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

EVALUACIÓN DE REGIONES SIGNIFICATIVAS PARA EL ENTRENAMIENTO DE UNA RED NEURONAL A TRAVÉS DE FOTOGRAFÍAS DE UNA SUPERFICIE

Estudiante: Millán Fernández de Landa, Mikel

Director/Directora: Zulueta Guerrero, Ekaitz

Curso: 2022-2023

Fecha: Bilbao, 16 de Julio del 2023

Resumen

En los últimos años, las redes neuronales convolucionales (CNN) han experimentado un progreso significativo en términos de arquitectura, rendimiento y aplicaciones. Estos avances han impulsado el campo de la visión artificial y han llevado a mejoras en diversas tareas, como el reconocimiento de objetos, la clasificación de imágenes, la detección de anomalías y la segmentación semántica.

La mejora de la arquitectura, el avance en el desarrollo de técnicas que benefician el aprendizaje y el aumento de la capacidad de cómputo han permitido crear redes neuronales más profundas (VGG o ResNet), y han demostrado un mejor rendimiento en la clasificación de imágenes. Sin embargo, la calidad de los datos empleados para el entrenamiento de una red neuronal deben de ser de la mayor posible para obtener mejores resultados. La calidad de los datos dependerá de la información que pueda aportar a la red.

En este trabajo, se entrenara una red neuronal convolucional utilizando fotos de una pared extraídas mediante una cámara posicionada al extremo del robot (Kuka LBR iiwa 7 R800) . La pared tendrá diferentes elementos, y los resultados de este entrenamiento se obtendrá información sobre las zonas de mayor interés para la red neuronal.

Abstract

In recent years, convolutional neural networks (CNNs) have seen significant progress in terms of architecture, performance and applications. These advances have boosted the field of computer vision and have led to improvements in a variety of tasks, such as object recognition, image classification, anomaly detection and semantic segmentation.

Improved architecture, advances in the development of techniques that benefit learning and increased computational capacity have enabled the creation of deeper neural networks (VGG or ResNet), and have demonstrated better performance in image classification. However, the quality of the data used for training a neural network must exhibit the utmost quality to obtain the best results. The quality of the data will be determined by the information that it can feed to the network.

In this work, a convolutional neural network will be trained using photos of a wall obtained using a camera placed on the arm of a robot (Kuka LBR iiwa 7 R800). The wall will have different elements, and the results of this training will provide information about the areas of greatest interest to the neural network.

Laburpena

Azken urteotan, Sare Neuronal Konboluzionalak (CNN) aurrerapen handia izan dute arkitekturari, errendimenduari eta aplikazioei dagokienez. Aurrerapen hauek ordenagailu bidezko ikusmenaren eremua bultzatu dute eta hainbat zereginetan hobekuntzak ekarri dituzte, hala nola, objektuen ezagupena, irudien sailkapena, anomaliak hautematea eta segmentazio semantikoa.

Arkitekturaren hobekuntzak, ikaskuntza mesedegarri duten tekniken garapenean izandako aurrerapenak eta konputazio-ahalmena handitzeak sare neuronal sakonagoak (VGG edo ResNet) sortzea ahalbidetu dute, eta irudien sailkapenean errendimendu hobea erakutsi dute. Hala ere, neurona-sare bat entrenatzeko erabiltzen diren datuen kalitateak ahalik eta kalitaterik handiena izan behar du emaitza hobeak lortzeko. Datuen kalitatea sareari eman liezaiokeen informazioaren araberakoa izango da.

Lan honetan, sare neuronal konboluzional bat lantzen da errobot baten (Kuka LBR iiwa 7 R800) izkinean kokatutako kamerak lortutako horma baten argazkiak erabiliz. Hormak elementu desberdinak izango ditu, eta prestakuntza horren emaitzek sare neuronalerako interes handiena duten arloei buruzko informazioa emango dute.

Palabras clave:

Redes Neuronales Convolucionales, Visión artificial, Cinemática, Kuka LBR iiwa 7 R800, Detección de patrones.

Índice

Lista de tablas

Lista de ilustraciones

Acrónimos

TFM Trabajo de Fin de Master

CNN Red Neuronal Convolucional

MSE Error Cuadrático Medio

1 Introducción y contexto

1.1 Introducción

El campo de la inteligencia artificial y el aprendizaje automático ha experimentado un crecimiento significativo en los últimos años. Las redes neuronales convolucionales (CNN, por sus siglas en inglés) han demostrado ser especialmente eficaces en la tarea de análisis de imágenes [5], permitiendo el reconocimiento de patrones y la extracción de características relevantes. Sin embargo, la calidad y diversidad del conjunto de datos utilizados para entrenar estas redes juegan un papel crucial en su desempeño y capacidad de generalización [17].

En este trabajo de fin de máster (TFM), nos centramos en mejorar la eficiencia y precisión del entrenamiento de una red neuronal utilizando fotografías de una superficie con elementos de interés, con el objetivo de evaluar las regiones más significativas para su aprendizaje. Específicamente, nos enfocamos en el contexto de un robot de 6 grados de libertad y la capacidad de este para reconocer y seguir objetos en una pared.

Para ello, el conjunto de datos con el que se entrenará la red neuronal se obtendrá mediante una cámara situada en el extremo de un brazo robótico. Anteriormente, se tendrá que implementar un control de posición para un robot de 6 grados de libertad.

1.2 Contexto

En los últimos años, el campo de la visión por computadora ha experimentado un rápido avance gracias al desarrollo de técnicas basadas en redes neuronales convolucionales[5]. Uno de los principales avances ha sido la introducción de arquitecturas más profundas y complejas. Las CNN tradicionales consistían en unas pocas capas convolucionales seguidas de capas de agrupación y capas completamente conectadas. Sin embargo, en los últimos años se han desarrollado arquitecturas más profundas, como la red neuronal VGG, la red neuronal residual (ResNet) y la red neuronal Inception, entre otras [12]. Otro avance importante ha sido el desarrollo de técnicas de regularización y normalización, como la normalización de lotes (batch normalization). Estas técnicas ayudan a reducir el sobreajuste (overfitting) y mejoran la generalización de los modelos entrenados [13].

Además de los avances en la arquitectura de las CNN, también ha habido mejoras en las técnicas de entrenamiento. Un ejemplo es el aprendizaje transferido, donde las redes pre-entrenadas se utilizan como punto de partida para tareas relacionadas [34]. Esta técnica ha demostrado ser eficaz para obtener resultados sólidos con conjuntos de datos más pequeños.

En cuanto a las aplicaciones, las CNN han sido ampliamente adoptadas en diversas industrias. En la medicina, se han utilizado para el diagnóstico de enfermedades a partir de imágenes médicas, como la detección de cáncer en mamografías. En el campo de los vehículos autónomos, se utilizan para la detección y clasificación de objetos en tiempo real. También se han aplicado en la industria de la seguridad, el análisis de vídeo, la realidad aumentada y muchas otras áreas.

Uno de los desafíos en el entrenamiento de una red neuronal para realizar tareas de visión por computadora es la capacidad de la red para identificar y centrarse en las regiones más relevantes de una imagen. En algunos casos, las imágenes pueden tener características irrelevantes que pueden afectar negativamente el rendimiento de la red, y en otros casos regiones muy complejas pueden resultar difíciles para el aprendizaje de la red.

En este contexto, el presente trabajo se enfoca en la evaluación de las regiones más relevantes presentes en una superficie mediante fotografías. Estas regiones tendrán características de distinta complejidad, y se evaluará la capacidad de aprendizaje de una red neuronal de las mismas.

2 Alcance y objetivos

El presente trabajo tiene como objetivo evaluar mediante el entrenamiento de una red neuronal cuales son las regiones más significativas en el aprendizaje. Para lograrlo, se tendrán que realizar diferentes tareas que se comentarán a continuación.

En primer lugar, para obtener las fotos necesarias para el entrenamiento de la red neuronal, se deben de realizar las funciones que permitan controlar la posición y orientación que adoptará el robot. Para ello, se debe resolver el problema cinemático directo e inverso del robot utilizado (KUKA LBR IIWA 7 R800). Para poner a prueba las funciones creadas y para la familiarización del uso de la cámara, se realizará un algoritmo que permita al robot cambiar de posición y de orientación en función de las necesidades del momento.

En segundo lugar, se procederá a obtener las fotos y a segmentarlas en imágenes de menor tamaño para que la red neuronal pueda utilizarlas como datos de entrada de la red. Para conseguirlo, primero se deberá crear un programa que permita al robot moverse por toda la pared que se quiere analizar y sacar fotos en cada una de las posiciones. Una vez obtenidas las fotos, se debe de crear un programa que divida y clasifique las fotos en función de si las fotos se utilizarán para entrenar, validar o testear la red neuronal.

En tercer lugar, se debe de entrenar la red neuronal. Para ello, se utilizarán técnicas que mejorarán el entrenamiento de la red neuronal. Una de estas técnicas será el uso del aprendizaje transferido [34], que consiste en utilizar una red neuronal previamente entrenada y modificar únicamente las últimas capas de la red neuronal. Un objetivo secundario del presente trabajo es evaluar la implementación de las distintas técnicas utilizadas.

Por último, se debe crear un programa que permita discutir gráficamente los datos obtenidos. Para ello, se pondrá a prueba la red neuronal creada y se compararán las predicciones de la red con los resultados obteniendo el error en cada una de las zonas de la pared estudiada. Con estos datos se crearán gráficos que permitan discutir los resultados y sacar conclusiones de este trabajo.

3 Antecedentes bibliográficos y estado del arte

3.1 Redes neuronales convolucionales

Las redes neuronales convolucionales (CNN, por sus siglas en inglés) son una clase especializada de redes neuronales artificiales que han revolucionado el campo de la visión por computadora. Estas redes están diseñadas específicamente para obtener un rendimiento mayor al procesar y analizar datos visuales, como imágenes y vídeos, con el objetivo de identificar patrones y características relevantes.

La arquitectura básica de una CNN se compone de varias capas interconectadas, cada una con una función específica en el procesamiento de la información visual. La capa inicial es la capa de entrada, que recibe la imagen en su forma cruda. A medida que la información fluye a través de la red, las capas posteriores realizan operaciones de convolución, agrupamiento (pooling) y activación para extraer gradualmente características más abstractas y significativas [5].

3.1.1 Estructura y los componentes clave de una CNN

Las capas principales de una red neuronal convolucional son:

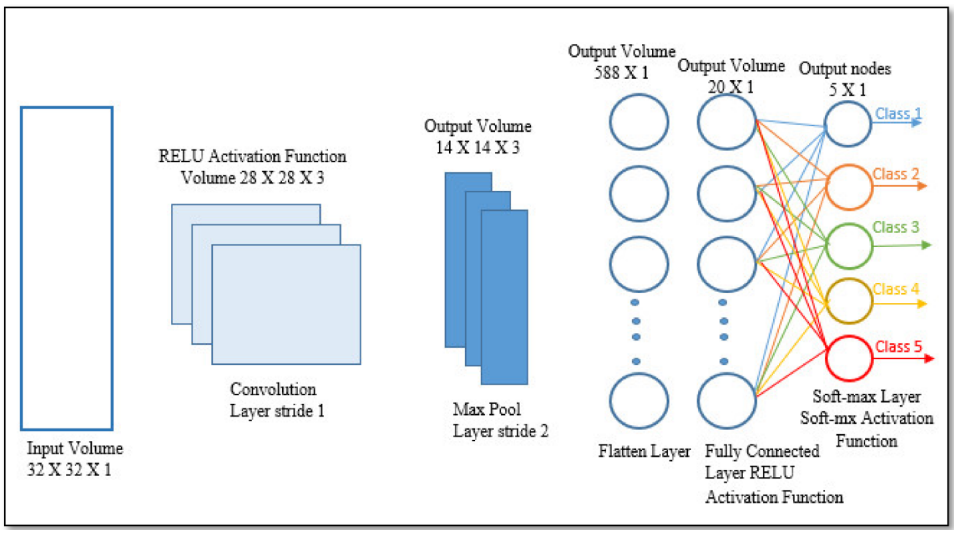


Figure 3.1: Ejemplo de capas de una red neuronal convolucional típica[5].

La capa de convolución: Es una de las partes fundamentales de una CNN. Consiste en aplicar un conjunto de filtros a la imagen de entrada para detectar car-

acterísticas locales, como bordes, esquinas y texturas. Estos filtros se deslizan sobre la imagen y realizan una operación matemática conocida como convolución, que combina la información de píxeles vecinos. Esta operación permite capturar patrones espaciales en la imagen y generar mapas de características convolucionales.

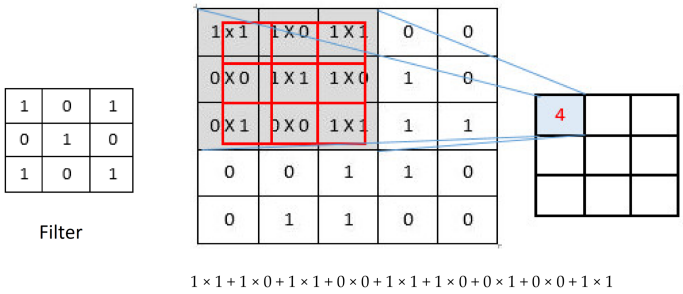


Figure 3.2: Ejemplo de capa convolucinal con filtro [5].

La capa de agrupamiento (pooling): Después de la capa de convolución, la capa de agrupamiento (pooling) reduce la dimensionalidad de los mapas de características, preservando las características más importantes y descartando el ruido o detalles irrelevantes. El agrupamiento se realiza dividiendo la imagen en regiones solapadas y tomando el valor máximo o promedio de cada región, lo que disminuye la cantidad de información a procesar en las capas posteriores.

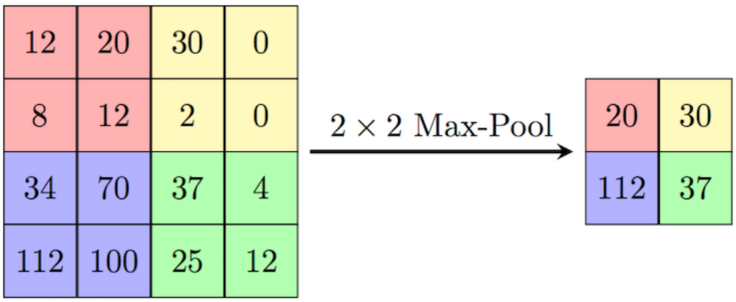


Figure 3.3: Ejemplo de pooling (max)[28].

Las capas totalmente conectadas: reciben las características extraídas y las utilizan para realizar la tarea específica, como la clasificación de imágenes, detección de objetos o reconocimiento facial. Estas capas se asemejan a las de una red neuronal tradicional y utilizan funciones de activación para introducir no linealidades en el modelo y permitir la representación de relaciones complejas entre las características [38].

Además de dichas capas, existen otros elementos, como son las funciones de activación [14]. Estas son funciones matemáticas que se aplican a la salida de una

neurona o unidad de procesamiento en una red neuronal. Estas funciones introducen no linealidad en el modelo, permitiendo a la red aprender y modelar relaciones complejas en los datos. Cuando se procesa una entrada en una neurona, se realiza una combinación lineal de las entradas ponderadas por los pesos sinápticos. Posteriormente, se aplica la función de activación a esta suma ponderada para determinar la salida de la neurona.

3.1.2 Aprendizaje en una red neuronal convolucional

El proceso de aprendizaje en una CNN se basa en la optimización iterativa de una función de pérdida mediante el algoritmo de retropropagación y descenso de gradiente. El objetivo es ajustar los pesos y sesgos de las neuronas de la red para minimizar la diferencia entre las salidas predichas y los valores reales.

En primer lugar, se realiza una pasada hacia adelante (forward pass) por la red, donde los datos de entrenamiento se propagan a través de las capas convolucionales, de agrupamiento y totalmente conectadas, generando una predicción. Luego, se calcula el error entre la predicción y los valores reales utilizando una función de pérdida, como el error cuadrático medio (MSE) o la entropía cruzada.

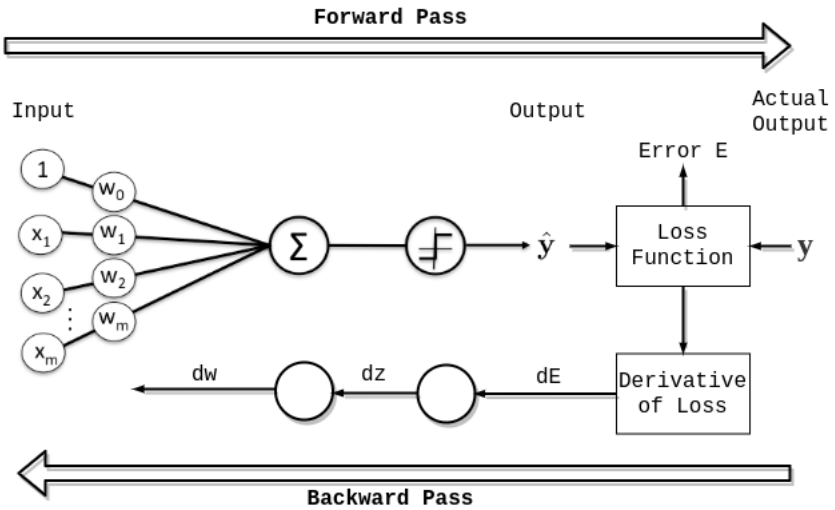


Figure 3.4: Diagrama de forward pass y backward pass [3].

A continuación, se realiza una pasada hacia atrás (backward pass) utilizando el algoritmo de retro-propagación. Durante esta etapa, se calculan las derivadas parciales del error con respecto a los pesos y sesgos de cada neurona de la red. Estas derivadas se utilizan para ajustar los parámetros de la red en dirección opuesta al gradiente del error, mediante el descenso de gradiente [1].

El descenso de gradiente estocástico (3.1) actualiza los pesos y sesgos de la red mediante la multiplicación de la derivada parcial del error con respecto a los parámetros por una tasa de aprendizaje. Esto se repite iterativamente para múltiples ejemplos de

entrenamiento hasta que la función de pérdida se minimiza o se alcanza un criterio de convergencia.

$$\theta_{t+1} = \theta_t - \alpha \nabla E(\theta_t) \tag{3.1}$$

Además, se utilizan técnicas de regularización, en este caso, la regularización L2 (3.2). Estas se usan para evitar que en vez de que la red neuronal aprenda, la red neuronal memorice. Este fenómeno se llama sobre-ajuste (over-fitting) y es perjudicial para el modelo. La regularización permite reducir la complejidad del modelo, lo cual permite mejorar la capacidad del modelo de generalizar.

$$L2(\theta) = \lambda \sum_{j=1}^{n} \theta_j^2 \tag{3.2}$$

El último concepto que se comentará es el de la normalización de los lotes (batch normalization). Esta es una técnica que se utiliza para reducir la dimensión de los cálculos que se realizan para calcular la media del error o la desviación estándar [22].

