



COMILLAS
UNIVERSIDAD PONTIFICIA

ICAI

MASTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

TRABAJO DE FINAL DE MASTER

SISTEMA DE VISIÓN ARTIFICIAL BASADO EN UN DATASET SINTÉTICO PARA UNA APLICACIÓN DE PICK AND PLACE

Autor: Ignacio Ortiz de Zúñiga Mingot

Directores:

Álvaro Jesús López López

Ignacio de Rodrigo Tobías

Madrid

July de 2022

Copyright © 2022 Ignacio Ortiz de Zúñiga Mingot

Este trabajo fue escrito con \LaTeX y compilado en $\text{\TeX} \text{maker}$ usando la distribución $\text{\TeX} \text{Live}$ -2013. Las familias de fuentes usadas son Bitstream Charter, Utopia, Bookman, and Computer Modern. A menos que se indique lo contrario, todas las figuras fueron creadas por el autor usando Lucidchart[®], draw.io[®] y Python[®].

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

.....
.....

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: (Nombre del alumno)

Fecha://

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: (Nombre del Director)

Fecha://



COMILLAS
UNIVERSIDAD PONTIFICIA

ICAI

MASTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

TRABAJO DE FINAL DE MASTER

SISTEMA DE VISIÓN ARTIFICIAL BASADO EN UN DATASET SINTÉTICO PARA UNA APLICACIÓN DE PICK AND PLACE

Autor: Ignacio Ortiz de Zúñiga Mingot

Directores:

Álvaro Jesús López López

Ignacio de Rodrigo Tobías

Madrid

July de 2022

Copyright © 2022 Ignacio Ortiz de Zúñiga Mingot

Este trabajo fue escrito con \LaTeX y compilado en $\text{\TeX} \text{maker}$ usando la distribución $\text{\TeX} \text{Live}$ -2013. Las familias de fuentes usadas son Bitstream Charter, Utopia, Bookman, and Computer Modern. A menos que se indique lo contrario, todas las figuras fueron creadas por el autor usando Lucidchart[®], draw.io[®] y Python[®].

*Para todos los que me habéis soportado
durante estos largos años de carrera*

Agradecimientos

En primer lugar me gustaría dar las gracias a todos los miembros de *CICLAB* por hacerme sentir parte de un grupo y un proyecto. Gracias por todas las horas de duro en la cueva intentado resolver complejos problemas intercalados por risas. Gracias por apoyarme en mis momentos de frustración y de locura. Y sobretodo gracias por soportarme. En especial quiero darle las gracias a Álvaro Lopez Lopez y a Ignacio Rodrigo de Tobías por darme la oportunidad de colaborar en este proyecto y por guiarme y apoyarme.

También quiero darle las gracias a mis amigos, compañeros de clase y profesores. Si no fuese por vosotros habría perdido el norte hace mucho tiempo. Gracias por ayudarme y soportarme durante todos estos años. Se cierra una etapa de mi vida (escolar) y comienza una nueva (laboral) pero no quiero que cambie todo. Espero poder seguir contando con vosotros durante muchos mas años. Y gracias a *Salita Warriors* por esas duras tardes de domingo estudiando en equipo. Sin todo vuestro apoyo y paciencia no estaría donde estoy ahora.

Por último pero no por ello menos importante quiero agradecérselo a mis padres por darme esta oportunidad y por creer en mí. Gracias de todo corazón por ayudarme y darme tanto. Y a Marina por estar y escucharme en mis momentos de delirio cuando me quedaba atascado en el proyecto. Gracias por creer en mí.

Índice general

Resumen	xv
Abstract	xxi
1. Introducción	1
2. Estado de la cuestión	5
2.1. Procesado de la imagen	5
2.1.1. Filtros, detección de borde y detección de formas	5
2.1.2. Redes neuronales convolucionales	10
2.2. Robótica industrial	12
3. Metodología de trabajo	15
3.1. Motivación	15
3.2. Objetivos	15
3.3. Arquitectura del sistema	16
3.4. Herramientas	17
3.5. Cronograma	18
4. Generación de un dataset	21
4.1. Estructura del dataset	22
4.2. Herramientas	23
4.2.1. Blender	24
4.2.2. BlenderProc	24
4.2.3. Aruco	24
4.3. Arquitectura del generador de datasets	24
4.4. Posprocesado	26
4.5. Aleatoriedad	29
4.6. Resultados	30
4.7. Conclusiones y futuros desarrollos	31
5. Sistema de visión artificial	33
5.1. YOLO	34
5.1.1. Estructura	35
5.1.2. Entrenamiento	36
5.1.3. Resultados	37
5.2. Tiny YOLO	42
5.2.1. Estructura	42
5.2.2. Entrenamiento	42

Índice general

5.2.3. Resultados	43
5.3. Regresor	47
5.3.1. Estructura	47
5.3.2. Entrenamiento	48
5.3.3. Resultados	49
6. Conclusiones y futuros desarrollos	53
6.1. Sistema de detección de puntos de agarre	53
6.2. Dataset sintético	54
7. Objetivos de desarrollo sostenible	55
A. Estructura YOLOv5	57
B. Camara RGBD	59
Bibliografía	63

