valores logrados en el apartado 2.
4- Obtener las coordenadas del cuadrado a recortar: Descomentar las líneas de código 13 y 14, ejecutar el script, y, en la imagen emergente, seleccionar la plataforma grande a través del ratón. Una vez hecho el rectángulo alrededor de la plataforma grande, se da click derecho encima del recuadro y se copian las coordenadas del mismo. Para acabar, se reemplazan las coordenadas de la línea de código 15 por las recientemente obtenidas.
5- Para acabar, volver a comentar las líneas de código 7,8,13 y 14. (Dejarlo igual que en la Figura 44) .

Anexo IV. IV – Alimentación del robot y encendido de la interfaz

PASO 1. Apretar el pulsador verde de la botonera de alimentación para alimentar el robot.

Figura 46. Botonera de alimentación.

PASO 2. Esperar 5 segundos y colocar el selector en la posición superior (3 INIT), cuando el piloto ver se encienda, bajar el selector a la posición inferior (4 RUN).

Figura 47. Selector de posición.

PASO 3. Ejecutar el script “control_planta.m” ubicada en la carpeta “Codigo_matlab”.

PASO 4. Una vez se cargue la aplicación, aparecerá la ventana de la interfaz donde habrá que pulsar en 'START' para comenzar a utilizar la interfaz gráfica.

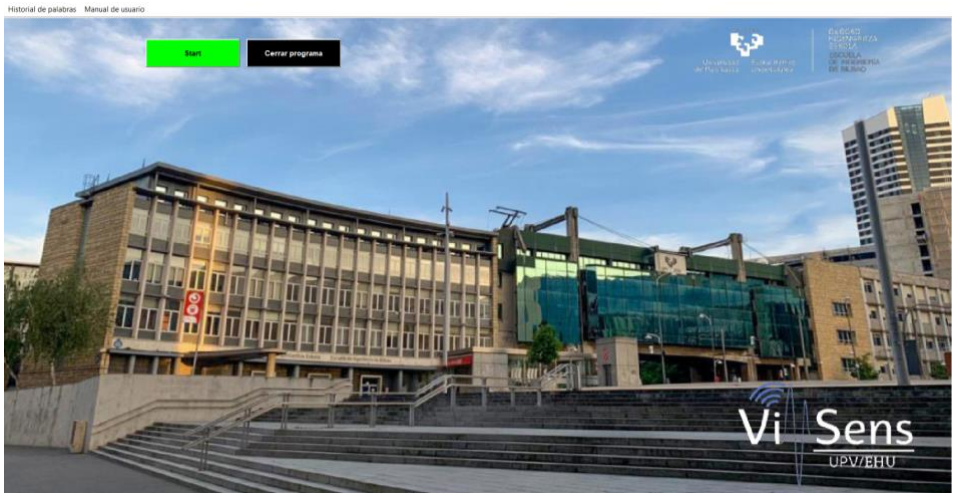


Figura 48. Pantalla inicial de la interfaz.

PASO 5. Después de pulsar 'START', se pueden abrir los paneles deseados pulsando sobre sus botones.

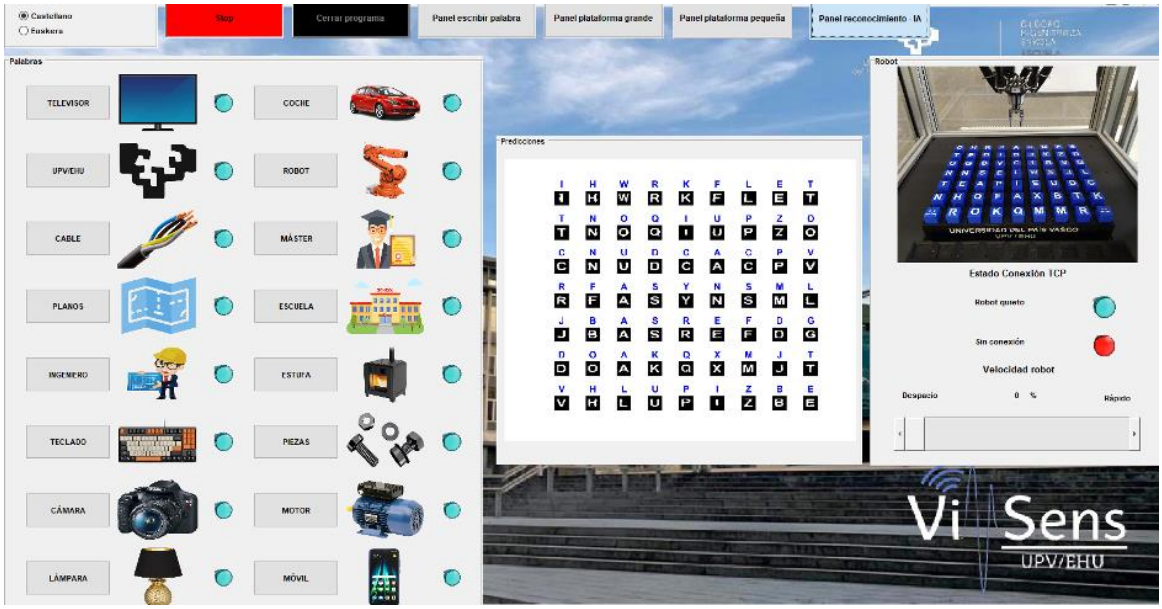


Figura 49. Pantalla de la interfaz de usuario.

REPETIR LOS PASOS 6, 7 Y 8 PARA ESCRIBIR LA SIGUIENTE PALABRA

PASO 6. Primero habrá que ajustar la velocidad del robot (0 – 100%) deseada de la barra de la derecha del panel robot deslizándola.

PASO 7. Con la velocidad seleccionada hay que seleccionar la palabra que se desea escribir, para ello hay dos paneles; el panel 'palabras' donde pulsando sobre los botones de cualquier palabra el robot comenzará a escribir esa palabra.

La segunda forma es desde el panel ‘escribir palabra’ donde se escribirá la palabra deseada en el cuadro en blanco y se pulsará en comprobar. Si la palabra se puede escribir con las letras de la plataforma del robot, este comenzará a moverse, en cambio, si no hay suficientes letras para escribir dicha palabra, mostrará un mensaje de ‘palabra no disponible’ y permitirá escribir otra.

PASO 8. Cuando el robot termine de recoger los cubos después de escribir una palabra, esperar 3 segundos y colocar el selector en la posición superior (3 INIT), esperar un segundo y bajar el selector a la posición inferior (4 RUN).