3.1.3 Importancia de los puntos clave en una imagen

En el procesamiento de imágenes y visión por computadora, los puntos clave (keypoints) desempeñan un papel fundamental para identificar regiones de interés y detectar patrones relevantes en una imagen. Estos puntos clave son puntos distintivos y significativos que se caracterizan por tener propiedades únicas y estables, como cambios en la intensidad, texturas o formas, que los hacen fácilmente detectables en diferentes escalas y orientaciones. Los métodos más utilizados son el SIFT y el SURF.

El SIFT utiliza una pirámide de imágenes en diferentes escalas para buscar puntos clave en distintos niveles de detalle [37]. Luego, para cada punto clave detectado, se calculan descriptores que codifican la información local de su vecindario, como gradientes de intensidad y orientación. Estos descriptores son invariantes a las transformaciones mencionadas y pueden ser utilizados para comparar y emparejar puntos clave entre diferentes imágenes [15].

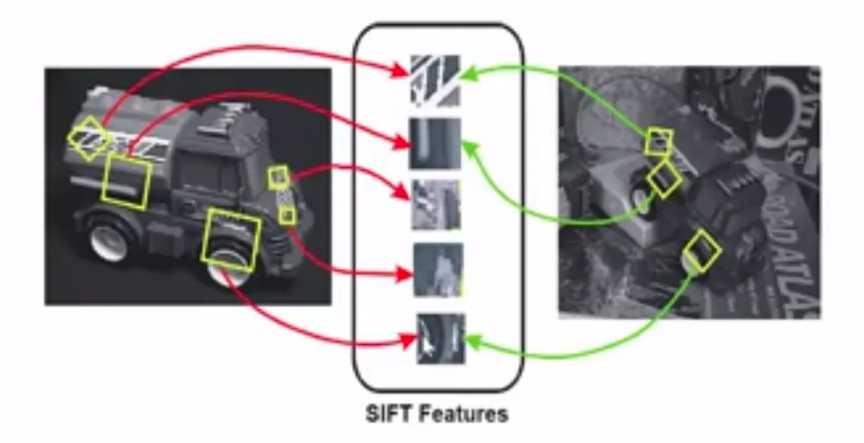


Figure 3.5: Ejemplo de SIFT [27].

El SURF es un algoritmo similar a SIFT pero más rápido y eficiente computacionalmente [30]. Al igual que SIFT, SURF también detecta puntos clave invariantes a cambios de escala, rotación y cambios en la iluminación. Sin embargo, SURF utiliza filtros de caja y una matriz hessiana aproximada para acelerar el cálculo de los descriptores, lo que lo hace especialmente adecuado para aplicaciones en tiempo real [29].

La importancia de los puntos clave radica en su capacidad para identificar regiones de interés y detectar patrones relevantes en una imagen. Al encontrar puntos clave distintivos y estables, se pueden localizar características específicas, como esquinas, bordes o texturas, que son fundamentales para reconocer objetos o realizar tareas de análisis visual [24].

3.1.4 Patrones naturales más fáciles de aprender para una CNN

Las redes neuronales convolucionales (CNN) han demostrado ser altamente eficientes en el aprendizaje y detección de patrones naturales en imágenes [5]. Algunos de los patrones visuales más comunes y fáciles de aprender para una CNN incluyen bordes, texturas simples, esquinas y puntos de interés. Estos patrones son ampliamente estudiados en la literatura y se ha demostrado que las CNN tienen una gran capacidad para aprender y detectar eficientemente estos patrones naturales. A continuación, se discutirán estos patrones y se mencionarán estudios previos relevantes.

Los bordes son transiciones abruptas de intensidad en una imagen y son uno de los patrones visuales más básicos [35]. En una imagen, los bordes representan cambios significativos en la intensidad de los píxeles y pueden corresponder a límites entre objetos, contornos o formas. Las CNN pueden aprender a detectar bordes mediante el uso de filtros convolucionales que responden a cambios de intensidad

en diferentes direcciones. Estos filtros convolucionales se utilizan para capturar características locales en la imagen y resaltar los bordes.

Las texturas simples se refieren a patrones repetitivos y regulares en una imagen, como rayas, puntos o cuadros. Estas texturas tienen una estructura definida y pueden ser fácilmente aprendidas por una CNN [18]. Al utilizar múltiples filtros convolucionales, una CNN puede capturar diferentes características texturales en la imagen y detectar patrones como líneas paralelas, puntos dispersos o áreas con texturas uniformes [8].

Las esquinas y los puntos de interés son características distintivas en una imagen que representan cambios bruscos en la orientación de los bordes. Estos puntos son fácilmente identificables y proporcionan información valiosa sobre la estructura y contenido de la imagen [4]. Las CNN pueden aprender a detectar esquinas y puntos de interés utilizando filtros convolucionales diseñados para responder a cambios de orientación en los bordes. Estos filtros ayudan a resaltar y localizar puntos clave en la imagen.

3.1.5 Métodos y técnicas para mejorar el aprendizaje de patrones

El aprendizaje de patrones complejos es un desafío importante en el entrenamiento de redes neuronales convolucionales (CNN). Afortunadamente, existen varios enfoques avanzados y técnicas que se han desarrollado para mejorar la capacidad de una CNN para aprender patrones complejos de manera más eficiente. En este caso se utilizarán dos técnicas ampliamente utilizadas: el uso de una arquitectura más profunda (en este caso el VGG16) y el aprendizaje transferido .

Utilizar una arquitectura de red neuronal más profunda tiene varios beneficios. En primer lugar, las redes neuronales más profundas tienen una mayor capacidad de capturar y aprender representaciones más complejas y abstractas de los datos. Además, tienen el potencial de mejorar el rendimiento. Por otro lado, una red neuronal pre-entrenada más profunda puede ayudar a regularizar el modelo y reducir el sobre-ajuste. Sin embargo, la razón principal del uso de esta red es la transferencia de características. Al re-entrenar una red neuronal profunda, especialmente en tareas relacionadas o conjuntos de datos similares, se pueden transferir las características aprendidas previamente. Las capas iniciales de una red pre-entrenada suelen ser responsables de extraer características de bajo nivel, como bordes y texturas simples, que son relevantes para una amplia gama de problemas. Al aprovechar estas características pre-entrenadas, se acelera el proceso de aprendizaje y se mejora la capacidad de generalización del modelo [32].

En este caso, solo se van a re-entrenar las últimas capas de la red neuronal. A esta técnica se la conoce como "ajuste fino" o "transfer learning" en inglés. Las últimas capas de una red neuronal suelen ser responsables de la extracción de características de alto nivel, que son más específicas para el conjunto de datos utilizado en el entrenamiento original. Estas capas contienen información que es más relevante para la tarea específica que se desea abordar [21].

3.1.6 Casos de estudio y aplicaciones

Las redes convolucionales han demostrado ser una de las técnicas más poderosas en el campo del reconocimiento de imágenes.

Esto ha permitido que se haya aplicado en todo tipo de sectores [20]. Uno de los sectores que más se está investigando e implementando es el sector de la medicina. Se está aplicando en diferentes campos, como puede ser la cirugía [23] o la identificación de enfermedades a través de imágenes [6]. También se está investigando a cerca del reconocimiento facial y los aspectos más abstractos de la misma como la detección de emociones [36].

Otro sector que está avanzando rápidamente en la implementación de la visión artificial es el sector industrial. Se puede aplicar de muchas formas, desde la detección de piezas defectuosas en una línea de producción [11] hasta la combinación de la visión artificial con la robótica, para saber situar el robot en el lugar adecuado en todo momento.

Otro ámbito en el que se está investigando es el de los vehículos autónomos. Se lleva décadas investigando sobre la conducción autónoma, y la visión artificial para la detección de carriles o para la detección de otros vehículos y señales ya está siendo implementada [19]. Además, ya se pueden encontrar los primeros modelos que son capaces de estacionar sin que el conductor tenga que intervenir en el proceso.

3.2 Robot de 6 grados de libertad

Para la realización de este proyecto, se va a tener que utilizar un robot de 6 grados de libertad. Los comandos y el control utilizado hasta ahora han causado errores y no permiten hacer lo que se quiere realizar en este proyecto. Por lo tanto, se va a hacer un control de posición para este robot en Matlab. Se estudiará, por lo tanto, la cinemática directa e inversa del modelo LBR iiwa 7 R800. Este robot tiene 7 grados de libertad, pero se bloqueará uno de ellos para que se simplifiquen las ecuaciones y resulten más fáciles los cálculos.

Figure 3.6: Foto del robot empleado.

3.2.1 Cinemática directa

La cinemática directa consiste en determinar la posición y orientación del extremo del robot en función de las posiciones de sus articulaciones. Para lograrlo, se utilizan transformaciones geométricas y matrices de transformación homogénea.

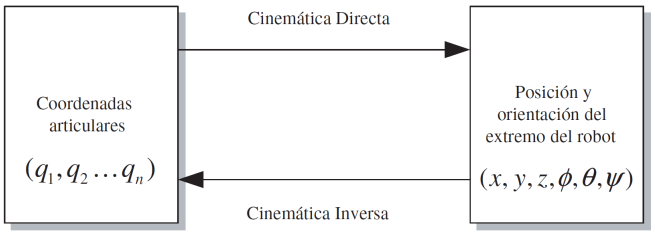


Figure 3.7: Diagrama de relación entre cinemática directa e inversa [2].

Cada articulación del robot tiene una posición y orientación específica, y estas se combinan sucesivamente para obtener la matriz de transformación total. Esta matriz representa la posición y orientación del extremo del robot en relación con un sistema de coordenadas de referencia.

Existen varios métodos para calcular la cinemática directa de un robot de seis grados de libertad, como el método Denavit-Hartenberg y el método de la matriz

de transformación homogénea. Estos métodos implican definir los sistemas de coordenadas locales de cada articulación, establecer los parámetros de Denavit-Hartenberg y utilizar ecuaciones de transformación para obtener la matriz total [16].

Una vez obtenida la matriz de transformación, se pueden extraer los valores de posición (coordenadas XYZ) y orientación (ángulos de Euler) del extremo del robot. Estos valores proporcionan información precisa sobre la ubicación y la orientación del robot en el espacio.

3.2.2 Cinemática Inversa

La cinemática inversa se refiere al proceso de determinar las configuraciones de las articulaciones de un robot necesario para lograr una posición y orientación específicas del extremo del robot, es decir, para mover el extremo del robot a una ubicación deseada en el espacio. En el caso de un robot de 6 grados de libertad implica encontrar los valores de las articulaciones que permitirán que el robot alcance una posición y orientación objetivo [7].

Existen diferentes métodos para abordar el problema de la cinemática inversa, pero en este proyecto se ha optado por el método geométrico. Este procedimiento se basa en encontrar relaciones geométricas en las que se intervendrán las coordenadas del extremo del robot. Este procedimiento es adecuado para robots de pocos grados de libertad, pero en este caso existe una forma de simplificar el problema, gracias al desacoplo cinemático [33].

Este método es posible aplicarlo cuando los últimos tres grados de libertad se cortan en un punto o existe muy poca distancia entre ellos, como es el caso. Este método divide el problema en dos partes. En primer lugar, posiciona el robot en el lugar indicado con los primeros tres ejes (q_1, q_2 y q_4 en el caso de este robot, ya que el eje bloqueado será q_3), y una vez situado el robot en la posición deseada, se orientará con los últimos tres ejes del robot (q_5, q_6 y q_7).

Una vez resueltos los problemas de la cinemática directa e inversa, se podrá hacer un programa que vaya cambiando la posición y la orientación del robot de la forma deseada.

4 Desarrollo de la solución

Para llevar acabo este proyecto, primero hacen falta unas funciones que permitan mover el robot LBR iiwa 7 R800 como se desea. Hasta ahora, se habían utilizado las funciones proporcionadas por el Toolbox KST-Kuka-Sunrise [9], pero este ha resultado ser en algunos casos problemáticos. El controlador del robot mandaba muchas veces señales de error, a veces en situaciones donde a priori no debería de haber ningún problema. Por lo tanto, se decide hacer un nuevo control de posición diseñado en Matlab. Aun así, esta toolbox se seguirá usando para establecer conexión entre Matlab y el robot.

Para comprobar si dichas funciones funcionan o no, se van a implementar para que, con la cámara fijada en el extremo del robot, este sea capaz de seguir un AprilTag. Con una función de Matlab, esta mandará la orientación y la traslación respecto del robot, y el robot se moverá para quedarse en frente del Apriltag .

4.1 Seguimiento de brazo robótico a marcador Tag

4.1.1 Organización del algoritmo

La secuencia esta distribuida en dos partes diferenciadas: la parte de inicialización y poner el robot en la posición inicial y la parte del loop. Una vez acabada la parte de inicialización, el programa entra en un loop que acaba cuando se cumpla la condición establecida. En la siguiente imagen se puede observar el diagrama del programa.

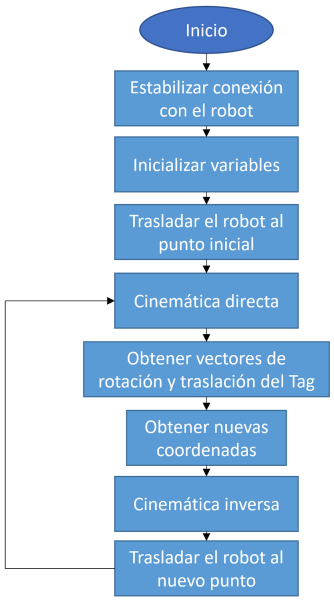


Figure 4.1: Secuencia del programa.

4.1.2 Conexión con el robot, inicialización de variables y punto inicial

Antes de establecer conexión con el robot, se depura el programa mediante los comandos "close all", "clear all" y "clc".

A continuación, se procede a establecer una conexión entre Matlab y el robot. Mediante el comando “netEstablishConnection()” se le vincula la dirección IP del robot. Por último, se comprueba que no haya surgido ningún problema en la conexión. De no ser así, se manda un mensaje de error y el programa terminará.

```
close all;
clear all;
clc;
%% Estabilizar conexion con Robot
warning('off');
ip='172.31.1.147'; % IP del Robot
% Comienza la conexión con el servidor
global t_Kuka;
t_Kuka=net_establishConnection( ip );

if ~exist('t_Kuka','var') || isempty(t_Kuka) || strcmp(t_Kuka.Status,'closed')
  warning('Connection could not be establised, script aborted');
  return; %en caso de no detectar el robot, el programa se para inmediatamente
else
```

Figure 4.2: Conexión entre Matlab y el Robot.

Una vez establecida la conexión, se inicializan los valores del giro de cada eje en radianes para la posición inicial. Después, se introduce la velocidad relativa a la que

se moverá el robot respecto a la velocidad máxima a la que se podría mover. Este valor será un número entre 0 y 1.

Por último, se mueve el robot a la posición de reposo, y después, a la posición que se ha inicializado con los valores de los ejes anteriores.

```
%% Inicialización de variables
    q1 = 0.62;
    q2 = -1.36;
    q3 = 0;
    q4 = 0;
    q5 = 0;
    q6 = 0;
    q7 = 0;
    Deltax = 0;
    Deltay = 0;
    Deltaz = 0;
%% Inicio movimiento

    % Posicion de descanso
    relVel=0.25; % sobrepasar velocidad relativa de las articulaciones
    pos={0, 0, 0, 0, 0, 0, 0};
    movePTPJointSpace( t_Kuka , pos, relVel);

    % Posicion de inicio

    pos={q1, q2, q3, q4, q5, q6, q7};
    movePTPJointSpace( t_Kuka , pos, relVel);
```

Figure 4.3: Parte del programa donde se sitúa el robot en la posición deseada para empezar con el loop.

Una vez acabado esto, el programa entra en el loop. A partir de aquí se analizará cada función utilizada.

4.1.3 Cinemática Directa

La función de la cinemática directa es una de las dos funciones fundamentales para el control de posición del robot. Al llamar a esta función, introduciendo los valores del giro de los ejes del robot en radianes, devolverá la matriz homogénea R_7^0. Esta contiene la información necesaria para obtener la posición y la orientación del extremo del robot respecto a la base.

```
[R07Directo]= CinematicaDirecta (q1, q2, q3, q4, q5, q6, q7);
```

Figure 4.4: Llamada a la función de cinemática directa.

El Robot Kuka iiwa R800 está orientado por los ángulos de euler, y el origen de las coordenadas están situadas en la base. Este Robot tiene además 7 articulaciones,

por lo tanto la posición y la orientación del extremo del robot vendrá dada por las relaciones:

$$\begin{aligned} x &= f_x(q_1, q_2, q_3, q_4, q_5, q_6, q_7) \\ y &= f_y(q_1, q_2, q_3, q_4, q_5, q_6, q_7) \\ z &= f_z(q_1, q_2, q_3, q_4, q_5, q_6, q_7) \\ \phi &= f_\alpha(q_1, q_2, q_3, q_4, q_5, q_6, q_7) \\ \theta &= f_\beta(q_1, q_2, q_3, q_4, q_5, q_6, q_7) \\ \psi &= f_\gamma(q_1, q_2, q_3, q_4, q_5, q_6, q_7) \end{aligned} \tag{4.1}$$

El modelo cinemático directo se ha obtenido mediante el método de Denavit-Hartenberg. Este método sirve para relacionar la articulación $i-1$ a la articulación i.

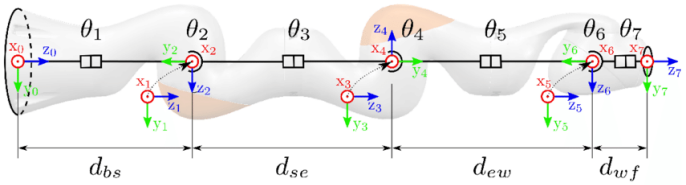


Figure 4.5: Diagrama del robot Kuka iiwa R800 [31].