Índice de figuras

Figura 2.1. Proceso de convolución en matrices	6
Figura 2.2. Comparativa de algoritmos de detección de borde	7
Figura 2.3. Detección de borde por algoritmo de Canny	8
Figura 2.4. Representación de la transforma de Hough	9
Figura 2.5. Estructura de AlexNet	11
Figura 2.6. Errores cometidos en ImageNet en los últimos años	12
Figura 2.7. Brazo robótico IRB 1400 YuMi	13
Figura 3.1. Esquema de la arquitectura del sistema	17
Figura 3.2. Diagrama de las etapas del sistema	17
Figura 3.3. Diagrama de gantt del proyecto	19
Figura 4.1. Esquema de la arquitectura del sistema	25
Figura 4.2. Proceso de generación de imágenes	27
Figura 4.2. Aleatoriedad de una distribución uniforma generada por numpy	30
Figura 4.3. Aleatoriedad del comando <i>choice</i> del paquete <i>random</i>	30
Figura 4.4. Aleatoriedad de <i>uniformSO3</i> del paquete BlenderProc	30
Figura 4.4. Muestra de imágenes obtenidas mediante el generador de imágenes	30
Figura 4.4. Instancias de cada pieza empleadas para entrenar el modelo basado en YOLO	32
Figura 4.5. Instancias de cada zona de interés empleadas para entrenar el modelo basado en Tiny YOLO	32
Figura 4.6. Distribución de las proyecciones normales de los puntos de agarre de G1	32
Figura 5.1. Esquema de la arquitectura del sistema	34
Figura 5.2. Esquema del funcionamiento de YOLO	35
Figura 5.3. Evaluación de instancias, forma y ubicación del <i>dataset</i> empleado en YOLO	38
Figura 5.4. Proceso de evolución de los hiperparámetros de YOLO	39
Figura 5.5. Resultados de la evolución de los hiperparámetros de YOLO	40
Figura 5.6. Métricas de la red neuronal YOLO (<i>mAP 0.5:0.95, precision & recall</i>)	40
Figura 5.7. Entrenamiento de la red neuronal YOLO (<i>bounding box loss, class loss & object loss</i>)	40
Figura 5.8. Validación de la red neuronal YOLO (<i>bounding box loss, class loss & object loss</i>)	40
Figura 5.9. Matriz de confusión en validación de YOLO	41
Figura 5.10. <i>Precision y Recall</i> en validación de YOLO	41
Figura 5.11. Curva F1 en validación de YOLO	41
Figura 5.12. Comparativa de los diferentes modelos de YOLO	42
Figura 5.13. Evaluación de instancias, forma y ubicación del <i>dataset</i> basado en G1 empleado en TINY YOLO	44

Índice de figuras

Figura 5.14. Métricas de la red neuronal TINY YOLO (<i>mAP 0.5:0.95, precision & recall</i>)	45
Figura 5.15. Entrenamiento de la red neuronal TINY YOLO (<i>bounding box loss, class loss & object loss</i>)	45
Figura 5.16. Validación de la red neuronal TINY YOLO (<i>bounding box loss, class loss & object loss</i>)	45
Figura 5.17. Matriz de confusión en validación de TINY YOLO	46
Figura 5.18. <i>Precision</i> y <i>Recall</i> en validación de TINY YOLO	46
Figura 5.19. Curva F1 en validación de TINY YOLO	46
Figura 5.20. Estructura del regresor	48
Figura 5.21. Comparativa de las fases de entrenamiento del regresor	50
Figura 5.22. Comparativa de las fases de validación del regresor	51
Figura 7.1. Objetivos de desarrollo sostenible aprobados en 2015	55
Figura A.1. Estructura de YOLOv5l	58
Figura B.1. Cámara Intel Realsense L515	59
Figura B.2. Comparativa de imágenes de color, infrarrojos y profundidad con Realsense L515	60
Figura B.3. Alineamiento de imágenes de color y profundidad	61

Índice de tablas

Table 1. Comparative of the mean error, the maximum error, precision, the typical deviation and speed of each method for calculating the orientation of pieces .	xxiv
Tabla 3.1. Cronograma del proyecto	18
Tabla 5.1. Ordenador empleado para el entrenamiento	34
Tabla 5.2. Modelos de YOLOv5	36
Tabla 5.3. Opciones de entrenamiento de YOLO	37
Tabla 5.4. Opciones de entrenamiento de TINY YOLO	43
Tabla 5.5. Opciones de entrenamiento de Regresor	48

Siglas

API Application Programming Interface

DLR-RM German Aerospace Center (DLR) - Institute of Robotics and Mechatronics (RM)

Faster R-CNN Faster Region-proposal Convolutional Neuronal Networks

PHT Probabilistic Hough Transform

PPHT Progressive Probabilistic Hough Transform

R-CNN Region-proposal Convolutional Neuronal Networks

R-RANSAC Recursive RANdom SAmple Consensus

RANSAC RANdom SAmple Consensus

RHT Randomized Hough Transform

YOLO You Only Look Once

Resumen

OPTIMIZACIÓN DEL SISTEMA DE VISIÓN ARTIFICIAL DE UN ROBOT INDUSTRIAL PARA UNA APLICACIÓN DE PICK AND PLACE

Autor: Ortiz de Zúñiga Mingot, Ignacio.

Directores: Boal Martín-Larrauri, Jaime y Rodríguez Mondéjar, José Antonio.

Entidad colaboradora: ICAI – Universidad Pontificia Comillas

Resumen del proyecto

En la última década se ha producido revolución industrial conocida como industria 4.0 gracias a los numerosos avances en el sector de la robótica. Este cambio se ha visto producido gracias a que con el desarrollo de los robots se ha conseguido dotar a estos de capacidades para completar tareas previamente imposibles para el ser humano y con un rendimiento y rapidez superior. Es por ello que se decidió implantar dichos sistemas en un robot industrial de Comillas ICAI con el fin de que este sea capaz de recolectar piezas de LEGO dispuestas de forma aleatoria sobre una mesa de trabajo. En este proyecto se parte de este sistema basado en la segmentación clásica y se mejora y actualiza con el desarrollo de sistemas basados en redes neuronales y aprendizaje profundo. Se han desarrollado múltiples redes de diferentes tamaños basadas en R-CNN, Faster R-CNN y YOLO y se comparan entre si y frente al sistema de segmentación clásico.

Palabras clave: Visión artificial, Redes neuronales, AlexNet, VGG-16, R-CNN, Faster R-CNN, YOLO, Robótica, LEGO

1. Introducción

Durante los últimos dos años se ha desarrollado un sistema que dote de visión artificial y autonomía a los robots de Comillas ICAI. La tarea escogida para llevar a cabo con estos robots es la recolección de piezas de LEGO dispuesta de forma aleatoria en una zona de trabajo. Este proyecto surge como la evolución de estos sistemas con el fin de mejorar el subsistema de visión artificial mejorando así las capacidades del robot. Para ello primero se ha mejorado el sistema presente con el desarrollo de nuevos análisis más detallados y precisos. Este sistema se basa en el filtrado con máscaras de color, la detección de bordes y la transformada de Hough. Además, se han desarrollado nuevos sistemas para el cálculo de la profundidad y el cálculo de la orientación de la pieza. Sin embargo, a pesar de estas mejoras se ha observado que este sistema no puede competir con los sistemas más modernos ya implantados en la industria. Es por ello que se han desarrollado nuevos sistemas basados en redes neuronales convolucionales. Tal y como su nombre indica estos se basan en el principio de la convolución y el aprendizaje profundo. Con la ayuda de una base de datos son capaces de extraer las características de un

objeto y aprender a identificarlo. Es por ello que se a optado por dotar de esta tecnología a los sistemas previos para mejorarlos y así perfeccionar los.