Los parámetros de Denavit-Hartenberg son los siguientes:

i	θ_i	d_i	a_i	α_i
1	θ_1	d_{bs}	0	$-\frac{\pi}{2}$
2	θ_2	0	0	$\frac{\pi}{2}$
3	θ_3	d_{se}	0	$\frac{\pi}{2}$
4	θ_4	0	0	$-\frac{\pi}{2}$
5	θ_5	d_{ew}	0	$-\frac{\pi}{2}$
6	θ_6	0	0	$\frac{\pi}{2}$
7	θ_7	d_{wf}	0	0

Table 4.1: Parámetros de DH, donde $d_{bs} = 340mm$, $d_{se} = 400mm$, $d_{ew} = 400mm$ y $d_{wf} = 126mm$.

Con la tabla de DH se obtienen las matrices de transformación homogénea, siguiendo la siguiente matriz:

$$T_i^{i-1} = \begin{bmatrix} cos(\theta_i) & -sin(\theta_i)cos(\alpha_i) & sin(\theta_i)sin(\alpha_i) & a_i cos(\theta_i) \\ sin(\theta_i) & cos(\theta_i)cos(\alpha_i) & -cos(\theta_i)sin(\alpha_i) & a_i sin(\theta_i) \\ 0 & sin(\alpha_i) & cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

Por lo tanto, las matrices quedan:

$$T_1^0 = \begin{bmatrix} cos(\theta_1) & 0 & -sin(\theta_1) & 0 \\ sin(\theta_1) & 0 & cos(\theta_1) & 0 \\ 0 & -1 & 0 & d_{bs} \\ 0 & 0 & 0 & 1 \end{bmatrix} T_2^1 = \begin{bmatrix} cos(\theta_2) & 0 & sin(\theta_2) & 0 \\ sin(\theta_2) & 0 & -cos(\theta_2) & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
$$T_3^2 = \begin{bmatrix} cos(\theta_3) & 0 & sin(\theta_3) & 0 \\ sin(\theta_3) & 0 & -cos(\theta_3) & 0 \\ 0 & 1 & 0 & d_{se} \\ 0 & 0 & 0 & 1 \end{bmatrix} T_4^3 = \begin{bmatrix} cos(\theta_4) & 0 & -sin(\theta_4) & 0 \\ sin(\theta_4) & 0 & cos(\theta_4) & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.3)$$
$$T_5^4 = \begin{bmatrix} cos(\theta_5) & 0 & -sin(\theta_5) & 0 \\ sin(\theta_5) & 0 & cos(\theta_5) & 0 \\ 0 & -1 & 0 & d_{ew} \\ 0 & 0 & 0 & 1 \end{bmatrix} T_6^5 = \begin{bmatrix} cos(\theta_6) & 0 & sin(\theta_6) & 0 \\ sin(\theta_6) & 0 & -cos(\theta_6) & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
$$T_7^6 = \begin{bmatrix} cos(\theta_7) & 0 & -sin(\theta_7) & 0 \\ sin(\theta_7) & 0 & cos(\theta_7) & 0 \\ 0 & -1 & 0 & d_{wf} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Así pues, se puede obtener la matriz T_7^0 que indica la localización del sistema asociado al extremo del robot respecto al sistema de referencia de la base del robot:

$$T_7^0 = T_1^0 T_2^1 T_3^2 T_4^3 T_5^4 T_6^5 T_7^6 = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.4)$$

Esto en el código se traduce de la siguiente forma:

```
function [R07]= CinematicaDirecta (q1, q2, q3, q4, q5, q6, q7)

%Longitudes necesarias del robot
L1 = 340; %mm
L2 = 400;
L3 = 400;

R01 = [cos(q1) 0 -sin(q1) 0 ; sin(q1) 0 cos(q1) 0 ; 0 -1 0 L1 ; 0 0 0 1];
R12 = [cos(q2) 0 sin(q2) 0 ; sin(q2) 0 -cos(q2) 0 ; 0 1 0 0 ; 0 0 0 1];
R23 = [cos(q3) 0 sin(q3) 0 ; sin(q3) 0 -cos(q3) 0 ; 0 1 0 L2; 0 0 0 1];
R34 = [cos(q4) 0 -sin(q4) 0 ; sin(q4) 0 cos(q4) 0 ; 0 -1 0 0 ; 0 0 0 1];
R45 = [cos(q5) 0 -sin(q5) 0 ; sin(q5) 0 cos(q5) 0 ; 0 -1 0 L3 ; 0 0 0 1];
R56 = [cos(q6) 0 sin(q6) 0 ; sin(q6) 0 -cos(q6) 0 ; 0 1 0 0; 0 0 0 1];
R67 = [cos(q7) -sin(q7) 0 0 ; sin(q7) cos(q7) 0 0 ; 0 0 1 0 ; 0 0 0 1];

R07 = R01*R12*R23*R34*R45*R56*R67;
end
```

Figure 4.6: Función de cinemática directa.

Un detalle es que no se ha tenido en cuenta la distancia de la muñeca del robot. Esto es porque esta distancia se tendrá en cuenta más adelante.

Con la matriz 4.4 solucionada, la cinemática directa está resuelta. Por lo tanto, en cada iteración se podrán obtener la ubicación y la orientación del extremo del robot.

4.1.4 Obtener vectores de rotación y traslación del Tag

Como se ha explicado previamente, la identificación del Tag se hará mediante una cámara situada en el extremo de robot. Para este cometido, se ha utilizado una función de Matlab llamada readAprilTag(), que está dentro de la función creada cámara.

Esta función es capaz de identificar el número que representa cada Tag y obtiene la matriz de rotación y el vector de traslación. Para la utilización de esta función, previamente se ha tenido que calibrar la cámara con la app de Matlab "Camera Calibrator" [25]. Una vez calibrada la cámara, la sesión de calibración se guarda como "calibrationSession.mat". Aquí se guardan todos los parámetros necesarios de la cámara para que la función readAprilTag() sea capaz de obtener la información del AprilTag.

En la función Camara, por lo tanto, primero se inicializa la cámara con la que se va a trabajar. En este caso siempre sera el 2, ya que el 1 es la cámara del portátil. A continuación recoge la información guardada en la sesión de calibración y recoge de esa sesión la información necesaria de la cámara. Esta es guardada en una variable llamada "intrinsics". Por último, antes del loop de esta función, se determina que tamaño tienen los AprilTAg utilizados. Esta información será necesaria para el cálculo de las distancias y de la orientación.

```
function [MatrizRotacion, VectorTraslacion, id] = Camara
cam = webcam(2);
load('calibrationSession.mat');
intrinsics = calibrationSession.CameraParameters.Intrinsics;
tagSize = 60; % mm
```

Figure 4.7: La inicialización de las variables necesarias en la función Camara para que se pueda leer el AprilTag.

A partir de aquí, la función entra en un bucle. No se saldrá de este bucle hasta que la función haya detectado el AprilTag deseado.

Para ello, primero, dentro del bucle, se saca una foto. A continuación, esa foto es analizada con la función readAprilTag(). Si hay un AprilTag en la foto, la función devolverá el ID (el número) del AprilTag detectado y el "pose". La variable "pose"

tiene como información la matriz de rotación del AprilTag respecto de la cámara y el vector de traslación respecto de la cámara.

La función readAprilTag() tiene cuatro entradas en este caso: La imagen, los "intrinsics", el tamaño del AprilTag y por último, la familia a la que pertenece el AprilTag, que en este caso es "tag36h11" [10]. Existen distintas familias de AprilTag en función de cuales sean las necesidades de la aplicación para la que se vayan a usar. En este caso, se escoge la familia "tag36h11", ya que son las más simples y son las recomendadas para aplicaciones en las que se necesite un tiempo de respuesta menor.

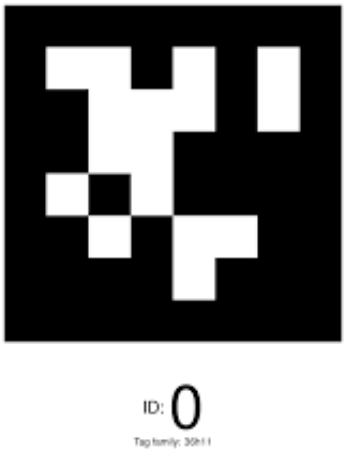


Figure 4.8: ID 0 de la familia de AprilTag 36h11 [10].

La matriz de rotación y el vector de traslación se devuelven de la función de la siguiente forma:

$$R_{3*3} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{4.5}$$

$$P_{3*1} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \tag{4.6}$$

Sin embargo, la matriz y el vector obtenidos no se podrán aplicar directamente, ya que no tienen los mismos ejes que tiene el extremo del robot, por lo tanto se tendrán que hacer unos cambios que se tratarán en la siguiente sección. Estos cambios se realizarán en la siguiente función.

Por lo tanto, el código de loop queda de la siguiente forma:

```
while true

    I=snapshot(cam);
    imshow(I);

    [id,~,pose] = readAprilTag(I,"tag36h11",intrinsics,tagSize);

    for i = 1:length(pose)
        MatrizRotacion=pose(i).Rotation;
        VectorTraslacion=pose(i).Translation;
    end

    if id(i)==2
        break
    end
end
end
```

Figure 4.9: Bucle dentro de la función Camara que busca el ID del AprilTag requerido.

4.1.5 Obtener nuevas coordenadas

Una vez obtenida la matriz de rotación del Tag respecto de la cámara y el vector de traslación, se tienen que juntar los dos componentes para hacer una matriz que se pueda multiplicar a la matriz homogénea previa para así poder obtener las nuevas coordenadas cartesianas respecto de la base y los nuevos ángulos de Euler.

Sin embargo, los ejes no coinciden con los del extremo del robot, y además al girar la cámara los ejes vuelven a cambiar. Esto hace que se tengan que obtener primero los ángulos de Euler de la matriz de rotación del AprilTag, reordenarlas en función de la configuración del robot y crear una nueva matriz de rotación, que esta vez si, al multiplicar dicha matriz con la matriz de rotación R_7^0, se obtenga la matriz de rotación final que dará los ángulos de Euler deseados.

Primero, se tienen que obtener los ángulos de Euler. Se pueden obtener de muchas formas, pero en este caso se utiliza la función de Matlab "rotm2eul()" que los obtiene directamente. Después se reordenan y se vuelve a crear la matriz con la función de Matlab "rotm2eul().

A esta matriz de rotación se la denominará como R_8^7, como si de una extensión del robot se tratase. Se crea una matriz homogénea T_8^7 con dicha matriz de rotación y el vector de traslación reordenado P_8^7.

$$T_8^7 = \begin{bmatrix} R_8^7 & P_8^7 \\ 0 & 1 \end{bmatrix} \tag{4.7}$$

De lo que se deduce:

$$T_8^0 = T_7^0 T_8^7 = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.8}$$

Un problema es que la función que se está utilizando para obtener la matriz y las coordenadas, el readAprilTag() cambia la dirección de las coordenadas en función de si la cámara está boca abajo o no. Por lo tanto, hay que hacer una condición que distinga esto.

Por lo tanto, de la matriz de rotación obtenida se obtienen los ángulos de Euler originales y se cambian en función de la posición del robot.

Por lo tanto, los ángulos de Euler de la cámara se reorganizan, y se hace una matriz homogénea con el vector de traslación también reorganizado. A esta matriz se la llama R_8^7, y si se multiplica la matriz homogénea original con la matriz de la cámara se obtienen los ángulos de Euler y la posición deseados a los que se quiere mover el extremo del robot.

4.1.6 Cinemática Inversa

El problema cinemático inverso trata de encontrar los valores que tienen que adoptar las coordenadas articulares del robot para que el robot se posicione en un punto y con una orientación definida.

Para ello, se ha diseñado una función en Matlab llamada CinematicaInversa(). Las entradas de la función son los ángulos de Euler del extremo del robot (a, b, c) y la posición del extremo del robot en coordenadas cartesianas (x, y, z). Las salidas serán el giro de los siete ejes del robot en radianes para que el extremo del robot llegue a la orientación y posición deseadas.

```
[q1, q2, q3, q4, q5, q6, q7]= CinematicaInversa (a, b, c, x, y, z);
```

Figure 4.10: Llamada a la función CinematicaInversa() desde el main.

El problema se ha abordado mediante el método geométrico y utilizando el desacoplo cinemático, por lo tanto primero se abordara el problema de la posición con los primeros cuatro ejes del robot(el tercer eje estará bloqueado) y por último se abordará el problema de la orientación con los últimos tres ejes.

En la función CinematicaInversa(), en primer lugar, se inicializan las longitudes del robot, como se ha hecho en la cinemática directa. En esta ocasión tampoco

se introduce la distancia de la muñeca, esta se tendrá en cuenta siempre en otra función.

También se introducen los límites de giro de cada eje. Esto se hace con el propósito de que se limite el giro de los ejes y no se obtenga una salida que el robot no pueda realizar. Este es un problema que se tenía con las funciones del ToolBox.

```
%% Parámetros del robot
% Longitudes del robot
L1 = 340; %mm
L2 = 400;
L3 = 400;
% Lfin = 126;

% Los ángulos máximos de cada articulación

Q1max = 2.967; % rad = 170º
Q2max = 2.094; % rad = 120º
% Q3max = 2.967;
Q4max = 2.094;
```

Figure 4.11: Inicialización de los parámetros del robot para la parte de posición en la función CinematicaInversa().

A continuación, se procede a calcular los valores de los primeros cuatro ejes. Teniendo el tercer eje bloqueado, el diagrama es el siguiente:

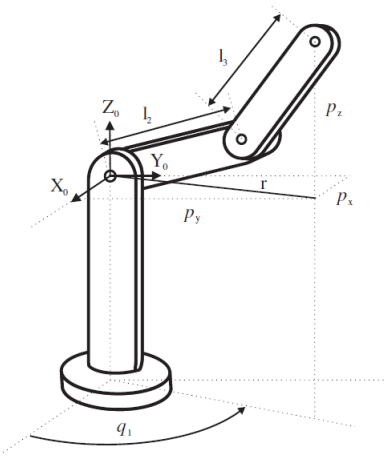


Figure 4.12: Representación de un robot articular de tres ejes [2].

Los datos de partida son p_x, p_y y p_z, ya que se conocen de las matrices homogéneas calculadas previamente.

Se observa que el valor de q_1 se obtiene como:

$$q1 = arctg(\frac{p_y}{p_x}) \tag{4.9}$$

Hay que tener en cuenta que para el cálculo de los próximos ángulos hay que hacer unos cálculos preliminares. Primero, se resta la altura de la base del robot, ya que será constante y no cambiara cuando q_2 y q_4 cambien. Por lo tanto:

$$z_u = z - l_1 \tag{4.10}$$

Esto en el código se refleja como:

```
q3 = 0;

q1 = atan(y/x); % Única solución.

zu = z-L1; % Solo se tiene en cuenta la parte del robot que va a variarse
           % en el eje z.
```

Figure 4.13: Cálculo del ángulo del primer eje y z_u

Además, hay que tener en cuenta la posibilidad de que se pida llegar a un punto fuera del alcance del robot. Por lo tanto se toman dos medidas principales. En el caso de que se quiera llegar a un punto demasiado lejano, se calcula el punto más cercano en la dirección del punto original. Primero se calcula el radio del punto requerido respecto al origen, que en este caso será la segunda articulación.

$$RadioPunto = \sqrt{x^2 + y^2 + z_u^2} \tag{4.11}$$

El radio máximo del robot (R_{max}) es de 800mm. Se hacen los cálculos con 790mm para dejar un margen de error. Si el punto de la ecuación 4.11 es superior, se calcula en que proporción es el radio del punto superior al radio máximo, para así multiplicarlo y calcular el nuevo punto al alcance del robot. El multiplicador queda:

$$Multiplicador = \sqrt{\frac{R_{max}^2}{x^2 + y^2 + z_u^2}} \tag{4.12}$$

Y entonces, el nuevo punto:

$$\begin{aligned} x &= x * Multiplicador \\ y &= y * Multiplicador \\ z_u &= z_u * Multiplicador \end{aligned} \tag{4.13}$$

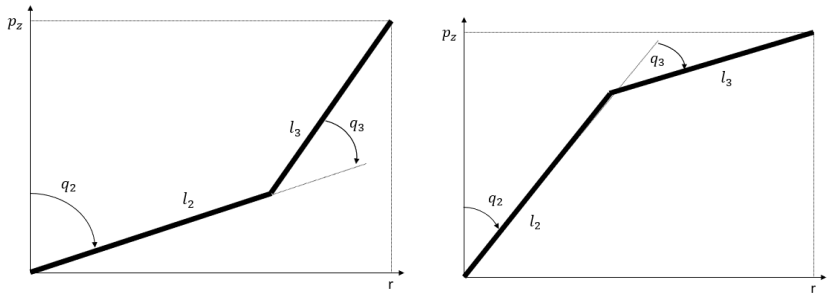
Esto se traduce en el código de la siguiente forma:

```
RadioMax = 790;
RadioPunto = sqrt(x^2 + y^2 +zu^2); % Radio de la esfera.

if RadioPunto > RadioMax
    % multiplicador para obtener el punto máximo que se encuentre en la
    % misma recta que el punto original.
    Multiplicador = sqrt((RadioMax)^2/(x^2+y^2+zu^2));
    x = x*Multiplicador;
    y = y*Multiplicador;
    zu = zu*Multiplicador;
end
```

Figure 4.14: Código del cálculo para que la posición del robot esté dentro del rango.

Una vez habiendo asegurado que el punto a calcular está dentro del rango, se procede a calcular q_2 y q_3. Para obtener q_2 y q_3, se tiene que tener en consideración que existen dos tipos de configuraciones para el codo: codo abajo y codo arriba.



(a) Codo abajo **(b)** Codo arriba

Figure 4.15: Las dos configuraciones que puede tener el codo.

Por lo tanto, para cada configuración de codo habrá una solución de $q2$ y $q4$. Con el teorema del coseno se obtiene que:

$$\begin{aligned} q_4(1) &= -acos(C4) \\ q_4(2) &= acos(C4) \end{aligned} \tag{4.14}$$

Donde:

$$C_4 = \frac{x^2 + y^2 + zu^2 - l_2^2 - l_3^2}{2l_2l_3} \tag{4.15}$$

Y queda:

$$q_2 = atan(\frac{z_u}{\sqrt{x^2 + y^2}}) + atan(\frac{l_3 sin(q_4)}{l_2 l_3 cos(q_4)}) - \frac{\pi}{2} \tag{4.16}$$

Trasladado al código:

```
C4 = ((x^2)+(y^2)+(zu^2)-(L2^2)-(L3^2))/(2*L2*L3);

q4aux(1) = -acos(C4);
q4aux(2) = acos(C4);

% A diferencia de las fórmulas habituales se ponen los dos signos igual
% ya que en el sin(q4) ya esta metído el signo.

q2aux(1) = atan(zu/sqrt(x^2+y^2))+atan((L3*sin(q4aux(1)))/ ...
    (L2+L3*cos(q4aux(1))))-pi/2;
q2aux(2) = atan(zu/sqrt(x^2+y^2))+atan((L3*sin(q4aux(2)))/ ...
    (L2+L3*cos(q4aux(2))))-pi/2;
```

Figure 4.16: Código del cálculo de la segunda y cuarta articulación.

Por último, se tiene que asegurar que ninguna articulación excede su rango de movimiento. Previamente se ha asegurado que el punto no esté demasiado lejos, pero puede suceder que por límites propios de las articulaciones el robot no sea capaz de llegar a ciertos puntos. En este caso, si el ángulo de alguna articulación supera sus posibilidades, este parámetro coge el valor del ángulo más cercano al que pueda llegar. Esta solución pretende única y exclusivamente que el controlador del robot no mande una señal de error, ya que en estos casos el robot deja de seguir el código y se queda congelado, teniendo que reiniciarlo. En el código:

```
if q1 < -Q1max
    q1 = -Q1max;
elseif q1 > Q1max
    q1 = Q1max;
end

% Selección del camino a recorrer con q2 y q4.

if q2aux(2) < -Q2max || q4aux(2) < -Q4max ...
    || q4aux(2) > Q4max || q2aux(2) > Q2max
    q2 = q2aux(1);
    q4 = q4aux(1);
else
    q2 = q2aux(2);
    q4 = q4aux(2);
end

if q2 < -Q2max
    q2 = -Q2max;
elseif q2 > Q2max
    q2 = Q2max;
elseif q4 < -Q4max
    q4 = -Q4max;
elseif q4 > Q4max
    q4 = Q4max;
end
```

Figure 4.17: Código para asegurar que los valores de las articulaciones se mantienen entre los límites.

Una vez calculados los ángulos que definirán la posición del extremo del robot, hay que definir los ángulos de las articulaciones que orientarán el extremo del robot, es decir, $q5$, $q6$ y $q7$. Para ello, denominando R_7^0 a la sub-matriz de rotación de T_7^0, se tiene que:

$$R_7^0 = R_4^0 R_7^4 \tag{4.17}$$

Donde:

$$R_4^0 = R_1^0 R_2^1 R_3^2 R_4^3 \tag{4.18}$$

Las matrices de la ecuación 4.18 son conocidas, ya que previamente se han calculado los ángulos de dichas articulaciones para obtener la traslación del extremo del robot.

Esto en el código se resuelve de la siguiente forma:

```
% Matrices de rotacion calculados con DH.

R01 = [cos(q1) 0 -sin(q1) ; sin(q1) 0 cos(q1) ; 0 -1 0];
R12 = [cos(q2) 0 sin(q2) ; sin(q2) 0 -cos(q2) ; 0 1 0];
R23 = [cos(q3) 0 sin(q3) ; sin(q3) 0 -cos(q3) ; 0 1 0];
R34 = [cos(q4) 0 -sin(q4) ; sin(q4) 0 cos(q4) ; 0 -1 0];

R04 = R01*R12*R23*R34;
R40 = inv(R04);
```

Figure 4.18: Código para el cálculo de la matriz R_4^0.

La matriz R_7^0 también es conocida. Se sabe la orientación que debe de tener el extremo del robot. Esto viene dado en ángulos de Euler a, b y c, donde c es el giro del vector x de la coordenadas base, b del vector y y a del vector z.

Se obtiene R_7^0 con:

$$R_7^0 = Rotz(a)Roty(b)Rotx(c) \tag{4.19}$$

Donde:

$$Rotx(c) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(c) & -sen(c) \\ 0 & sen(c) & cos(c) \end{bmatrix} \tag{4.20}$$

$$Roty(b) = \begin{bmatrix} cos(b) & 0 & sen(b) \\ 0 & 1 & 0 \\ -sen(b) & 0 & cos(b) \end{bmatrix} \tag{4.21}$$

$$Rotz(a) = \begin{bmatrix} cos(a) & -sen(a) & 0 \\ sen(a) & cos(a) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.22}$$

Por lo tanto, la matriz que queda por saber es la matriz R_7^4. Se puede obtener haciendo:

$$R_7^4 = R_0^4 R_7^0 = (R_4^0)^{-1} R_7^0 \tag{4.23}$$

En el código:

```
% Las matrices de rotación, para calcular la matriz de la posición final.

Rx = [1 0 0 ; 0 cos(a) -sin(a) ; 0 sin(a) cos(a)];
Ry = [cos(b) 0 sin(b) ; 0 1 0 ; -sin(b) 0 cos(b)];
Rz = [cos(c) -sin(c) 0 ; sin(c) cos(c) 0 ; 0 0 1];

R = Rz*Ry*Rx;
```

Figure 4.19: Código para el cálculo de la matriz de rotación que se desea obtener.

Por otra parte, la matriz R_7^4 es una matriz de rotación correspondiente a la matriz de transformación homogénea R_7^4, donde ya se conocen las matrices homogéneas con las que se obtiene gracias a Denavit-Hatenberg. Las matrices son:

$$\begin{aligned} R_5^4 &= \begin{bmatrix} cos(\theta_5) & 0 & -sen(\theta_5) \\ sen(\theta_5) & 0 & cos(\theta_5) \\ 0 & -1 & 0 \end{bmatrix} \\ R_6^5 &= \begin{bmatrix} cos(\theta_6) & 0 & sen(\theta_6) \\ sen(\theta_6) & 0 & -cos(\theta_6) \\ 0 & 1 & 0 \end{bmatrix} \\ R_7^6 &= \begin{bmatrix} cos(\theta_7) & -sen(\theta_7) & 0 \\ sen(\theta_7) & cos(\theta_7) & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{4.24}$$

Entonces, se tiene que:

$$R_7^4 = \begin{bmatrix} C_5C_6C_7 - S_5S_7 & -C_5C_6S_7 - S_5C_7 & C_5S_6 \\ S_5C_6C_7 + C_5S_7 & -S_5C_6S_7 + C_5C_7 & -S_5C_6 \\ -S_6C_7 & S_6S_7 & C_6 \end{bmatrix} \tag{4.25}$$

La ecuación 4.26 se puede reescribir como:

$$r_{ij} = \begin{bmatrix} C_5C_6C_7 - S_5S_7 & -C_5C_6S_7 - S_5C_7 & C_5S_6 \\ S_5C_6C_7 + C_5S_7 & -S_5C_6S_7 + C_5C_7 & -S_5C_6 \\ -S_6C_7 & S_6S_7 & C_6 \end{bmatrix} \tag{4.26}$$

Teniendo esto en cuenta, se obtiene que:

$$\begin{aligned} q_5 &= arctan(\frac{r_{23}}{r_{13}}) \\ q_6 &= arcos(r_{33}) \\ q_7 &= arctan(\frac{r_{32}}{-r_{31}}) \end{aligned} \tag{4.27}$$

Estas ecuaciones se traducen en el código por:

```
q6 = acos(R47(3,3));
q5 = atan2(R47(2,3),R47(1,3));
q7 = atan2(R47(3,2),-R47(3,1));
```

Figure 4.20: Código para el cálculo de las últimas articulaciones del robot.

Por último, se asegura mediante código que ninguno de los ángulos calculados excede las limitaciones del robot, asegurando que en el peor de los casos se quede en el límite.

Con esto, el problema cinemático inverso del robot está solucionado, por lo tanto solo queda girar los ejes a los ángulos calculados y se podría volver a empezar el ciclo.

4.2 Entrenamiento de la Red

En esta sección, primero se necesita obtener el conjuntos de datos de entrenamiento, es decir, las fotos que se utilizarán para el entrenamiento de la red. Después, se procederá a la configuración y posteriormente al entrenamiento de dicha red neuronal. Por último, se pondrá a prueba a la red neuronal para así poder analizar los resultados obtenidos.

4.2.1 Obtener las Fotos Originales de la Pared

Se ha explicado previamente que el conjunto de datos de entrenamiento de la red neuronal serán fotos que se obtendrán de una pared. Sin embargo, estas fotos tampoco pueden ser utilizadas automáticamente para el entrenamiento de la red neuronal. El tamaño original de las fotos es demasiado grande como para que la foto pueda ser utilizada en la red neuronal que se utilizará, por lo tanto se dividirán de tal forma que se crearán nuevas fotos partiendo de las fotos originales.

En este apartado se explicará el código y el procedimiento con el que se han obtenido las fotos originales, y después se explicará el código con el que se han partido dichas fotos para crear las nuevas fotos que servirán para entrenar a la red neuronal.

Pero primero, se ha de escoger la pared que se quiere analizar. En este proyecto, se ha escogido la siguiente:

Figure 4.21: Pared de estudio.

Es una pared sencilla gris con pocos elementos. El fondo es liso para que la red neuronal solo se fije en los tres AprilTag y la canaleta que se sitúa en la parte derecha de la imagen. Los AprilTag escogidos son de la familia más simple que hay, las "tag36h11".

La poca cantidad de elementos en la imagen pretende que la red neuronal tenga que aprender menos elementos, si se ponen demasiados AprilTag podría ser contraproducente y no es el objetivo que buscamos. Se espera que cuando intente predecir en zonas donde no haya nada se confunda, ya que no debería de haber ningún elemento que pudiese dar pistas a la red de su localización.

Una vez escogida la zona que se trabajará, se tienen que obtener las fotos. Para ello, se hace uso de los programas creados previamente para el robot. Sin embargo, el programa principal se cambia sustancialmente. A continuación se explicará cada parte del mismo.

Primero, como con el programa del robot, se borran las variables anteriores y se establece conexión con el robot. Después, se inicializa la variable de la cámara y se posiciona el robot en posición de reposo.

Acabado esto, se inicializan las variables que servirán para posicionar y orientar el robot como se desea. Para ello, primero se inicializa la posición inicial del extremo del robot en coordenadas cartesianas y en milímetros. Los valores se han obtenido mediante prueba y error para que apunte a la zona deseada. Igualmente, se inicializan los ángulos de Euler que se desean, obtenidos también mediante prueba y error.

```
% Posicion de inicio
x = -141;
y = -565;
z = 650;
% Los angulos por experimentacion
a = 45*pi/180;
b = -90*pi/180;
c = 0*pi/180;
```

Figure 4.22: Código para el cálculo de la matriz R_4^0.

A continuación, se inicializa el número de iteraciones que realizará tanto en X (Nx) como en Y (Ny) el bucle que moverá poco a poco al robot, es decir, el espacio en el que se moverá la cámara. Pero primero, hay que entender en que posición se encuentran las coordenadas base del robot respecto a las que tendrá la pared.

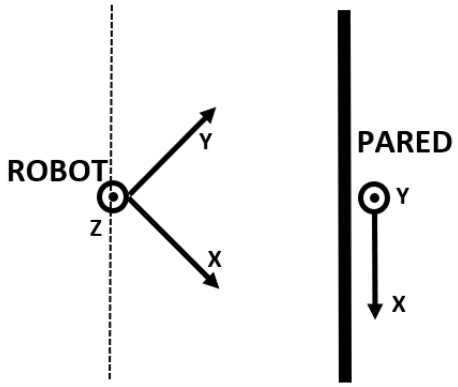


Figure 4.23: Croquis de como se ven las coordenadas del robot y las de la pared visto desde arriba.

Como se puede observar en la imagen, tal y como se encuentra el robot, como se quiere que las fotos siempre se saquen a la misma distancia respecto a la pared, se tendrán que hacer los cálculos teniendo en cuenta que no se utilizan los mismos sistemas de coordenadas.

Se decide que el robot se mueva 7mm en X y en Y y 10mm en Z cada vez que se mueva horizontalmente o verticalmente respecto a la pared, de tal forma que el recorrido total del robot será:

$$\begin{aligned} HorizontalTotal &= 7(Nx-1)\sqrt{2}(mm) \\ VerticalTotal &= 10(Ny-1)(mm) \end{aligned} \tag{4.28}$$

Esto en el código:

```
Nx=50;
Ny = 60;

HorizontalTotal = sqrt(2)*7*(Nx-1);
VerticalTotal = 10*(Ny-1);
```

Figure 4.24: Código para el cálculo del recorrido total del robot en las coordenadas de la pared.

A continuación, se coloca el extremo del robot en el punto deseado. Para ello, se utiliza la función de cinemática inversa que se ha creado previamente. Una vez calculados los ángulos de cada eje, se coloca el robot.

```
[q1, q2, q3, q4, q5, q6, q7]= CinematicaInversa (a, b, c, x, y, z);

pos={q1, q2, q3, q4, q5, q6, q7};
movePTPJointSpace( t_Kuka , pos, relVel);
```

Figure 4.25: Código para la colocación del robot en las coordenadas iniciales deseadas.

Una vez colocado el robot, se procede a entrar en los bucles que irán moviendo el robot por toda la pared. Para ello, se recalcula la posición que debe de adoptar el extremo del robot en cada iteración. Las formulas son:

$$\begin{aligned} x &= x - 7(i-1)(mm) \\ y &= y - 7(i-1)(mm) \\ z &= z - 10(j-1)(mm) \end{aligned} \tag{4.29}$$

La posición que se usa para calcular las nuevas posiciones siempre es la inicial. Acto seguido, se calcula cual es esa posición para la pared, ya que el origen de las coordenadas de la pared están en medio.

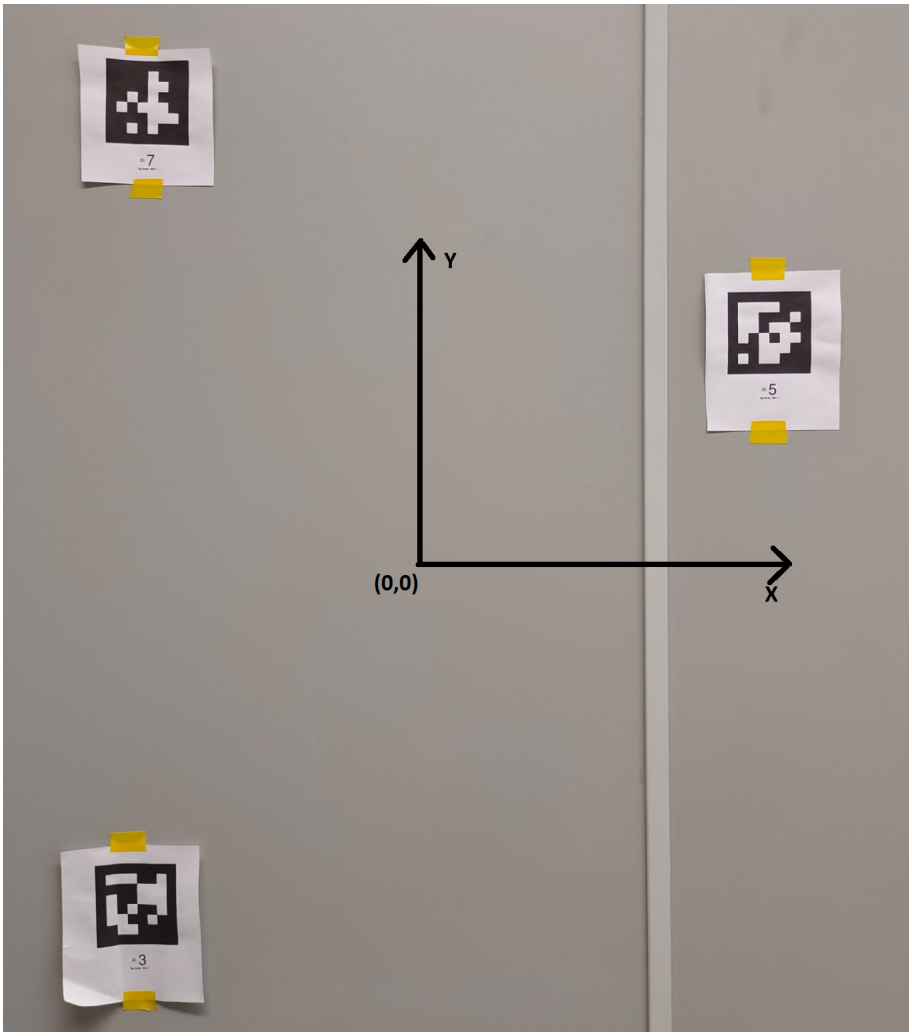


Figure 4.26: Coordenadas de la pared para el nombramiento de las fotos según la posición.

Por lo tanto, el nombre de cada foto que se vaya a sacar será dependiendo de las coordenadas de la pared en las que se haya sacado. Las ecuaciones para obtener dichas coordenadas son:

$$\begin{aligned} NombreHorizontal &= -\frac{HorizontalTotal}{2} + 7(i-1)\sqrt{2}(mm) \\ NombreVertical &= \frac{VerticalTotal}{2} - 10(j-1)(mm) \end{aligned} \tag{4.30}$$

Después, con la posición del robot calculada, se utiliza la función de cinemática inversa para obtener los ángulos de cada eje del robot. Acto seguido se mueve el robot al punto deseado.

El cálculo de las posiciones y los nombres de las fotos, en el código:

```
for i=1:1:Nx %50
    for j=1:1:Ny % 60
        % Calculo x, y, z
        x = x-(i-1)*7;
        y = y+(i-1)*7;
        z = z-(j-1)*10;
        %Nombres Fotos

        NombreHorizontal = -HorizontalTotal/2 + sqrt(2)*7*(i-1);
        NombreVertical = VerticalTotal/2 - 10*(j-1);

        [q1, q2, q3, q4, q5, q6, q7] = ...
            CinematicaInversa (a, b, c, x, y, z);

        pos={q1, q2, q3, q4, q5, q6, q7};
        movePTPJointSpace( t_Kuka , pos, relVel);
```

Figure 4.27: Código para el cálculo de las posiciones que adoptará el robot para sacar las fotos y el cálculo de los nombres de las fotos que se sacaran respecto a las coordenadas de la pared.