2. Metodología

El sistema de visión artificial implantado en el brazo robótico esta constituido por tres elementos que deben de cooperar entre si y comunicarse a través de conexiones USB y TCP/IP. La captura de imágenes se realiza con una cámara Intel Realsense D435 que cuenta con sensores RGB y de profundidad y ha sido conectada a través de una conexión USB. El procesado de las imágenes RGB y de profundidad es llevado a cabo por un ordenador con la ayuda de MATLAB. Y por último, se mandarán las instrucciones contenido la posición, altura y orientación de cada pieza a través de una conexión TCP/IP al brazo robótico IRB 120. Esta estructura se puede ver en la ??

El primer paso en el desarrollo de este proyecto ha sido la mejora el sistema actual basado en filtros con máscaras de color, la detección de bordes y la transformada de Hough. Se han revisado los filtros de color reduciendo así los falsos positivos y se ha perfeccionado el análisis por detección de bordes de las piezas. Se ha rediseñado y calibrado la cámara para permitir una captura más rápida y eliminar las distorsiones presentes en la imagen de profundidad. También se ha rediseñado el sistema de cálculo de la orientación con el desarrollo de un análisis más exhaustivo de todas las caras de las piezas de LEGO mediante la transformada de Hough.

Una vez este sistema ha sido mejorado y perfeccionado, se ha desarrollado desde cero nuevos detectores de objetos basados en redes neuronales convolucionales. Con el fin de poder realizar un análisis riguroso de estas redes y poder seleccionar un buen método o combinación de métodos a emplear, se han desarrollado múltiples redes de diferentes tamaños y basadas en diferentes tecnologías. Las redes desarrolladas son del tipo: tipo R-CNN, Faster R-CNN y YOLO. Y para cada tipo de tecnología se han creado dos redes basadas en clasificadores reentrenados basados en AlexNet y VGG-16.

Para poder llevar esto a cabo es necesario primero rediseñar y reentrenar los clasificadores antes de poder crear los detectores de objetos. El motivo por el que se ha llevado a cabo este paso es que es común y altamente recomendable a la hora de desarrollar un detector de objetos, que este sea basado en un clasificador. Este proceso es conocido como aprendizaje por transferencia y ayuda a reducir el tiempo de entrenamiento y el número de datos necesarios a la vez que mejora los resultados. Es por ello que en este proyecto se ha decidido emplear el aprendizaje por transferencia partiendo de dos clasificadores diferentes. Los clasificadores empleados son AlexNet y VGG-16, sin embargo, estos clasificadores no han sido entrenados para la identificación de piezas de LEGO. Es por ello que primero serán reentrenados para la identificación de dichas piezas. Y a continuación, se partirá de estos dos nuevos clasificadores bautizados como LEGONet y LEGO16 para el desarrollo de los detectores de objetos mencionados previamente. Es decir, dos redes del tipo R-CNN, dos del tipo Faster R-CNN y dos del tipo YOLO.

R-CNN surgió como evolución de los primeros detectores de objetos basados en ventanas flotantes. Esta red primero analiza la imagen y determina aproximadamente 2000 regiones de interés para que estas sean analizadas de forma independiente por un clasificador. Con este sistema se redujo los tiempos de ejecución frente a los previos sistemas, pero sigue siendo un sistema lento ya que debe de analizar cada sección por separado. A cambio de un mal rendimiento, se caracteriza por ser uno de los sistemas más precisos y capaces.

Durante el desarrollo de este proyecto se han desarrollado dos redes tipo R-CNN basadas en LEGONet y LEGO16. Estas redes han sido entrenadas con una base de datos desarrollada por nosotros y se han realizado múltiples pruebas con el fin de determinar las mejores opciones de entrenamiento.

Faster R-CNN surge tres años después de R-CNN como una evolución del mismo. Al igual que R-CNN se basa en la propuesta de regiones pero este proceso es llevado a cabo por una red RPN, una red neuronal convolucional cuya salida son las regiones de interés. Esta red se basa en la información extraída por las primeras capas del clasificador. El análisis de las regiones de interés también se lleva a cabo con el resto del clasificador y de forma independiente. En este sistema no se tiene que analizar la imagen original tantas veces como regiones propuestas sino que solo se analizan las regiones de interés extraídas de las capas de convolución iniciales. Y es por ello que se consigue reducir los tiempos de ejecución. Durante el desarrollo de este proyecto se han desarrollado dos redes tipo Faster R-CNN basadas en LEGONet y LEGO16. Estas redes han sido entrenadas con una base de datos desarrollada por nosotros y se han realizado múltiples pruebas con el fin de determinar las mejores opciones de entrenamiento.

YOLO es uno de los sistemas más modernos y actuales empleados para la detección de objetos. Se caracteriza por su rapidez de entrenamiento y de ejecución. Esto se debe a que tal y como su nombre indica, *You Only Look Once*, es un tipo de red en la que la imagen solo se analiza una vez. Y mediante el uso de un sistema de predicción de *bounding boxes* determina los objetos presentes en la imagen. Este sistema se caracteriza porque a diferencia de los sistemas mencionados anteriormente, es capaz de ver toda la imagen a la vez y por ello puede establecer relaciones entre *bounding boxes*. Durante el desarrollo de este proyecto se han desarrollado dos redes tipo YOLO basadas en LEGONet y LEGO16. Estas redes han sido entrenadas con una base de datos desarrollada por nosotros y se han realizado múltiples pruebas con el fin de determinar las mejores opciones de entrenamiento.

Una vez identificadas todas las piezas mediante el uso de los detectores de objetos, es necesario calcular su orientación y altura. Este proceso ya ha sido perfeccionado al principio del proyecto. La información obtenida con todos estos análisis es mandada al brazo robótico y las piezas detectadas son recolectadas.