Cada vez que se mueva el robot, hay que sacar una foto y guardarla en una carpeta para luego poder utilizarla. Teniendo en cuenta que la línea anterior a esta es la que mueve el robot de posición, se introduce un tiempo de pausa para que al robot le de tiempo a colocarse en la nueva posición. Después se saca la foto, se crea el nombre de la foto con las variables de la posición previamente comentadas y se guarda la foto en la carpeta deseada. Por último, se vuelven a inicializar los valores de las coordenadas del robot a las iniciales. Con esto, el bucle quedaría definido.

```
        % Fotos
        pause(0.5)
        imagen = snapshot(cam);
        NombreFichero=strcat('ImagenesMas\F',sprintf('%.1f,%.1f', ...
            NombreHorizontal,NombreVertical),'.jpg');
        imwrite(imagen,NombreFichero);
        x = -141;
        y = -565;
        z = 650;
    end
end
```

Figure 4.28: Código para obtener las fotos y guardarlas en la carpeta deseada.

Una vez terminados los bucles, se apaga el servidor y se termina la conexión con el robot. Con esto, se termina el código que permite sacar las fotos necesarias de la pared mediante el robot.

4.2.2 División de las fotos obtenidas para poder entrenar la red neuronal

Previamente se ha comentado como las fotos deben de ser divididas en un tamaño que la red neuronal pueda tratar. La red neuronal que se va a utilizar como base para el entrenamiento es la VGG16. Esta está diseñada para que las fotos tengan una resolución de 128*128, pero se puede cambiar para que pueda entrenarse con fotos de 64*64 o 32*32, por ejemplo. Las fotos originales tienen una resolución de 640*480, resolución no válida para la red.

Figure 4.29: Fotos obtenidas a través de la cámara pegada a al final del Robot. Resolución 640*480.

Segmentar las fotos servirá en primer lugar para poder utilizarlas para el entrenamiento de la red, pero también servirá para que tenga mayor cantidad de datos de entrada para ser entrenada. Para realizar esto, se ha diseñado dos programas que dividen las fotos originales. Son dos porque se van a hacer dos entrenamientos con dos tamaños de imagen de entrenamiento, de 64*64 y 128*128. Los dos programas son idénticos, por lo tanto se explicará un único programa, el de 64*64.

El programa es un script de Matlab que se llama "DividirFotos.m". Lo primero que se hace en el script es limpiar la memoria y borrar todo lo que haya en unas carpetas llamadas "ImagenesTrain", "ImagenesTest" y "ImagenesVal". A continuación, se crean tres arrays con los mismos nombres.

```
clc
close all
clear all

delete(['C:\Users\Mikel\Desktop\TFM-Publicaciones' ...
    '\RED_NEURONAL\Imagenes\ImagenesTrain\*'])
delete(['C:\Users\Mikel\Desktop\TFM-Publicaciones' ...
    '\RED_NEURONAL\Imagenes\ImagenesTest\*']);
delete(['C:\Users\Mikel\Desktop\TFM-Publicaciones' ...
    '\RED_NEURONAL\Imagenes\ImagenesVal\*']);

SalidasTrain=[];
SalidasVal=[];
SalidasTest=[];
```

Figure 4.30: Primeras líneas del script "DividirFotos.m".

Las fotos que se vayan a dividir se separaran en tres carpetas diferentes, que son las que se han limpiado inicialmente. Esto es porque a la hora de entrenar una red neuronal, una cantidad de datos sirve para entrenar (Train), otra para asegurarse durante el entrenamiento que no se está produciendo sobre-ajuste (Val) y por último las fotos que permitirán al final poner a prueba la red neuronal entrenada (Test). Las carpetas tendrán un porcentaje distinto de fotos. En este caso, La carpeta del entrenamiento tendrá un 70% de las fotos, la carpeta de validación un 15% y la carpeta de testeo el último 15%. Más adelante en el programa se realizará la partición.

```
Ptrain=0.7;
Pval=0.15;
Ptest=0.15;
```

Figure 4.31: Variables que permitirán situar las fotos en las carpetas adecuadas.

La siguiente parte del código está dedicada a dividir las fotos y a darles a cada división de las fotos el nombre con las coordenadas apropiadas. Para ello, primero se realizan las mismas cuentas que en el script que obtiene las fotos para poder llamar a todas las fotos desde este script. Después, se hacen los cálculos de cuál será la posición de las nuevas fotos partiendo de la posición de la foto original que se ha partido.

Para ello, primero se hace el cálculo de cuanta distancia abarca cada píxel de una foto en la pared. La distancia se mide de forma experimental, y en el código se tiene que:

```
DistImagen= 492; %mm
DistPixel = DistImagen/NxmaxSub;
```

Figure 4.32: Cálculo de la distancia de pared por píxel.

Donde 492 son los milímetros de pared que abarca horizontalmente una foto original y NxmaxSub son los 640 píxeles que tiene horizontalmente cada imagen.

Por lo tanto, si se tienen las coordenadas del centro de cada foto original y la distancia por píxel, se pueden obtener las coordenadas de las nuevas fotos que se vayan a obtener.

Primero, se leen las fotos originales en el siguiente fragmento de código:

```
for i=1:1:Nxmax %50
    for j=1:1:Nymax % 60
        %Nombres Fotos

        NombreHorizontal = -HorizontalTotal/2+sqrt(2)*7*(i-1);
        NombreVertical = VerticalTotal/2-10*(j-1);

        NombreFichero=sprintf('F%.1f,%.1f.jpg' ...
            ,NombreHorizontal,NombreVertical);
        I=imread(NombreFichero);
```

Figure 4.33: Leer las fotos originales para luego poder trocearlas.

Una vez obtenida la foto que se troceará, se hace un nuevo bucle que tomará una sección de la foto con el tamaño deseado y lo guardará. En este caso, por cada 100 píxeles de la foto original, ser guardará una foto con el nuevo tamaño, como muestra la siguiente imagen.

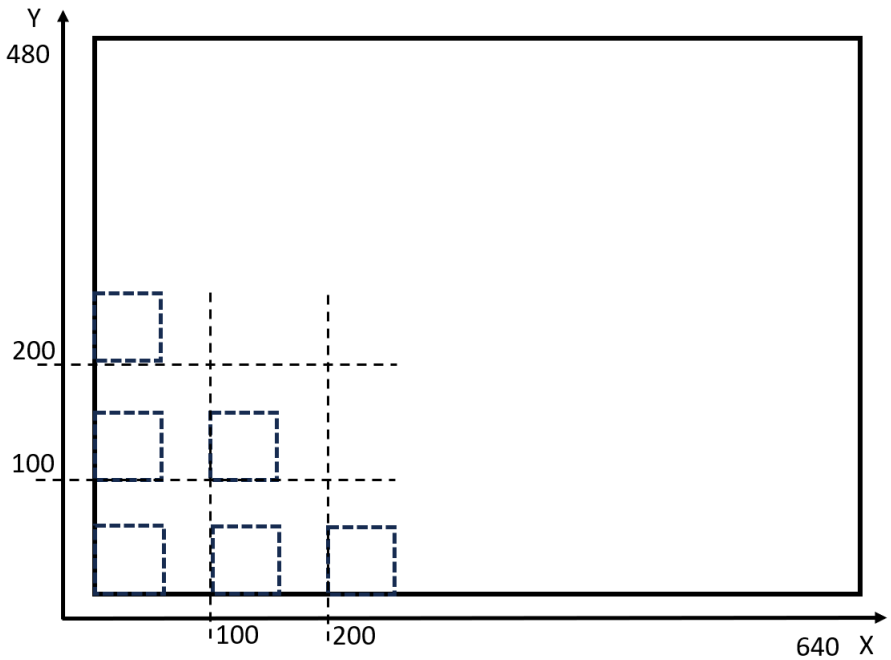


Figure 4.34: División de las fotos originales en fotos más pequeñas. Cada 100 píxeles en la foto original, una nueva foto de tamaño 64*64.

Después, se crea un número aleatorio entre 0 y 1 que determinará en cual de las tres carpetas se guardará la nueva imagen recortada.

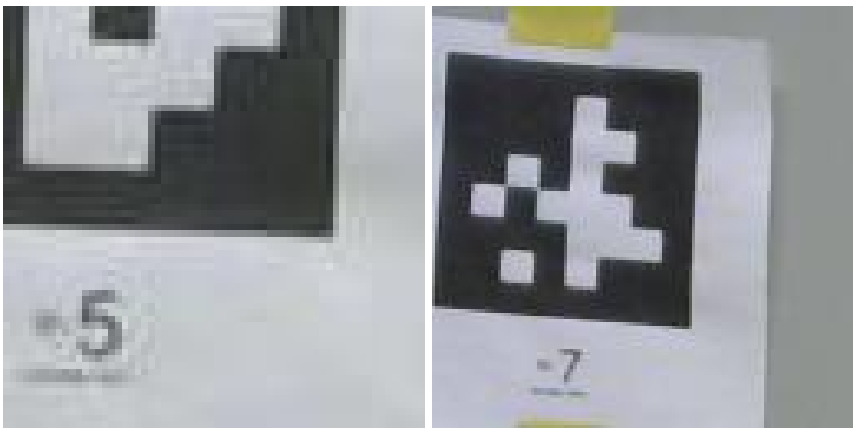
El código de esta parte es:

```
for i1=1:100:NxmaxSub-Nx
    for i2=1:100:NymaxSub-Ny

        NombreHorizontalSub = NombreHorizontal-DistPixel*...
            (NxmaxSub-2*Nx)/2+i1*DistPixel;
        NombreVerticalSub = NombreVertical+DistPixel*...
            (NymaxSub-2*Ny)/2-i2*DistPixel;
        Isub=I(i2:i2+Nx-1,i1:i1+Ny-1,:);
        Prob=random('Uniform',0,1,1,1);
        if Prob<Ptrain
            NombreFichero=strcat('ImagenesTrain\F',sprintf ...
                ('%.1f,%.1f.jpg',NombreHorizontalSub,NombreVerticalSub));
            imwrite(Isub,NombreFichero);
        else
            if Prob<Ptrain+Pval
                NombreFichero=strcat('ImagenesVal\F',sprintf ...
                    ('%.1f,%.1f.jpg',NombreHorizontalSub, ...
                    NombreVerticalSub));
                imwrite(Isub,NombreFichero);
            else
                NombreFichero=strcat('ImagenesTest\F',sprintf ...
                    ('%.1f,%.1f.jpg',NombreHorizontalSub, ...
                    NombreVerticalSub));
                imwrite(Isub,NombreFichero);
            end
        end
    end
end
```

Figure 4.35: Código para dividir las fotos originales, ponerles nombre y clasificarlos en la carpeta correspondiente.

Como se muestra en las siguientes imágenes, el resultado de dividir la imagen original es la siguiente.



(a) Resolución 64*64. **(b)** Resolución 128*128.

Figure 4.36: Las dos resoluciones que se van a utilizar las redes neuronales.

Una vez divididas las fotos y tras haberlas guardado en las carpetas correspondientes, queda organizar los datos de entrada de tal forma que la red neuronal las acepte. Para ello, hay que crear un "imageDatastore" de todas las imágenes de cada carpeta y atribuir a cada imagen unas coordenadas. Aunque se le hayan puesto los nombres con las coordenadas, no son datos que la red neuronal entienda. Hace falta juntar cada imagen con su celda correspondiente de coordenadas. El "imageDatastore" tiene que ir unido a un "arrayDatastore" que guarde las coordenadas de cada imagen en el mismo orden que se han guardado las imágenes en el "imageDatastore".

Para ello, se ha creado una nueva función llamada "Irakurri.m". Esta función lee todos los títulos de las imágenes del "imageDatastore" y guarda las coordenadas en un array (data) de tal forma que se irán acumulando todas las coordenadas de las imágenes en el mismo orden en los que se han guardado las imágenes.

Por último, se normalizan las coordenadas de tal forma que el tiempo de computación de la red neuronal se reduzca cuando calcule la media y la varianza del error. En el código:

```
function [data, xmax, ymax]=Irakurri(Filename,L)
data1 = [];
data = [];
for i=1:L
    Refkoma=strfind(Filename(i,1),',');
    RefF=strfind(Filename(i,1),'F');
    x=str2double(Filename{i,1}(RefF{1,1}(2)+1:Refkoma{1,1}-1));
    y=str2double(Filename{i,1}(Refkoma{1,1}+1:end-4));
    data1=[data1;[x,y]];
end
xmax = max(data1(:,1));
ymax = max(data1(:,2));
data(:,1) = data1(:,1)/xmax;
data(:,2) = data1(:,2)/ymax;
end
```

Figure 4.37: Código que lee los nombres de cada foto y guarda las coordenadas en un array.

Una vez obtenido el array, se puede crear un conjunto de datos que la red neuronal puede leer. La última parte del código del programa de la división de fotos:

```
ImagenesInputTrain=imageDatastore("ImagenesTrain\", ...
    "FileExtensions",".jpg");
LTrain=length(ImagenesInputTrain.Files);
[SalidasTrain, xmaxTrain, ymaxTrain]=Irakurri(ImagenesInputTrain.Files, ...
    LTrain);
SalidasDSTrain=arrayDatastore(SalidasTrain);
ImagenesConSalidasDSTrain=combine(ImagenesInputTrain,SalidasDSTrain);

ImagenesInputTest=imageDatastore("ImagenesTest\","FileExtensions",".jpg");
LTest=length(ImagenesInputTest.Files);
[SalidasTest, xmaxTest, ymaxTest]=Irakurri(ImagenesInputTest.Files, LTest);
SalidasDSTest=arrayDatastore(SalidasTest);
ImagenesConSalidasDSTest=combine(ImagenesInputTest,SalidasDSTest);

ImagenesInputVal=imageDatastore("ImagenesVal\","FileExtensions",".jpg");
LVal=length(ImagenesInputVal.Files);
[SalidasVal, xmaxVal, ymaxVal]=Irakurri(ImagenesInputVal.Files, LVal);
SalidasDSVal=arrayDatastore(SalidasVal);
ImagenesConSalidasDSVal=combine(ImagenesInputVal,SalidasDSVal);

save all
```

Figure 4.38: Parte del código que guarda los datos de entrada de la red neuronal de tal forma que las pueda leer.

Una vez acabado con esto, se puede proceder a entrenar la red neuronal. Todos los datos de este programa quedan guardados y no se deben de perder por si se desea cambiar de configuración de la red neuronal y entrenarla otra vez.

4.2.3 Entrenamiento de la Red Neuronal

Para crear el código de entrenamiento que se ha utilizado, se ha utilizado la App "DeepNetworkDesigner" [26]. Esta App permite entrenar redes neuronales viejas o crear nuevas redes neuronales de forma sencilla.

En la sección "ImportData", se cargan los datos de entrenamiento y validación creados previamente.

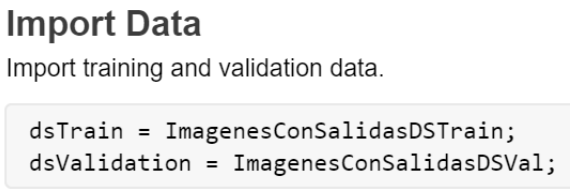


Figure 4.39: Código que carga los datos de entrenamiento y validación necesarios para el entrenamiento.

A continuación, Matlab brinda la opción de cambiar varios parámetros del entrenamiento de la red neuronal.

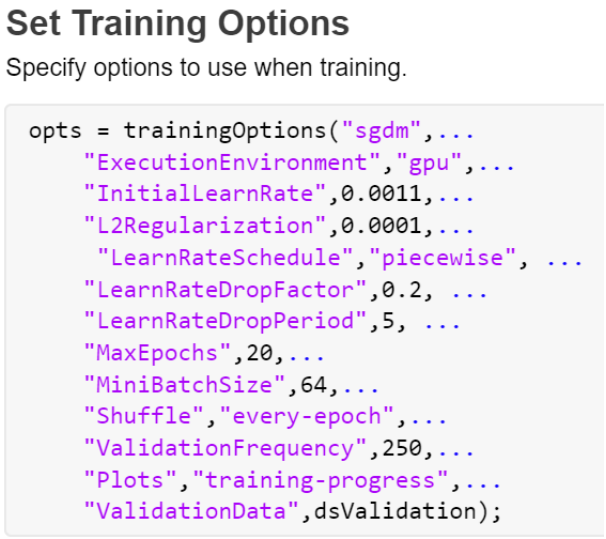


Figure 4.40: Algunos de los parámetros que se pueden cambiar para entrenar la red neuronal.

Se han hecho varios intentos de entrenamiento, y algunos de estos parámetros se han ido cambiando hasta dejarlos como en la imagen, como la frecuencia de validación. Se ha aumentado la frecuencia de validación ya que se ha visto que no hace falta validar tanto si el entrenamiento va correctamente y ahorra muchísimo tiempo.

La siguiente parte del código corresponde a las diferentes capas de la red neuronal. Se han hecho tres cambios. En primer lugar, en la capa de entrada, se ha ajustado el tamaño de la imagen para cada caso. Esto se hace en la primera línea.

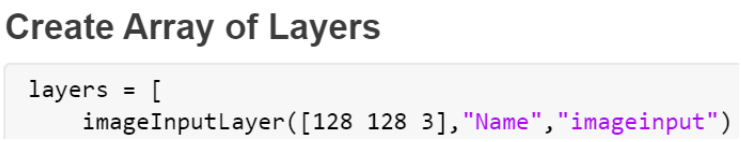


Figure 4.41: Zona del código donde se escoge el tamaño de las imágenes de entrada.

En el caso de la imagen anterior, las imágenes de entrada serán del tamaño 128*128, pero esto se puede cambiar a 64*64, por ejemplo.

El segundo cambio, es que para utilizar la técnica de aprendizaje transferido, se han bloqueado todas las capas excepto las últimas tres. Para bloquear el aprendizaje, se utiliza el comando "WeightL2Factor".

```
convolution2dLayer([3 3],64,"Name","conv1_1","Padding", ...
[1 1 1 1],"WeightL2Factor",0)
```

Figure 4.42: Zona del código donde se bloquean los pesos sinápticos de las capas.

Por último, se han cambiado los nombres de las tres últimas capas para dejar claro cuales son las que cambian. Estas no tienen la restricción para los pesos que tienen las anteriores.

```
fullyConnectedLayer(1000,"Name","Benito")
reluLayer("Name","relu6")
dropoutLayer(0.5,"Name","drop6")
fullyConnectedLayer(100,"Name","Patxi")
reluLayer("Name","relu7")
dropoutLayer(0.5,"Name","drop7")
fullyConnectedLayer(2,"Name","fc")
regressionLayer("Name","regressionoutput")];
```

Figure 4.43: Las tres últimas capas y la capa de salida.

Una vez acabado el entrenamiento, se guardan los resultados obtenidos.

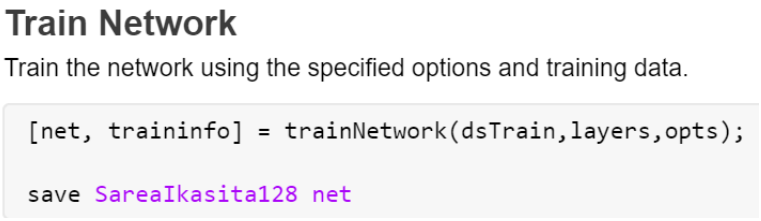


Figure 4.44: Zona del código que guarda la red neuronal entrenada.

4.2.4 Poner a Prueba la Red Neuronal

Para poner a prueba la red neuronal entrenada se han realizado dos programas.

El primero se llama "EjecutarRedEntrenaDatosTest.m". En este programa, con las fotos guardadas en la carpeta de test, se calculan los errores de posicionamiento que hace la red neuronal respecto a la realidad tanto en el eje X como en el eje Y.

Para ello, primero se limpia la memoria, se inicializa la red neuronal que se quiere poner a prueba y se inicializan las variables que más tarde se van a usar.

```
clear all
clc

load all
load('SareaIkasita.mat')

errorvecx=[];
errorvecy=[];
Posmat=[];
```

Figure 4.45: Inicialización de las variables necesarias para el testeo de la red neuronal.

A continuación, se crea un bucle while. Este bucle no terminará hasta que ya no queden más imágenes por analizar. Una vez en el bucle, se leen los datos de entrada que se van a analizar, es decir, la imagen y la posición real de la imagen. Después, la red neuronal predice la posición en la que cree que está esa imagen, dando unas coordenadas X e Y. Acto seguido se hace la resta entre la posición real y la predicha y se van guardando los resultados en un array, tanto para el eje X como para el eje Y.

Los errores se añaden a un array con las posiciones originales para saber que error se ha cometido en cada zona de la pared.

```
while hasdata(ImagenesConSalidasDSTest)
    data=read(ImagenesConSalidasDSTest);

    PosTeorica=data{1,2};

    PosPredicha=predict(net,data{1,1});

    errorx = abs(xmaxTest*PosTeorica(1)-xmaxTest*PosPredicha(1));
    errorvecx=[errorvecx,errorx];
    errory = abs(ymaxTest*PosTeorica(2)-ymaxTest*PosPredicha(2));
    errorvecy=[errorvecy,errory];
    Posmat=[Posmat,[xmaxTest*PosTeorica(1),ymaxTest*PosTeorica(2)]'];
end
```

Figure 4.46: Bucle para obtener el error tanto en el eje X como en el eje Y.

Estos valores se ordenan de menor a mayor y se guardan para utilizarlos en el siguiente programa.

```
MatrizTotal=[errorvecx;errorvecy;Posmat]';
MatrizTotal=sortrows(MatrizTotal,1);

save redEntrenada
```

Figure 4.47: Código para guardar la matriz obtenida y utilizarla más tarde.

Análisis de resultados

5

A continuación, se comentarán los resultados obtenidos con el entrenamiento de la red neuronal.

5.1 Entrenamiento de la Red

En este proyecto, se ha entrenado una red neuronal de diferentes formas para entender mejor cuales son las regiones más significativas de una red neuronal para su aprendizaje. Para ello, se han obtenido fotos a través de una cámara situada en el extremo de un robot de 6 grados de libertad y se ha fotografiado una pared con diferentes elementos. Los elementos principales de la pared son: tres AprilTag y una canaleta. La pared no se ha llenado de objetos para que la red neuronal se pudiese centrar en esos cuatro elementos.

Cada foto se ha etiquetado con una ubicación en coordenadas cartesianas respecto al centro de la pared que se ha estudiado. Estas coordenadas se han usado como otro elemento de entrada. Si la red neuronal fuese capaz de identificar la posición de una fotografía no utilizada en el entrenamiento significaría que ha aprendido a distinguir las características que hacen a esa fotografía única y la diferencian respecto al resto de la pared.

Por lo tanto, una región con menor error en la predicción de la ubicación implica que tiene características más sencillas de aprender para una red neuronal, mientras que una región con mayor error indicaría que o las características son demasiado complejas para poder entenderlas o que no ha encontrado suficientes características que permitan diferenciar esa región de las demás.

Se han hecho diferentes pruebas, y a continuación se mostrarán las redes neuronales que mejores resultados han obtenido. Estas redes neuronales tienen la misma configuración de entrenamiento que se ha mostrado en la imagen 4.40. La única diferencia entre las dos redes neuronales es el tamaño de las imágenes de entrada. En una, la resolución de las imágenes ha sido de 64*64, mientras que en la otra de 128*128.

En la siguiente imagen se muestra el entrenamiento de la red neuronal de 128*128.

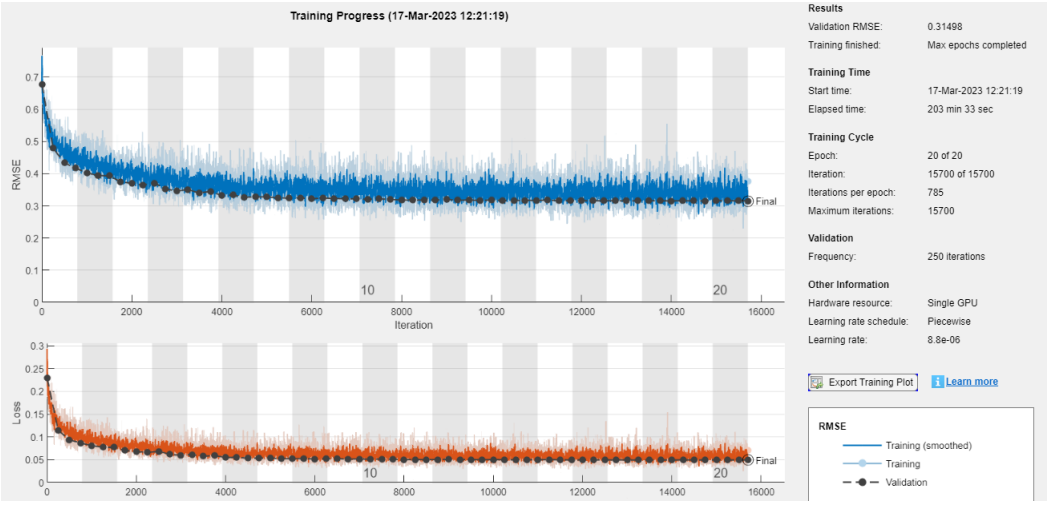


Figure 5.1: Entrenamiento de la red neuronal con imágenes de resolución 128*128.

La media de error al final del entrenamiento es de 0,31498. Es una cifra bastante alta, pero a la vez lógica, Gran parte de la pared estudiada no tiene ningún elemento aparente que la red neuronal pudiese distinguir, por lo tanto es normal que en todas estas zonas falle y que esto aumente el error global de las estimaciones. Lo importante es que las zonas con elementos distinguibles tengan menor error.

Para estudiar esto, se ha utilizado el programa "Graficos.m" explicado previamente. Se analizarán uno por uno los resultados obtenidos.

Primero, se va a analizar los resultados del error en el eje X con el entrenamiento de imágenes de entrada de resolución 64*64. Se puede observar una franja de azul oscuro muy marcado en la zona derecha de la pared. Los colores azules oscuros indican que hay muy poco error en esa zona. Esa zona es sin duda la zona de la canaleta, y esto demuestra que la red es capaz de identificar perfectamente donde está la canaleta situada.

Con un azul menos marcado se pueden ver tres pequeñas zonas con menos error que el resto de la pared. Son las zonas con el AprilTag. Sin embargo, las zonas están bastante difuminadas, lo que indica que no es capaz de aprender adecuadamente cada AprilTag.

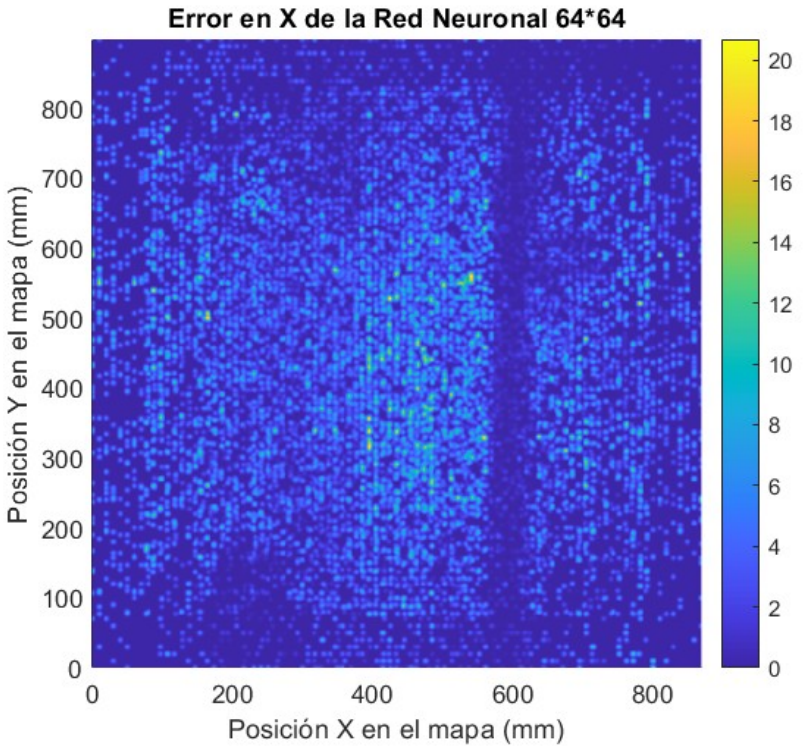


Figure 5.2: Error en X en entrenamiento con imágenes de resolución 64*64.

En cuanto al eje Y, los resultados son más difusos. Se puede distinguir ligeramente el AprilTag de la parte derecha de la pared, pero en el resto de la pared no es capaz de distinguir las posiciones. La zona de la canaleta tampoco es distinguible, pero esto tiene cierta lógica, ya que la canaleta es vertical y homogénea, por lo tanto aunque en el eje X tiene claro donde se sitúa la canaleta, en el eje Y es incapaz de diferencial las diferentes partes de la canaleta.

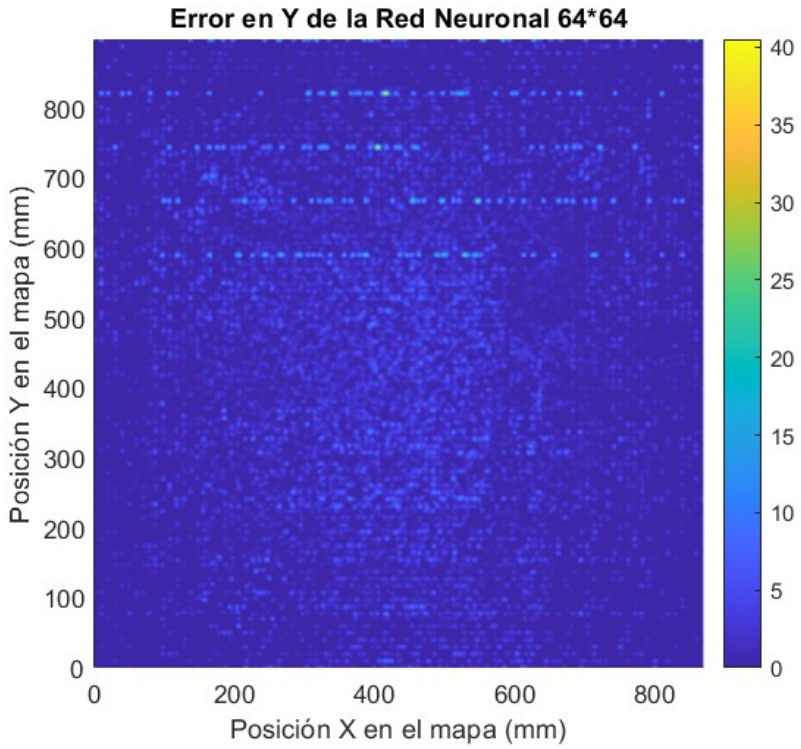


Figure 5.3: Error en Y en entrenamiento con imágenes de resolución 64*64.

Pasando a la red neuronal entrenada con imágenes de entrada de resolución 128*128, existen diferencias notables respecto a los resultados de la red neuronal anterior. En la gráfica del error en el eje X, se puede observar la zona de la canaleta, pero esta vez la zona es más ancha. Esto se puede deber a que las fotografías usadas en esta red neuronal abarcan más espacio de la pared, y por lo tanto se ha mostrado la canaleta en un número mayor de fotografías. Se puede distinguir las tres zonas del AprilTag con una claridad superior a la red neuronal anterior. Aun así, sigue habiendo mayor error en la zonas de AprilTag en comparación a la zona de la canaleta, lo cual indica que aunque la zona del AprilTag haya mejorado, la red neuronal sigue aprendiendo mejor los patrones de la canaleta.

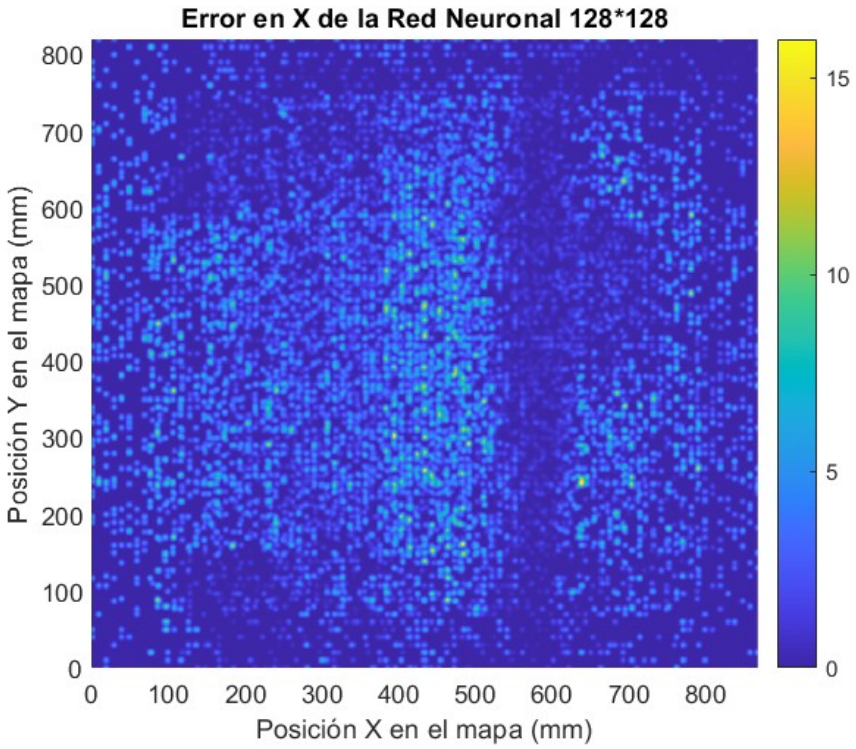


Figure 5.4: Error en X en entrenamiento con imágenes de resolución 128*128.

En cuanto al eje Y, el resultado sigue siendo muy parecido al del eje Y con las imágenes de resolución 64*64. Se puede distinguir algo mejor el AprilTag de la derecha, y se pueden intuir las otras dos AprilTag, pero el error sigue siendo bastante alto en general.

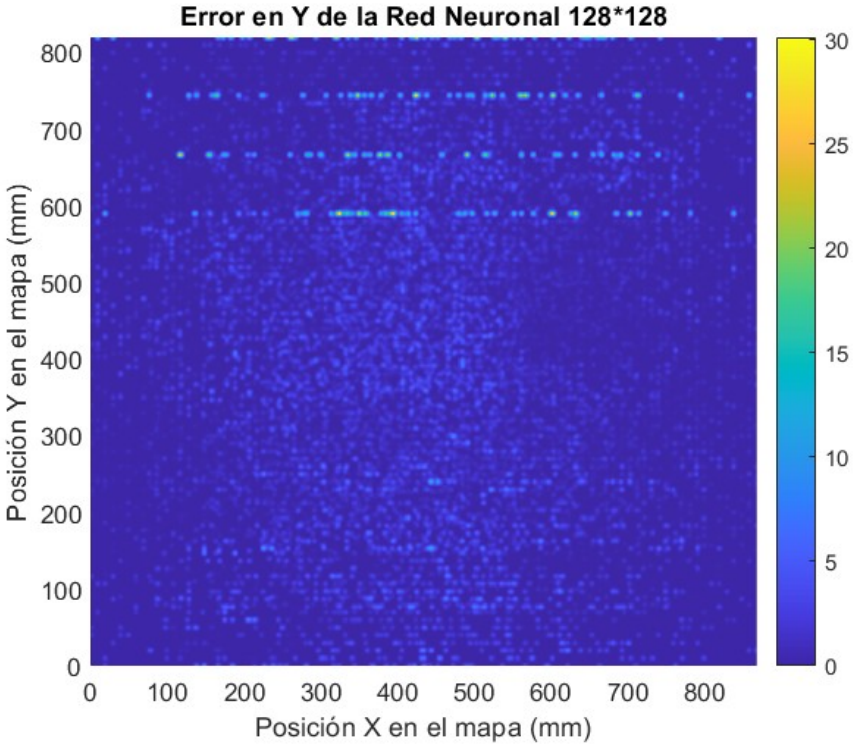


Figure 5.5: Error en Y en entrenamiento con imágenes de resolución 128*128.

6 Conclusiones y trabajos futuros

En el presente trabajo se ha entrenado una red neuronal con diferentes configuraciones para detectar las zonas más significativas de una superficie.

Para realizar el entrenamiento es necesario un conjunto de datos. Este conjunto de datos se ha obtenido mediante una cámara situada en el extremo de un robot. En este caso, se trata del robot Kuka LBR iiwa 7 R800, que tiene 7 grados de libertad de los cuales se ha bloqueado uno para poder trabajar con el como si fuese un robot de 6 grados de libertad convencional. El brazo del robot ha recorrido una superficie elegida (pared con canaleta y AprilTags) haciendo fotos.

A continuación, estas fotos se han segmentado en imágenes de menor tamaño para poder usarlos como datos de entrada de una red neuronal convolucional y tener un tiempo de entrenamiento manejable. Se han usado 3000 fotos originales, y cada una de estas se ha segmentado en 20 imágenes más.

Para el entrenamiento, se ha utilizado la técnica de aprendizaje transferido, que consiste en utilizar una red neuronal previamente entrenada (VGG16) y bloquear todos los pesos sinápticos menos los de las últimas capas. Con esto, la red neuronal es capaz de especializar en la tarea específica que se desea.