Como desarrollo adicional del proyecto, se ha decidido crear dos modelos de regresión basados en LEGONet y LEGO16 con el fin de estimar la orientación de una pieza de LEGO. Para el desarrollo de estas redes se ha preparado una nueva base de datos desarrollada por nosotros y constituida por miles de imágenes de piezas de LEGO rotadas diferentes ángulos. Para el entrenamiento también se han llevado a cabo múltiples pruebas con el objetivo de encontrar las mejores opciones de entrenamiento.

3. Resultados

Detectores de objetos

Tras desarrollar y entrenar todos los sistemas para la detección de objetos, se ha llevado a cabo una comparativa enfrentando los entre sí y frente a la versión mejorada del sistema clásico. Para la evaluación se han analizado un total de 380 piezas de LEGO de diferentes colores y en diferentes escenas. Con la evaluación se han llevado a cabo múltiples mediadas para evaluar el comportamiento de cada sistema.

Precisión, Exhaustividad, Tasa de fallos y FPPI

Como se ha mencionado anteriormente, se han medido diferentes parámetros para evaluar el comportamiento de cada sistema. La precisión indica el nivel de similitud entre la detección de cada método y la verdadera región en la que se encuentra la pieza. La exhaustividad determina la relación entre los verdaderos positivos y los falsos negativos. La tasa de fallos determina la probabilidad de no detectar una de las piezas de LEGO. Y FPPI representa el número de falsos positivos por imagen detectadas. Tras evaluar todos los sistemas desarrollados, se puede observar la clara superioridad de las redes neuronales. Los nuevos sistemas son capaces de detectar piezas de LEGO bajo diferentes escenarios y condiciones lumínicas. Además, han reducido notablemente el número de falsos positivos y la tasa de fallo. Y entre ellos destaca R-CNN por su precisión y YOLO. Este último destaca más si se tiene en cuenta su menor tiempo de ejecución. A continuación, se muestran los resultado al evaluar las piezas LEGO de color rojo en la ??.

Velocidad

Otro factor importante de considerar a la hora de implantar un sistema es el tiempo de ejecución. Por ello durante la evaluación también se han medido los tiempos de ejecución de cada sistema. Con el fin de reflejar las diferencias en ejecución de cada sistema, las medidas han sido transformadas de décimas de segundo a imágenes por segundo. Como se puede observar, las nuevas redes neuronales no solo destacan por dar mejores resultados sino que también son más rápidas. A excepción de R-CNN cuyo rendimiento es similar al sistema clásico. Las redes que más destacan son las basadas en YOLO ya que su rendimiento es claramente superior rompiendo siempre la barrera de las 30 imágenes por segundo e incluso rompiendo la de los 60 en el caso de YOLO basado en LEGONet.

Análisis de la orientación

También se ha llevado a cabo una evaluación de los sistemas desarrollados para el cálculo de la orientación. Es decir, el sistema clásico basado en detección de bordes y la transformada de Hough y los dos modelos de regresión basados en LEGONet y LEGO16. En el caso de estos sistemas, en la evaluación se han medido los errores medios, el error máximo, la desviación típica y la velocidad de cada sistema. De nuevo, la velocidad se ha medido en piezas de LEGO por segundo ya que refleja mejor la capacidad de cada sistema.

4. Conclusiones

El objetivo final de este proyecto es el perfeccionamiento del sistema previamente desarrollado por Ana Berjón Valles [TFGAna] mejorando la capacidad de reconocimiento. Todos los objetivos planteados durante el desarrollo del proyecto se han podido llevar a cabo y se ha mejorado en su totalidad el sistema. A continuación, se va desarrollar más en detalle los objetivos cumplidos.

Uno de los principales objetivos consiste en subsanar los problemas del sistema antiguo ante cambios de iluminación. Como el sistema clásico se basa puramente en filtros de color y detección de borde, se ve notablemente afectado por cambios en la iluminación. Falta comprobar el funcionamiento de los sistemas desarrollados durante este proyecto en el laboratorio, pero con los resultados obtenidos durante la evaluación de estos se considera que este problema ya ha sido solventado y mitigado. Dados los resultados se recomienda para futuros proyectos la

implantación del sistema basado en YOLO con LEGO16 y el modelo de regresión basado en LEGONet. También es recomendable plantearse el uso de un sistema combinado con varias redes neuronales.

Otra gran limitación del sistema anterior era los fallos debidos al análisis de profundidad. Debido al tipo de tecnología empleada para detectar la profundidad, esta se ve bastante afectada por los reflejos de las luces del laboratorio sobre las piezas de LEGO. Este problema ha podido ser solventado mediante la modificación del sistema de análisis de profundidad. Se ha introducido la posibilidad de calibrar rápidamente la cámara para poder tomar referencias de alturas. Además, se ha repartido el área de trabajo en secciones y se ha calculado la profundidad de dichas secciones con la mediana y teniendo en cuenta el entorno a la sección. De esta forma se mitiga el efecto de los reflejos.

Generalización de los algoritmos empleados. Durante el desarrollo del proyecto se ha tenido en cuenta este objetivo y es por ello que todo el procesado de la imagen no depende de la cámara, del brazo ni del laboratorio. La cámara ha sido tratada como una clase y el proceso de segmentación ya no depende tanto del calibrado de color de la cámara empleada. Además, con el procesado de la imagen de profundidad se ha creado un sistema fácil de calibrar y reutilizar para diferentes circunstancias. Esto implica que este sistema puede ser reutilizado para el reconocimiento de diferentes piezas, con diferentes cámaras y diferentes puntos de trabajo. Todo proceso que involucre al brazo robótico es llevado a cabo en forma de funciones, por ello solo modificando estas se puede implantar el sistema en cualquier otro brazo robótico del laboratorio.

Analizando el proyecto de forma global, este nuevo sistema supone un gran avance frente al anterior sistema y una vez sea implantado supondrá una gran mejora en la tecnología y capacidades de los robots industriales de Comillas ICAI.

Abstract

OPTIMIZATION OF AN ARTIFICIAL VISION SYSTEM OF AN INDUSTRIAL ROBOT FOR A PICK AND PLACE APPLICATION

Author: Ortiz de Zúñiga Mingot, Ignacio.

Supervisors: Boal Martín-Larrauri, Jaime y Rodríguez Mondéjar, José Antonio.

Collaborating Entity: ICAI – Universidad Pontificia Comillas

Abstract

In the last decade an industrial revolution known as industry 4.0 has started. This change has been capable thanks to the development and growth of industrial robots. These robots are now capable of completing tasks previously impossible to humans and at a faster and more efficient rate. Because of these advances Comillas ICAI has decided to implant these new systems to its industrial robots. These robots have to be capable of identifying and recollecting LEGO pieces randomly set at a workbench. To do so this project takes upon the previous work of Ana Berjón Valles to improve it. It will also create new systems based on convolutional neuronal networks and R-CNN, Faster R-CNN and YOLO. **Keywords:** Artificial vision, Neuronal networks, AlexNet, VGG-16, R-CNN, Faster R-CNN, YOLO, Robotics, LEGO

1. Introduction

In the past two years Comillas ICAI has been developing a system capable of endowing artificial vision and autonomy to its robots. These robots have to be capable of identifying and recollecting LEGO pieces randomly set at a workbench. This project takes upon this previous work done by Ana Berjón Valles to improve it. To do so, we have first upgraded the current segmentation system based on colour masking, edge detection and the Hough transform. Also the systems designed to analyse the depth and rotation of the pieces have been redesigned and upgraded. However, despite these upgrades our system is still incapable of competing with the modern systems already well implanted in the industry. These new systems are based on convolutional neuronal networks. As its name indicates these networks are based on the principles of convolution and deep learning. Because of it we have decided to upgrade the robots with these new technologies and improve its capabilities.

2. Methodology

The current artificial vision system implanted on the robot is composed by three elements that must cooperate and communicate with each other. The communications are done via USB

Abstract

and TCP/IP. The capture of images is done by the camera Intel Realsense D435. This camera contains RGB and Depth sensors and it is connected to the system with and USB connexion. The processing of both RGB and Depth images is done by a computer with the help of MATLAB. And lastly, the instructions containing the position, height and orientation of the pieces is send to the IRB120 robot with a TCP/IP connection. The architecture of the system can be seen on ??.

The first step on the development of this project is to upgrade the current system based on colour masking, edge detection and the Hough transform. The colour filters have been revised and upgraded to reduce the number of false positives and the analysis based on edge detection has been perfected. The communication with the camera has been redone to allow for a faster capture and to eliminate the distortion of the depth image. The analysis and calculation of the orientation of each pieces has been redone and upgraded with a new and more sophisticated analysis method also based on the Hough transform.

Once the current system has been upgraded and perfected, new object detectors based on convolutional neuronal networks have been developed from zero. In order to create a rigorous comparison between different neuronal networks multiple networks have been created. To evaluate different technologies we have created networks of the type R-CNN, Faster R-CNN and YOLO. Also to be able to evaluate the impact of the size of the network ours are based on two different classifiers. The selected classifiers are a retrained version of AlexNet and VGG-16. The retraining of these classifiers has also been done by us.

The use of classifiers as the base for an object detector is because it reduces the training time while also improving the results of the network. This process is known as transfer learning and is common step used on the creation of object detectors. That's the reason why we have decided to use AlexNet and VGG-16. We have decided to use these networks so that we could evaluate the impact of different networks size and to demonstrate the evolution of neuronal networks. AlexNet was created in 2012 while VGG-16 is from 2014 and currently is still a very modern and capable network. But because these networks were never created to identify LEGO pieces we had to first retrain them so that they could learn the characteristics of LEGOs. These new classifiers have been renamed LEGONet and LEGO16 respectively and will be used as the base for our object detectors. Two networks type R-CNN, two type Faster R-CNN and two type YOLO.

R-CNN emerged as the evolution of first object detectors based on Sliding windows. This network firstly analyses the image and proposes approximately 2000 regions of interest to be analyse. Then a classifier analyses each of the regions independently to determine the existence of an object inside the regions. With this system the execution time was improved compared to the Sliding windows systems but it still is a slow process compared to newer systems. In exchange for its poor performance this system is characterized for having good precision. During the development of this project two networks based on LEGONet and LEGO16 have been created. These networks type R-CNN have been trained with a dataset created by us and multiple testing has been conducted to determine the best training options.

Faster R-CNN was created three years after R-CNN and emerges as an evolution of this system. As R-CNN this new system is also based on regions proposal but this task is done by a RPN network. RPN is a convolutional neuronal network whose output are the proposed regions to analyse. The RPN network is located after the first convolutional networks and takes information from them and then the rest of the network analyses the regions proposed by the RPN. In this

system the original image is not analyse as many times as regions proposed instead is the output from the convolutional layers that is analyse multiple times. Thanks to this the system has a better performance than R-CNN and it manages to reduce training and executing times. During the development of this project two networks type Faster R-CNN base on LEGONet and LEGO16 have been created. This networks have been trained with a dataset created by us and multiple testing has been conducted to determine the best training options.

YOLO is one of the newest and modern systems for object detection. It is characterize for it's performance and speed both in training and execution. This is because as it's name indicates, You Only Look Once, it only analyses the image one time. And system designed to predict the bounding boxes predicts them. Compare to the previously mentioned systems, YOLO is characterize for it's ability to see the whole image. This allows it to create connexions between different elements of the image and it helps it to reduce the number of false positives. During the development of this project two networks type YOLO base on LEGONet and LEGO16 have been created. This networks have been trained with a dataset created by us and multiple testing has been conducted to determine the best training options.

Once the pieces have been identified with the use of the object detector it's time to calculate their orientation and height. This process has already been perfected at the beginning of the project. The information extracted from all this systems is then process and send to the robot to relocate the pieces.

As an additional development of the project we have decided to create to regression models based on LEGONet and LEGO16 to estimate the orientation of each LEGO piece. For the development of this networks a new dataset has been created by us composed by thousand of images of LEGOs rotated with different angles. For the training of the networks multiple testing has been done to determine the best training options.

3. Results

Object detectors

A comparative of all the object detectors has been done. This includes the classic system based of colour masking, edge detection and th Hough transform. The evaluation has been done by analyzing a total of 380 LEGO pieces of different colours and in different scenes. With these evaluation multiple measures have been taken to evaluate the behaviour of each object detector.