Por último, la red neuronal creada ha sido puesta a prueba. Se ha hecho la comparativa de dos redes neuronales creadas, la que ha sido entrenada con imágenes de tamaño 64*64 y la que ha sido entrenada con imágenes 128*128.

La red neuronal con mejores resultados ha sido la entrenada con imágenes de 128*128, ya que ha sido la que menor error ha tenido en las zonas significativas de la pared. Se demuestra que dependiendo de la complejidad del elemento de la pared, la red neuronal puede ser incapaz de entender los patrones del mismo, provocando que no aprenda el elemento. Esto se demuestra especialmente en la red neuronal de 64*64, donde las zonas del AprilTag tienen una tasa de error muy elevada aun siendo claramente zonas muy diferenciadas del resto de la pared.

En contraste con con los AprilTag, el elemento simple de la pared, la canaleta, ha sido capaz de detectarla en ambas redes neuronales, con una tasa de error mínima en las dos.

En resumen, una red neuronal con imágenes de entrada más pequeñas es peor detectando patrones más complejos, y por esta razón, zonas diferenciadas para el ojo humano pueden no significar nada para red neuronal.

Un punto que podría ser de interés como continuación de este trabajo es el entrenamiento de una red neuronal únicamente con las fotos en las que aparezca algún elemento de interés y descartar las fotos donde solo se vea la pared. El uso de imágenes que únicamente contengan información de interés podría mejorar el rendimiento de la red neuronal incluso en los elementos más complejos de la pared.

Referencias bibliográficas

[1]Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. „Understanding of a convolutional neural network“. In: IEEE, Aug. 2017, pp. 1–6 (cit. on p. 7).

[2]Carlos Balaguer et al Antonio Barrientos Luis Felipe Peñin. *Fundamentos de Robótica*. 2nd. Universidad Politécnica de Madrid. Mc Graw Hill (cit. on pp. 12, 24).

[4]Axel Barroso-Laguna, Edgar Riba, Daniel Ponsa, and Krystian Mikolajczyk. „Key.Net: Keypoint Detection by Handcrafted and Learned CNN Filters“. In: (2019) (cit. on p. 10).

[5]Dulari Bhatt, Chirag Patel, Hardik Talsania, et al. „CNN Variants for Computer Vision: History, Architecture, Application, Challenges and Future Scope“. In: *Electronics* 10 (20 Oct. 2021), p. 2470 (cit. on pp. 1, 5, 6, 9).

[6]R. Ezhilarasi and P. Varalakshmi. „Tumor Detection in the Brain using Faster R-CNN“. In: IEEE, Aug. 2018, pp. 388–392 (cit. on p. 11).

[7]Carlos Faria, Flora Ferreira, Wolfram Erlhagen, Sérgio Monteiro, and Estela Bicho. „Position-based kinematics for 7-DoF serial manipulators with global configuration control, joint limit and singularity avoidance“. In: *Mechanism and Machine Theory* 121 (Mar. 2018), pp. 317–334 (cit. on p. 13).

[8]Zhen-Hua Feng, Josef Kittler, Muhammad Awais, Patrik Huber, and Xiao-Jun Wu. „Wing Loss for Robust Facial Landmark Localisation with Convolutional Neural Networks“. In: IEEE, June 2018, pp. 2235–2245 (cit. on p. 10).

[11]Gonzalez and Safabakhsh. „Computer Vision Techniques for Industrial Applications and Robot Control“. In: *Computer* 15 (12 Dec. 1982), pp. 17–32 (cit. on p. 11).

[12]Md Foysal Haque, Hye-Youn Lim, and Dae-Seong Kang. „Object Detection Based on VGG with ResNet Network“. In: IEEE, Jan. 2019, pp. 1–3 (cit. on p. 1).

[13]Asmida Ismail, Siti Anom Ahmad, Azura Che Soh, Khair Hassan, and Hazreen Haizi Harith. „Improving Convolutional Neural Network (CNN) Architecture (miniVGGNet) with Batch Normalization and Learning Rate Decay Factor for Image Classification“. In: *International Journal of Integrated Engineering* 11 (4 Sept. 2019) (cit. on p. 1).

[14]Asifullah Khan, Anabia Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. „A survey of the recent architectures of deep convolutional neural networks“. In: *Artificial Intelligence Review* 53 (8 Dec. 2020), pp. 5455–5516 (cit. on p. 6).

[15]Muhammad Zeeshan Khan, Saad Harous, Saleet Ul Hassan, et al. „Deep Unified Model For Face Recognition Based on Convolution Neural Network and Edge Computing“. In: *IEEE Access* 7 (2019), pp. 72622–72633 (cit. on p. 8).

[16] K. Kreutz-Delgado, M. Long, and H. Seraji. „Kinematic analysis of 7 DOF anthropomorphic arms“. In: IEEE Comput. Soc. Press, 1990, pp. 824–830 (cit. on p. 13).

[17] Yixian Lau. „Understanding how noise affects the accuracy of CNN image classification“. In: 2021 (cit. on p. 1).

[18] Louis Lettry, Michal Perdoch, Kenneth Vanhoey, and Luc Van Gool. „Repeated Pattern Detection Using CNN Activations“. In: *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)* (Mar. 2017), pp. 47–55 (cit. on p. 10).

[19] Peiliang Li, Xiaozhi Chen, and Shaojie Shen. „Stereo R-CNN Based 3D Object Detection for Autonomous Driving“. In: IEEE, June 2019, pp. 7636–7644 (cit. on p. 11).

[20] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. „A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects“. In: *IEEE Transactions on Neural Networks and Learning Systems* 33 (12 Dec. 2022), pp. 6999–7019 (cit. on p. 11).

[21] Guosheng Lin, Chunhua Shen, Anton van den Hengel, and Ian Reid. „Efficient Piecewise Training of Deep Structured Models for Semantic Segmentation“. In: IEEE, June 2016, pp. 3194–3203 (cit. on p. 11).

[22] Shuying Liu and Weihong Deng. „Very deep convolutional neural network based image classification using small training sample size“. In: IEEE, Nov. 2015, pp. 730–734 (cit. on p. 8).

[23] M.J.H. Lum, J. Rosen, M.N. Sinanan, and B. Hannaford. „Optimization of a Spherical Mechanism for a Minimally Invasive Surgical Robot: Theoretical and Experimental Approaches“. In: *IEEE Transactions on Biomedical Engineering* 53 (7 July 2006), pp. 1440–1445 (cit. on p. 11).

[24] Aravindh Mahendran and Andrea Vedaldi. „Understanding deep image representations by inverting them“. In: IEEE, June 2015, pp. 5188–5196 (cit. on p. 9).

[29] Je-Kang Park, Bae-Keun Kwon, Jun-Hyub Park, and Dong-Joong Kang. „Machine learning-based imaging system for surface defect inspection“. In: *International Journal of Precision Engineering and Manufacturing-Green Technology* 3 (3 July 2016), pp. 303–310 (cit. on p. 9).

[30] Yoga Dwi Pranata, Kuan-Chung Wang, Jia-Ching Wang, et al. „Deep learning and SURF for automated classification and detection of calcaneus fractures in CT images“. In: *Computer Methods and Programs in Biomedicine* 171 (Apr. 2019), pp. 27–37 (cit. on p. 9).

[32] Hoo-Chang Shin, Holger R. Roth, Mingchen Gao, et al. „Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning“. In: *IEEE Transactions on Medical Imaging* 35 (5 May 2016), pp. 1285–1298 (cit. on p. 10).

[33] Deepak Tolani, Ambarish Goswami, and Norman I. Badler. „Real-Time Inverse Kinematics Techniques for Anthropomorphic Limbs“. In: *Graphical Models* 62 (5 Sept. 2000), pp. 353–388 (cit. on p. 13).

[34] Lisa Torrey and Jude Shavlik. *Transfer Learning*. IGI Global, 2010, pp. 242–264 (cit. on pp. 2, 3).

[35]Ruohui Wang. *Edge Detection Using Convolutional Neural Network*. 2016, pp. 12–20 (cit. on p. 9).

[36]Biao Yang, Jinmeng Cao, Rongrong Ni, and Yuyu Zhang. „Facial Expression Recognition Using Weighted Mixture Deep Neural Network Based on Double-Channel Facial Images". In: *IEEE Access* 6 (2018), pp. 4630–4640 (cit. on p. 11).

[37]Liang Zheng, Yi Yang, and Qi Tian. „SIFT Meets CNN: A Decade Survey of Instance Retrieval". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40 (5 May 2018), pp. 1224–1244 (cit. on p. 8).

[38]Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. „Learning Deep Features for Discriminative Localization". In: IEEE, June 2016, pp. 2921–2929 (cit. on p. 6).

Páginas Web

[3]*baeldung.com*. 2023. URL: https://www.baeldung.com/cs/epoch-neural-networks (cit. on p. 7).

[9]*github.com*. 2023. URL: https://github.com/Modi1987/KST-Kuka-Sunrise-Toolbox (cit. on p. 15).

[10]*github.com*. 2023. URL: https://github.com/KamaljeetSahoo/AprilTag-Detector (cit. on p. 21).

[25]*mathworks.com*. 2023. URL: https://www.mathworks.com/help/vision/ug/using-the-single-camera-calibrator-app.html (cit. on p. 20).

[26]*mathworks.com*. 2023. URL: https://www.mathworks.com/help/deeplearning/gs/get-started-with-deep-network-designer.html (cit. on p. 42).

[27]*medium.com*. 2023. URL: https://medium.com/data-breach/introduction-to-sift-scale-invariant-feature-transform-65d7f3a72d40 (cit. on p. 9).

[28]*paperswithcode.com*. 2023. URL: https://paperswithcode.com/method/max-pooling (cit. on p. 6).

[31]*researchgate.net*. 2023. URL: https://www.researchgate.net/figure/Manipulator-generic-structure-joint-variables-and-DH-frames-assigned-The-7-DoF_fig1_320895915 (cit. on p. 18).

eman ta zabal zazu
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 MÁSTER

ANEXO A: PROGRAMAS FUENTE

EVALUACIÓN DE REGIONES SIGNIFICATIVAS PARA EL ENTRENAMIENTO DE UNA RED NEURONAL A TRAVÉS DE FOTOGRAFÍAS DE UNA SUPERFICIE

Estudiante	Millán, Fernández de Landa, Mikel
Director/Directora	Zulueta, Guerrero, Ekaitz
Departamento	Ingeniería de Sistemas y Automática
Curso académico	2022-2023

Bilbao, 16 de Julio de 2023

Programas fuente

A

A.1 Seguimiento de brazo robótico a marcador Tag

A.1.1 Main.m

```
close all;
clear all;
clc;
%% Estabilizar conexion con Robot
warning('off');
ip='172.31.1.147'; % IP del Robot
% Comienza la conexion con el servidor
global t_Kuka;
t_Kuka=net_establishConnection( ip );

if ~exist('t_Kuka','var') || isempty(t_Kuka) || strcmp(
   t_Kuka.Status,'closed')
  warning('Connection could not be establised, script
     aborted');
  return; %en caso de no detectar el robot, el programa
     se para inmediatamente
else
%% Inicializacion de variables
    q1 = 0.62;
    q2 = -1.36;
    q3 = 0;
    q4 = 0;
    q5 = 0;
    q6 = 0;
    q7 = 0;
    Deltax = 0;
    Deltay = 0;
    Deltaz = 0;
%% Inicio movimiento
```

```
    % Posicion de descanso
    relVel=0.25; % sobrepasar velocidad relativa de las
       articulaciones
    pos={0, 0, 0, 0, 0, 0, 0};
    movePTPJointSpace( t_Kuka , pos, relVel);

    % Posicion de inicio

    pos={q1, q2, q3, q4, q5, q6, q7};
    movePTPJointSpace( t_Kuka , pos, relVel);
    for d = 1:2
        [R07Directo]= CinematicaDirecta (q1, q2, q3, q4,
           q5, q6, q7);

        [MatrizRotacion, VectorTraslacion, id] = Camara;

        [a, b, c, x, y, z]= Corregir (VectorTraslacion,
           MatrizRotacion, R07Directo);

        [q1, q2, q3, q4, q5, q6, q7]= CinematicaInversa (
           a, b, c, x, y, z);

        pos={q1, q2, q3, q4, q5, q6, q7};
        movePTPJointSpace( t_Kuka , pos, relVel);
    end

    pos={0, 0, 0, 0, 0, 0, 0};
    movePTPJointSpace( t_Kuka , pos, relVel);
end
%% apagar el servidor

net_turnOffServer( t_Kuka );
fclose(t_Kuka);
```

A.1.2 CinematicaDirecta.m

```
function [R07]= CinematicaDirecta (q1, q2, q3, q4, q5, q6
   , q7)

%Longitudes necesarias del robot
L1 = 340; %mm
L2 = 400;
L3 = 400;

R01 = [cos(q1) 0 -sin(q1) 0 ; sin(q1) 0 cos(q1) 0 ; 0 -1
   0 L1 ; 0 0 0 1];
R12 = [cos(q2) 0 sin(q2) 0 ; sin(q2) 0 -cos(q2) 0 ; 0 1 0
    0 ; 0 0 0 1];
R23 = [cos(q3) 0 sin(q3) 0 ; sin(q3) 0 -cos(q3) 0 ; 0 1 0
    L2; 0 0 0 1];
R34 = [cos(q4) 0 -sin(q4) 0 ; sin(q4) 0 cos(q4) 0 ; 0 -1
   0 0 ; 0 0 0 1];
R45 = [cos(q5) 0 -sin(q5) 0 ; sin(q5) 0 cos(q5) 0 ; 0 -1
   0 L3 ; 0 0 0 1];
R56 = [cos(q6) 0 sin(q6) 0 ; sin(q6) 0 -cos(q6) 0 ; 0 1 0
    0; 0 0 0 1];
R67 = [cos(q7) -sin(q7) 0 0 ; sin(q7) cos(q7) 0 0 ; 0 0 1
    0 ; 0 0 0 1];

R07 = R01*R12*R23*R34*R45*R56*R67;
end
```

A.1.3 CinematicaInversa.m

```
function [q1, q2, q3, q4, q5, q6, q7]= CinematicaInversa
    (a, b, c, x, y, z)
% Autor: Mikel Millan

% Esta funcion esta disenada para que con las entradas de
     las coordenadas
% x, y, z del extremo del robot y los angulos de la
    direccion a, b y c este
% devuelva los angulos de cada una de las articulaciones.

% El programa esta particularizado para el robot KUKA LBR
     iiwa R800.

%% Parametros del robot
% Longitudes del robot
L1 = 340; %mm
L2 = 400;
L3 = 400;
% Lfin = 126;

% Los angulos maximos de cada articulacion

Q1max = 2.967; % rad = 170
Q2max = 2.094; % rad = 120
% Q3max = 2.967;
Q4max = 2.094;

%% Calculo de q1, q2, q3 y q4

% El problema para hacerlo mas sencillo se va a dividir
    en dos partes, en
% la parte de la traslacion y en la parte de la rotacion.
     Los primeros
% cuatro ejes se dedicaran a la parte de la traslacion,
    ya que con estos se
% ocupa practicamente la totalidad de la longitud del
    robot.
% Los ultimos tres ejes se dedicaran a rotar el robot.
```

```
% Para la traslacion solo se necesitan utilizar tres ejes
   , por lo tanto se
% bloqueara una articulacion y asi simplificar el
   problema.

q3 = 0;

q1 = atan(y/x); % unica solucion.

zu = z-L1; % Solo se tiene en cuenta la parte del robot
   que va a variarse
            % en el eje z.

%v Se procede a calcular el punto maximo si el primer
   sunto supera los
%limites del robot.

RadioMax = 790;
RadioPunto = sqrt(x^2 + y^2 +zu^2); % Radio de la esfera.

if RadioPunto > RadioMax
    % multiplicador para obtener el punto maximo que se
       encuentre en la
    % misma recta que el punto original.
    Multiplicador = sqrt((RadioMax)^2/(x^2+y^2+zu^2));
    x = x*Multiplicador;
    y = y*Multiplicador;
    zu = zu*Multiplicador;
end

C4 = ((x^2)+(y^2)+(zu^2)-(L2^2)-(L3^2))/(2*L2*L3);

q4aux(1) = -acos(C4);
q4aux(2) = acos(C4);

% A diferencia de las formulas habituales se ponen los
   dos signos igual
% ya que en el sin(q4) ya esta metido el signo.

q2aux(1) = atan(zu/sqrt(x^2+y^2))+atan((L3*sin(q4aux(1)))
   / ...
    (L2+L3*cos(q4aux(1))))-pi/2;
```

```
q2aux(2) = atan(zu/sqrt(x^2+y^2))+atan((L3*sin(q4aux(2)))
   / ...
    (L2+L3*cos(q4aux(2))))-pi/2;


if q1 < -Q1max
    q1 = -Q1max;
elseif q1 > Q1max
    q1 = Q1max;
end

% Seleccion del camino a recorrer con q2 y q4.

if q2aux(2) < -Q2max || q4aux(2) < -Q4max ...
    || q4aux(2) > Q4max || q2aux(2) > Q2max
    q2 = q2aux(1);
    q4 = q4aux(1);
else
    q2 = q2aux(2);
    q4 = q4aux(2);
end

if q2 < -Q2max
    q2 = -Q2max;
elseif q2 > Q2max
    q2 = Q2max;
elseif q4 < -Q4max
    q4 = -Q4max;
elseif q4 > Q4max
    q4 = Q4max;
end

% El objetivo de esta funcion es orientar el robot. Para
   ello se calcularan
% los ultimos tres angulos del robot.

Q5max = 2.967; % rad = 170
Q6max = 2.094; % rad = 120
Q7max = 3.054; % rad = 175

% Matrices de rotacion calculados con DH.
```

```
R01 = [cos(q1) 0 -sin(q1) ; sin(q1) 0 cos(q1) ; 0 -1 0];
R12 = [cos(q2) 0 sin(q2) ; sin(q2) 0 -cos(q2) ; 0 1 0];
R23 = [cos(q3) 0 sin(q3) ; sin(q3) 0 -cos(q3) ; 0 1 0];
R34 = [cos(q4) 0 -sin(q4) ; sin(q4) 0 cos(q4) ; 0 -1 0];

R04 = R01*R12*R23*R34;
R40 = inv(R04);
% Las matrices de rotacion, para calcular la matriz de la
    posicion final.

Rx = [1 0 0 ; 0 cos(a) -sin(a) ; 0 sin(a) cos(a)];
Ry = [cos(b) 0 sin(b) ; 0 1 0 ; -sin(b) 0 cos(b)];
Rz = [cos(c) -sin(c) 0 ; sin(c) cos(c) 0 ; 0 0 1];

R = Rz*Ry*Rx;


% Calcular R47 para calcular las ultimas articulaciones.

R47 = R40*R;

q6 = acos(R47(3,3));
q5 = atan2(R47(2,3),R47(1,3));
q7 = atan2(R47(3,2),-R47(3,1));

if q5 > Q5max
    q5 = Q5max;
elseif q5 < -Q5max
    q5 = -Q5max;
end
if q6 > Q6max
    q6 = Q6max;
elseif q6 < -Q6max
    q6 = -Q6max;
end
if q7 > Q7max
    q7 = Q7max;
elseif q7 < -Q7max
    q7 = -Q7max;
end

end
```