Precision, Recall, Failure rate and FPPI

As mentioned before, multiple measures have been taken from different parameters to evaluate the behaviour of each system. The precision indicates the level of similarity between the region proposed by the detector and true region of the piece. The recall determines the relationship between the number of true positives and the number of false negatives. The failure rate indicates the probability of not detecting a piece on an image. And the FPPI represents the number of False Positives Per Image detected. Analyzing the results of all the object detectors on the evaluation we can observe the clear superiority of neuronal networks. The new detectors are capable of detecting the pieces under different illumination and scenes. While also reducing the number of false positives and the failure rate.

Speed

Another important factor to consider when analyzing all the object detectors is the execution time. Because of it we have measure the time taken by each detector to analyse all the images from the evaluation. This measures on tenths of a second have been transform to images per second to help understand the capabilities of each detector. We can see that all the new systems are faster than the previous classical system. With the exception of R-CNN whose performance is similar to the classical system. The networks taht stand out the most are the ones based on YOLO. Their performance is clearly superior always achieving to break the 30 fps barrier and in the case of YOLO based on LEGONet also the 60 fps barrier.

Analysis of the orientation

An evaluation of the methods designed to calculate the orientation of each piece has also been done. We have compare all three systems. The classical system based on edge detection and the Hough transform and the two networks based on LEGONet and LEGO16. The evaluation has measured different parameters of these systems. The mean error, the maximum error, the typical deviation and the speed of each method. In this evaluation the execution time of each system has also been transform into pieces per second to help understand the capabilities of each method.

	Hough transform	LEGONet	LEGO16
Mean error	0.98°	0.33°	0.71°
Maximum error	26°	1.3°	16°
Precision	83.67%	97.67%	79.33%
Typical deviation	1.82°	0.43°	1.23°
Pieces per second	89.7	74.6	178.5

Table 1. Comparative of the mean error, the maximum error, precision, the typical deviation and speed of each method for calculating the orientation of pieces

4. Conclusions

The objective of this project is the perfection of a previous system developed by Ana Berjón Valles [TFGAna]. All the objectives set during the development of this project have been successfully completed. Below we will develop in more detail all these objectives.

One of the principal task was to subsidize the problems of the old system with the changes of illumination. The classic system is based on colour filtering and because of it a change on the illumination could easily affect its capacity to detect a piece. The new systems based on neuronal networks are more robust and their performance does not variate as much with changes in illumination. More testing needs to be done to evaluate this results on the laboratory but considering the results of the evaluation we consider that this problem has been solved and mitigated. Considering the results we recommend to future projects to implement a system with YOLO based on LEGO16 and the regression model based on LEGONet. Or a combination of multiple neuronal networks.

Another limitation of the previous system was the failures due to a bad analysis of the depth image. Due to the type of technology used to obtain the depth image, the camera is really sensible to reflexes. This problem has been solved with the modification of the analysis of depth. The work bench is divided of sections and the height of each section is estimated with a

median. Thanks to the inclusion of these steps and the capability to recalibrate de system we have mitigated the effect of the reflexes

Generalization of the algorithms. During the development of this project we have consider the possibility to implement these system for different applications or robots. Because of it the process of both RGB and Depth images doesn't depend of the camera, the position or the robot. The analyse of the depth image can be easily calibrated to work under different circumstances and positions with the help of the calibration function created. The images process can also be retrain to detect different pieces. And all the processed related with the robot has been done separate from the image processing so it can easily modify to be use on another robot.

Globally analyzing the project we can conclude that the new systems suppose a bid upgrade compare to the previous systems. Once it is implanted it will suppose a big upgrade on the technologies and capabilities of the industrial robot of Comillas ICAI.

1

Introducción

Es siempre sabio mirar adelante, pero difícil mirar más allá de lo que puedes.

Winston Churchill (1874–1965)

Este primer capítulo introduce a la visión artificial y demuestra el gran interés otorgado por la comunidad científica en este área. También se muestra la motivación de este proyecto, así como sus objetivos y se desarrolla la estructura del mismo.

Los orígenes de la visión artificial surgieron en 1960 como sistemas para el reconocimiento de patrones y centrados en su posible implantación en el sector industrial debido a la gran cantidad de tareas repetitivas fácilmente automatizables [AT13]. A pesar de la carencia de los recursos en su momento, pero debido al gran interés del mercado estos siguieron siendo desarrollados y poco a poco implantados. Una de las primeras empresas en implantar estos sistemas fue Hitachi Labs en 1964 en Japón [AT13].

Desgraciadamente estos primeros desarrollos carecían de precisión y tenían una tasa de acierto del 95 % (no se consideró suficiente para su implementación en una línea de producción). Pero esto se vería resuelto en los años venideros de forma que en la década de los 70 estos sistemas pasaron a formar parte de un gran porcentaje del sector industrial. Un claro ejemplo fue la automatización de la producción de transistores con semiconductores en 1974 por Hitachi ya que, debido a su complejidad, esta tarea no podía ser llevada a cabo por humanos de forma segura [Lab].

En la actualidad se pueden distinguir dos tipos de visión artificial en función de sus capacidades de adaptación. En primer lugar, se encuentran los sistemas desarrollados a partir 1960 que se caracterizan por ser programas para cumplir un único objetivo bajo unas circunstancias dadas. Estos son los menos potentes ya que no se adaptan y no saben reaccionar ante un cambio, pero son los más fáciles de desarrollar. Estos sistemas se basan en características diferenciadoras del objeto de trabajo como puede ser la forma, color, un patrón... Además, en espacios controlados pueden llegar a dar mejores resultados que sistemas más complejos y avanzados [Cas+13]. Pero es debido a su falta de flexibilidad y a las limitaciones de características diferenciadoras de las piezas de trabajo que están siendo sustituidos.

Por otro lado, con el desarrollo de las redes neuronales y de inteligencias artificiales, se están desarrollando sistemas capaces de aprender y adaptarse a la situación a base de prueba y error. Se trata de sistemas muy modernos todavía en desarrollo que prometen traer avances como la conducción autónoma, detección de emociones en humanos, etc. Estos sistemas necesitan de grandes cantidades de bases de datos de las que aprender. Y sobre las que generar sus propias conclusiones y reglas de conducta. Su nivel de adaptabilidad y flexibilidad se ve definido por las capacidades de aprendizaje del sistema (neuronas y estructura de estas) así como de la riqueza de la base de datos. Pero gracias a los últimos avances, han conseguido superar la barrera de tecnología en desarrollo y se han empezado a implantar en sistemas reales.

El fin de este proyecto es la generación de un nuevo sistema de visión artificial para el reconocimiento de piezas de uso industrial. Este sistema se implantará dentro de una cadena de suministro del Grupo Antolín® que a su vez alimentará a una línea de montaje y ensamblaje. El sistema deberá de poder identificar múltiples piezas y determinar el punto de agarre óptimo de cada pieza. Este se ve definido por sus coordenadas así como el vector normal a la superficie. De esta forma un brazo robótico con un sistema de agarre por aspiración, ventosa o *soft-robotics* (varía en función de la pieza a coger) podrá recolectarlas. La multitud de herramientas de agarre así como de la necesidad de un sistema de determinación de puntos de agarre se debe a la gran variedad de piezas existentes con formas y tamaños de gran variedad (1-30 cm). El proceso completo se puede dividir en varias etapas:

1. Generación del pedido: en función de la demanda y de los requisitos del cliente se debe de generar una lista con todos los componentes necesarios para cada pedido.
2. Estructuración del pedido: Las piezas necesarias se encuentran distribuidas en diferentes secciones y es por ello que se debe de crear un que determine el orden de recolección de piezas óptimo que reduzca el tiempo recolección.
3. Recolección de piezas: Se trata de un proceso iterativo que se debe de realizar para cada una de las piezas que constituyen el pedido.
 - 3.1. Desplazamiento hasta la pieza. Dependiendo de la configuración del robot este sistema variará, pero el objetivo siempre será el mismo. Trasladar el robot hasta la región donde se encuentra la pieza a recolectar con el fin de poder capturarla y situarla dentro de la región de alcance.
 - 3.2. Detección de la pieza: aplicando algoritmos de detección se identificará la pieza que se desea recolectar. Dentro de una misma zona se detectarán numerosas instancias de una misma pieza. Se debe de escoger la pieza/piezas mejor ubicadas y con más probabilidad de éxito.
 - 3.3. Punto de agarre: tras detectar la pieza se debe de determinar como se debe de agarrar la pieza. Para ello se debe de determinar el punto de agarre óptimo y el vector normal a dicho.
 - 3.4. Recolección de la pieza: se transfiere la información necesaria al robot para que este pueda recolectar la pieza a través del punto de agarre definido. El sistema de agarre a emplear dependerá del tipo de pieza.
 - 3.5. Deposición y control de calidad: por último se debe de depositar la pieza dentro de la cesta que constituye el pedido. Y mediante el sistema de visión artificial se debe de comprobar que la pieza deseada ha sido correctamente depositada en la cesta.

4. Control de calidad y trazabilidad: antes de dar por finalizado en pedido se analiza por última vez para comprobar que todas las piezas necesarias se encuentran dentro de la cesta. Se registra en el sistema el pedido y toda la información necesaria para futura trazabilidad.
5. Traslado del pedido: una vez se da por finalizado el pedido este se debe de trasladar hasta la zona de ensamblaje para comenzar el proceso de montado.

En este proyecto nos centraremos únicamente en el desarrollo de una herramienta/sistema de visión artificial que permitirá la detección de la pieza así como la definición del punto de agarre y el vector normal a este. Este sistema se implantará para llevar a cabo la etapa 3.2 así como 3.3. El sistema también puede ser usado en las etapas 3.5 y 4 para llevar acabo las tareas de control de calidad y trazabilidad.

Una vez desarrollado el contexto de este proyecto se puede introducir la solución desarrollada. Debido a la complejidad de la salida deseada (detección más puntos de agarre) ha sido necesario desarrollar un sistema complejo compuesto por tres redes neuronales:

- *YOLO*: se trata de la primera red neuronal empleada y desarrollada. Se encargará de detectar las piezas dentro de la zona de trabajo y se empleará como base para decidir que pieza se debe de recolectar. Es la única capa común a todas las piezas.
- *Tiny YOLO*: una red neuronal menor y menos potente pero con una mayor especialización. Esta red se encargará de determinar regiones con posibles puntos de agarre dentro de las piezas (previamente identificadas por *YOLO*). Esta segunda capa es específica para cada pieza y solo se aplicará a aquellas piezas de elevado volumen.
- Regresor: se trata de la última capa del sistema de visión artificial. Se basa en la salida del *Tiny YOLO* y determina para cada una de las posibles regiones de interés el punto de agarre óptimo y su vector normal. Esta última capa es específica para cada pieza y solo se aplicará a aquellas piezas de elevado volumen.

Como se ha mencionado anteriormente, las capacidades de una red neuronal se ven limitadas tanto por la estructura de la red como por la riqueza de la base de datos. Y para esta aplicación concreta la obtención de dicha base de datos presenta un gran desafío debido a la gran cantidad de información que se requiere de cada imagen (identificación de las piezas, posibles puntos de agarre de cada pieza y el vector normal a dichos puntos). Es por lo que se ha optado por desarrollar a su vez una base de datos sintética que permita generar todas las imágenes e información necesarias. De esta forma se podrá automatizar el proceso de aprendizaje de la red para la introducción de nuevas piezas.

2

Estado de la cuestión

En este capítulo se desarrollará la evolución de la visión artificial hasta llegar a la situación actual. También se introducirá la componente robótica del proyecto así como otras aplicaciones *Pick and Place*.

Como ya se ha indicado en la Sección 3.3, este proyecto está claramente definido por dos elementos que deben de cooperar entre sí para poder cumplir el objetivo de identificar y recolectar las piezas. Se necesita de un buen sistema de reconocimiento de objetos a través de imágenes, así como de un brazo robótico para poder interactuar con el entorno. Por ello, se van a analizar cada uno de estos elementos por separado y después se desarrollará la interacción entre estos mismos. Primero se analizará la situación actual del procesado de imágenes, después se analizará los brazos robóticos y por último, se hablará de los sistemas *Pick and Place*.