A.1.4 Camara.m

```
function [MatrizRotacion, VectorTraslacion, id] = Camara
cam = webcam(2);
load('calibrationSession.mat');
intrinsics = calibrationSession.CameraParameters.
   Intrinsics;
tagSize = 60; % mm

while true

    I=snapshot(cam);
    imshow(I);

    [id,~,pose] = readAprilTag(I,"tag36h11",intrinsics,
       tagSize);

    for i = 1:length(pose)
        MatrizRotacion=pose(i).Rotation;
        VectorTraslacion=pose(i).Translation;
    end

    if id(i)==2
        break
    end
end
end
```

A.1.5 Corregir.m

```
abc78 = rotm2eul(MatrizRotacion(1:3,1:3),'ZYX');
abc78(2)=abc78(2);
abc78(3)=abc78(3);
abc78(1)
if abc78(1)<=pi/2 && abc78(1)>=-pi/2
    R78Acond = eul2rotm([-abc78(1) abc78(3) -abc78(2)]);
else
    R78Acond = eul2rotm([-abc78(1) -abc78(3) +abc78(2)]);
end

Rot07 = R07(1:3,1:3);

R08 = Rot07*R78Acond;
abc08 = rotm2eul(R08,'ZYX');
AngX = abc08(3);
AngY = abc08(2);
AngZ = abc08(1);

%% Correccion de la camara
% Esto se ha hecho de forma experimental
Dx = VectorTraslacion(1);
Dy = VectorTraslacion(2);
Dz = VectorTraslacion(3);

Dz = 3.4347*Dz + 23.75;

%% Distacia que se quiere mantener con la camara

Dz = Dz-526;
R78Tot= [R78Acond(1,1:3) -Dy; R78Acond(2,1:3) Dx;
   R78Acond(3,1:3) Dz ; 0 0 0 1];
R08Tot = R07*R78Tot;

%% Calculo Matriz
%R = [-Dy ; Dx ; Dz ; 1];
R=[0 ; 0 ; 0 ; 1];
% Dist = R07*R;
Dist = R08Tot*R;
x = Dist(1);
y = Dist(2);
```

```
z = Dist(3);
```

A.2 Entrenamiento de la Red

A.2.1 MainFotos.m

```
close all;
clear all;
clc;
%% Estabilizar conexion con Robot
warning('off');
ip='172.31.1.147'; % IP del Robot
% Comienza la conexion con el servidor
global t_Kuka;
t_Kuka=net_establishConnection( ip );

if ~exist('t_Kuka','var') || isempty(t_Kuka) || strcmp(
   t_Kuka.Status,'closed')
  warning('Connection could not be establised, script
     aborted');
  return; %en caso de no detectar el robot, el programa
     se para inmediatamente
else
    %% Setup Fotos
    %Inicializar webcam
    cam = webcam(2);

    %% Inicio movimiento

    % Posicion de descanso
    relVel=0.25; % sobrepasar velocidad relativa de las
       articulaciones

    pos={0, 0, 0, 0, 0, 0, 0};
    movePTPJointSpace( t_Kuka , pos, relVel);

    % Posicion de inicio
    x = -141;
    y = -565;
    z = 650;
    % Los angulos por experimentacion
```

```
a = 45*pi/180;
b = -90*pi/180;
c = 0*pi/180;

Nx=50;
Ny = 60;

HorizontalTotal = sqrt(2)*7*(Nx-1);
VerticalTotal = 10*(Ny-1);

[q1, q2, q3, q4, q5, q6, q7]= CinematicaInversa (a, b
   , c, x, y, z);

pos={q1, q2, q3, q4, q5, q6, q7};
movePTPJointSpace( t_Kuka , pos, relVel);
for i=1:1:Nx %50
    for j=1:1:Ny % 60
        % Calculo x, y, z
        x = x-(i-1)*7;
        y = y+(i-1)*7;
        z = z-(j-1)*10;
        %Nombres Fotos

        NombreHorizontal = -HorizontalTotal/2 + sqrt
           (2)*7*(i-1);
        NombreVertical = VerticalTotal/2 - 10*(j-1);

        [q1, q2, q3, q4, q5, q6, q7] = ...
            CinematicaInversa (a, b, c, x, y, z);

        pos={q1, q2, q3, q4, q5, q6, q7};
        movePTPJointSpace( t_Kuka , pos, relVel);
        % Fotos
        pause(0.5)
        imagen = snapshot(cam);
        NombreFichero=strcat('ImagenesMas\F',sprintf(
           '%.1f,%.1f', ...
            NombreHorizontal,NombreVertical),'.jpg');
        imwrite(imagen,NombreFichero);
        x = -141;
        y = -565;
        z = 650;
```

```
        end
    end
    pos={0, 0, 0, 0, 0, 0, 0};
    movePTPJointSpace( t_Kuka , pos, relVel);

end
%% apagar el servidor

net_turnOffServer( t_Kuka );
fclose(t_Kuka);
```

A.2.2 DividirFotos.m

```
clc
close all
clear all

delete(['C:\Users\Mikel\Desktop\TFM-Publicaciones' ...
    '\RED_NEURONAL\Imagenes\ImagenesTrain\*'])
delete(['C:\Users\Mikel\Desktop\TFM-Publicaciones' ...
    '\RED_NEURONAL\Imagenes\ImagenesTest\*']);
delete(['C:\Users\Mikel\Desktop\TFM-Publicaciones' ...
    '\RED_NEURONAL\Imagenes\ImagenesVal\*']);

SalidasTrain=[];
SalidasVal=[];
SalidasTest=[];

Ptrain=0.7;
Pval=0.15;
Ptest=0.15;
%% Nombres de las fotos

NxmaxSub = 640;
NymaxSub = 480;

Nx=64;
Ny=64;

Nxmax = 50;
Nymax = 60;

DistImagen= 492; %mm
DistPixel = DistImagen/NxmaxSub;

HorizontalTotal = sqrt(2)*7*(Nxmax-1);
VerticalTotal = 10*(Nymax-1);

for i=1:1:Nxmax %50
    for j=1:1:Nymax % 60
        %Nombres Fotos
```

```
NombreHorizontal = -HorizontalTotal/2+sqrt(2)*7*(
   i-1);
NombreVertical = VerticalTotal/2-10*(j-1);

NombreFichero=sprintf('F%.1f,%.1f.jpg' ...
    ,NombreHorizontal,NombreVertical);
I=imread(NombreFichero);

for i1=1:100:NxmaxSub-Nx
    for i2=1:100:NymaxSub-Ny

        NombreHorizontalSub = NombreHorizontal-
           DistPixel*...
            (NxmaxSub-2*Nx)/2+i1*DistPixel;
        NombreVerticalSub = NombreVertical+
           DistPixel*...
            (NymaxSub-2*Ny)/2-i2*DistPixel;
        Isub=I(i2:i2+Nx-1,i1:i1+Ny-1,:);
        Prob=random('Uniform',0,1,1,1);
        if Prob<Ptrain
            NombreFichero=strcat('ImagenesTrain\F
               ',sprintf ...
                ('%.1f,%.1f.jpg',
                   NombreHorizontalSub,
                   NombreVerticalSub));
            imwrite(Isub,NombreFichero);
        else
            if Prob<Ptrain+Pval
                NombreFichero=strcat('ImagenesVal
                   \F',sprintf ...
                    ('%.1f,%.1f.jpg',
                       NombreHorizontalSub, ...
                    NombreVerticalSub));
                imwrite(Isub,NombreFichero);
            else
                NombreFichero=strcat('
                   ImagenesTest\F',sprintf ...
                    ('%.1f,%.1f.jpg',
                       NombreHorizontalSub, ...
                    NombreVerticalSub));
                imwrite(Isub,NombreFichero);
            end
```

```
                end
            end
        end
    end
end
ImagenesInputTrain=imageDatastore("ImagenesTrain\", ...
    "FileExtensions",".jpg");
LTrain=length(ImagenesInputTrain.Files);
[SalidasTrain, xmaxTrain, ymaxTrain]=Irakurri(
   ImagenesInputTrain.Files, ...
    LTrain);
SalidasDSTrain=arrayDatastore(SalidasTrain);
ImagenesConSalidasDSTrain=combine(ImagenesInputTrain,
   SalidasDSTrain);

ImagenesInputTest=imageDatastore("ImagenesTest\","
   FileExtensions",".jpg");
LTest=length(ImagenesInputTest.Files);
[SalidasTest, xmaxTest, ymaxTest]=Irakurri(
   ImagenesInputTest.Files, LTest);
SalidasDSTest=arrayDatastore(SalidasTest);
ImagenesConSalidasDSTest=combine(ImagenesInputTest,
   SalidasDSTest);

ImagenesInputVal=imageDatastore("ImagenesVal\","
   FileExtensions",".jpg");
LVal=length(ImagenesInputVal.Files);
[SalidasVal, xmaxVal, ymaxVal]=Irakurri(ImagenesInputVal.
   Files, LVal);
SalidasDSVal=arrayDatastore(SalidasVal);
ImagenesConSalidasDSVal=combine(ImagenesInputVal,
   SalidasDSVal);

save all
```

A.2.3 CodigoEntrenamiento.m

```
%% Import Data
%Import training and validation data.
dsTrain = ImagenesConSalidasDSTrain;
dsValidation = ImagenesConSalidasDSVal;

%% Set Training Options
%Specify options to use when training.
opts = trainingOptions("sgdm",...
    "ExecutionEnvironment","gpu",...
    "InitialLearnRate",0.0011,...
    "L2Regularization",0.0001,...
     "LearnRateSchedule","piecewise", ...
    "LearnRateDropFactor",0.2, ...
    "LearnRateDropPeriod",5, ...
    "MaxEpochs",20,...
    "MiniBatchSize",64,...
    "Shuffle","every-epoch",...
    "ValidationFrequency",250,...
    "Plots","training-progress",...
    "ValidationData",dsValidation);

%% Create Array of Layers
layers = [
    imageInputLayer([128 128 3],"Name","imageinput")
    convolution2dLayer([3 3],64,"Name","conv1_1","Padding
       ", ...
    [1 1 1 1],"WeightL2Factor",0)
    reluLayer("Name","relu1_1")
    convolution2dLayer([3 3],64,"Name","conv1_2","Padding
       ",[1 1 1 1],"WeightL2Factor",0)
    reluLayer("Name","relu1_2")
    maxPooling2dLayer([2 2],"Name","pool1","Stride",[2
       2])
    convolution2dLayer([3 3],128,"Name","conv2_1","
       Padding",[1 1 1 1],"WeightL2Factor",0)
    reluLayer("Name","relu2_1")
    convolution2dLayer([3 3],128,"Name","conv2_2","
       Padding",[1 1 1 1],"WeightL2Factor",0)
    reluLayer("Name","relu2_2")
```

```
maxPooling2dLayer([2 2],"Name","pool2","Stride",[2
   2])
convolution2dLayer([3 3],256,"Name","conv3_1","
   Padding",[1 1 1 1],"WeightL2Factor",0)
reluLayer("Name","relu3_1")
convolution2dLayer([3 3],256,"Name","conv3_2","
   Padding",[1 1 1 1],"WeightL2Factor",0)
reluLayer("Name","relu3_2")
convolution2dLayer([3 3],256,"Name","conv3_3","
   Padding",[1 1 1 1],"WeightL2Factor",0)
reluLayer("Name","relu3_3")
maxPooling2dLayer([2 2],"Name","pool3","Stride",[2
   2])
convolution2dLayer([3 3],512,"Name","conv4_1","
   Padding",[1 1 1 1],"WeightL2Factor",0)
reluLayer("Name","relu4_1")
convolution2dLayer([3 3],512,"Name","conv4_2","
   Padding",[1 1 1 1],"WeightL2Factor",0)
reluLayer("Name","relu4_2")
convolution2dLayer([3 3],512,"Name","conv4_3","
   Padding",[1 1 1 1],"WeightL2Factor",0)
reluLayer("Name","relu4_3")
maxPooling2dLayer([2 2],"Name","pool4","Stride",[2
   2])
convolution2dLayer([3 3],512,"Name","conv5_1","
   Padding",[1 1 1 1],"WeightL2Factor",0)
reluLayer("Name","relu5_1")
convolution2dLayer([3 3],512,"Name","conv5_2","
   Padding",[1 1 1 1],"WeightL2Factor",0)
reluLayer("Name","relu5_2")
convolution2dLayer([3 3],512,"Name","conv5_3","
   Padding",[1 1 1 1],"WeightL2Factor",0)
reluLayer("Name","relu5_3")
maxPooling2dLayer([2 2],"Name","pool5","Stride",[2
   2])
fullyConnectedLayer(1000,"Name","Benito")
reluLayer("Name","relu6")
dropoutLayer(0.5,"Name","drop6")
fullyConnectedLayer(100,"Name","Patxi")
reluLayer("Name","relu7")
dropoutLayer(0.5,"Name","drop7")
fullyConnectedLayer(2,"Name","fc")
```

```
    regressionLayer("Name","regressionoutput")];

%% Train Network
% Train the network using the specified options and
   training data.
[net, traininfo] = trainNetwork(dsTrain,layers,opts);

save SareaIkasita128 net
```

A.2.4 Irakurri.m

```
function [data, xmax, ymax]=Irakurri(Filename,L)
data1 = [];
data = [];
for i=1:L
    Refkoma=strfind(Filename(i,1),',');
    RefF=strfind(Filename(i,1),'F');
    x=str2double(Filename{i,1}(RefF{1,1}(2)+1:Refkoma
       {1,1}-1));
    y=str2double(Filename{i,1}(Refkoma{1,1}+1:end-4));
    data1=[data1;[x,y]];
end
xmax = max(data1(:,1));
ymax = max(data1(:,2));
data(:,1) = data1(:,1)/xmax;
data(:,2) = data1(:,2)/ymax;
end
```

A.2.5 Irakurri.m

```
clc
clear all

load redEntrenada.mat

NuevaMatrizx = [];
NuevaMatrizy = [];

NuevaMatrizx128 = [];
NuevaMatrizy128 = [];

xmin = round(min(MatrizTotal(:,3)));
xmax = round(max(MatrizTotal(:,3)));
ymin = round(min(MatrizTotal(:,4)));
ymax = round(max(MatrizTotal(:,4)));
xrange = xmax-xmin;
yrange = ymax-ymin;

Matrizusada = round(MatrizTotal);
Matrizusada(:,3) = Matrizusada(:,3)-xmin+1;
Matrizusada(:,4) = Matrizusada(:,4)-ymin+1;

ptsx = linspace(0, xrange, xrange+1);
ptsy = linspace(0, yrange, yrange+1);

[xG, yG] = meshgrid(-5:5);
sigma = 2.5;
g = exp(-xG.^2./(2.*sigma.^2)-yG.^2./(2.*sigma.^2));
g = g./sum(g(:));

for i = 1 : length(Matrizusada(:,2))
       NuevaMatrizx(Matrizusada(i,4),Matrizusada(i,3))=
          Matrizusada(i,1);
end

for i = 1 : length(Matrizusada(:,2))
       NuevaMatrizy(Matrizusada(i,4),Matrizusada(i,3))=
          Matrizusada(i,2);
end
```

```
%% Figurak ploteatu 64*64

figure(1)
title('64*64-ko errorearen heatmap-a')
subplot(2,2,1)
xlabel('X')
ylabel('Y')
hold on
imagesc(ptsx, ptsy, conv2(NuevaMatrizx, g, 'same'));
axis equal;
set(gca, 'XLim', ptsx([1 end]), 'YLim', ptsy([1 end]), '
   YDir', 'normal');
colorbar
subplot(2,2,2)
xlabel('X')
ylabel('Y')
hold on
imagesc(ptsx, ptsy, conv2(NuevaMatrizy, g, 'same'));
axis equal;
set(gca, 'XLim', ptsx([1 end]), 'YLim', ptsy([1 end]), '
   YDir', 'normal');
colorbar
hold on

MatrizTotal = [];
Matrizusada = [];
NuevaMatrizx =[];
NuevaMatrizy = [];
ptsx = [];
ptsy = [];

load redEntrenada128.mat

xmin = round(min(MatrizTotal(:,3)));
xmax = round(max(MatrizTotal(:,3)));
ymin = round(min(MatrizTotal(:,4)));
ymax = round(max(MatrizTotal(:,4)));
xrange = xmax-xmin;
yrange = ymax-ymin;

Matrizusada = round(MatrizTotal);
Matrizusada(:,3) = Matrizusada(:,3)-xmin+1;
```

```
Matrizusada(:,4) = Matrizusada(:,4)-ymin+1;

ptsx = linspace(0, xrange, xrange+1);
ptsy = linspace(0, yrange, yrange+1);

for i = 1 : length(Matrizusada(:,2))
       NuevaMatrizx(Matrizusada(i,4),Matrizusada(i,3))=
          Matrizusada(i,1);
end

for i = 1 : length(Matrizusada(:,2))
       NuevaMatrizy(Matrizusada(i,4),Matrizusada(i,3))=
          Matrizusada(i,2);
end

subplot(2,2,3)
xlabel('X')
ylabel('Y')
hold on
imagesc(ptsx, ptsy, conv2(NuevaMatrizx, g, 'same'));
axis equal;
set(gca, 'XLim', ptsx([1 end]), 'YLim', ptsy([1 end]), '
   YDir', 'normal');
colorbar
subplot(2,2,4)
xlabel('X')
ylabel('Y')
hold on
imagesc(ptsx, ptsy, conv2(NuevaMatrizy, g, 'same'));
axis equal;
set(gca, 'XLim', ptsx([1 end]), 'YLim', ptsy([1 end]), '
   YDir', 'normal');
colorbar
```