2.1. Procesado de la imagen

Como ya se ha explicado en la introducción, los primeros sistemas de visión artificial surgieron en la década de los sesenta y desde entonces han avanzado notablemente hasta el nivel de convertirse en uno de los pilares fundamentales de la industria moderna. Actualmente los nuevos sistemas se han desviado drásticamente de los primeros intentos y emplean complejos sistemas neuronales y *machine learning* para obtener mejores resultados y mejores respuestas. En este proyecto se va a perfeccionar el actual sistema basado en filtros de color y detección de borde y también se va a desarrollar una alternativa basada en redes neuronales.

2.1.1. Filtros, detección de borde y detección de formas

La identificación de objetos no es un proceso directo, sino que está compuesto por varias etapas y dentro de estas existen multitud de herramientas. En esta sección se va a intentar segmentar este proceso y explicar de forma independiente cada proceso, así como las diferentes herramientas que se pueden usar.

Filtros

Los filtros son diseñados con el objetivo de modificar la imagen para facilitar la extracción de características. Estos se pueden dividir en dos grandes categorías:

- **Filtros lineales:** Consisten en aplicar a cada píxel un filtro con sus píxeles vecinos. Dependiendo de los pesos que se den a los píxeles vecinos se pueden obtener diferentes efectos [GSA16]. Se muestra la expresión matemática 2.1 que rige este proceso.

$$g(i, j) = \sum_{k,l} f(i + k, j + l) * h(k, l) \quad (2.1)$$

Se trata de un proceso por convolución en el que dependiendo de la matriz a usar y el peso de sus componentes se pueden obtener diferentes resultados [GSA16]. Se puede ver su aplicación en la figura 2.1.

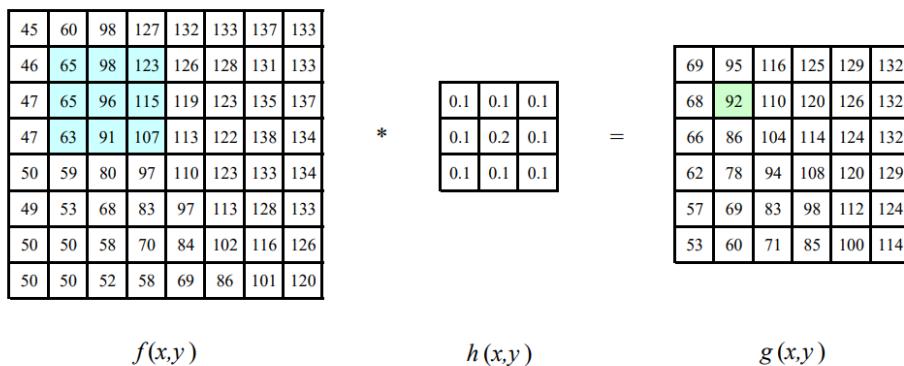


Figura 2.1. Ejemplo de un proceso de convolución aplicado a una matriz (Fuente: Computer Vision: Algorithms and Applications, Richard Szeliski figure 3.10 [Sze22])

Un claro ejemplo de este tipo de filtros es el suavizado. Permite suavizar la imagen y hacerla más homogénea al combinar cada pixel con sus vecinos. Este filtro permite reducir los valores atípicos reduciendo así el efecto del ruido. Pero también se puede dar el caso contrario en el que nos interese resaltar los pequeños detalles. Este filtro es conocido como *Sharpening* y se basa en un filtro lineal de suavizado. A la imagen original se le resta las diferencias producidas por el suavizado y de esta forma se resaltan más los pequeños detalles de la imagen. Este proceso es conocido como *unsharp masking* [Sze22].

- **Filtros no lineales:** A pesar de que muchos de los filtros usados se pueden obtener de forma lineal, en muchas ocasiones se consigue un mejor rendimiento aplicando filtros no lineales. Tal y como su nombre indica, esta categoría engloba todos aquellos filtros que no se puedan expresar como una suma de diferentes argumentos.

El beneficio de los filtros no lineales se puede ver al aplicar transformaciones que usen por ejemplo la mediana. El cálculo de la mediana puede suponer una alta carga computacional frente al resto de filtros. Por eso se han desarrollado alternativas para mejorar su cálculo. Tales como *α -trimmed mean* [LR90].

Un buen ejemplo de filtro no lineal que emplee la mediana es la transformación a escala de grises. Se suele usar la mediana ya que es más robusta ante variaciones atípicas. En el reconocimiento de objetos es muy recomendable usar la escala de grises ya que se

consigue una mejor detección de borde [AMS18] y reduce la carga computacional al trabajar con una sola matriz (*Black*) en lugar de tres (RGB). Existen múltiples algoritmos para transformar a escala de grises, aunque uno de los más empleados y que mejores resultados da para el reconocimiento de borde es *Lightness color-to-greyscale conversión algorithm* [AMS18]. Sin embargo, gracias a los avances en redes neuronales y a las mejoras computacionales los sistemas actuales no se ven tan limitados y por ello son capaces de trabajar con imágenes de alta resolución y los tres canales RGB.

Detección de bordes

Los bordes son el pilar fundamental de la visión artificial. Representan la separación entre diferentes objetos y definen sus formas. En una imagen, un borde se define como un salto de la intensidad de los pixeles.

En la práctica, estos cambios no son tan bruscos, sino que se producen gradualmente a lo largo de varios pixeles. Por ello no son tan fáciles de detectar y suelen aparecer bordes residuales producidos por ruido y errores en la imagen. En la detección de borde existen dos claras vertientes, aquellos métodos que se basan en el gradiente y los que emplean el laplaciano.

- **Operadores basados en gradientes:** El gradiente de una función es la colección de todas las derivadas parciales en forma de vector. La dirección del vector es la de máximo crecimiento y su módulo es el valor de la pendiente [Aca]. Al aplicar este concepto a imágenes, podemos obtener para cada píxel un vector que indique la dirección en la que aumenta la intensidad y su módulo. Analizando los gradientes obtenidos con la ayuda de máscaras, podemos encontrar los bordes ya que se producen por un cambio brusco de intensidad.

Existen varios algoritmos basados en el gradiente, aunque algunos de los más conocidos son el de Robert, Prewitt y el de Sobel. La diferencia entre estos son las máscaras a aplicar. Robert es un operador diferencial discreto con una matriz de 2x2, mientras que Prewitt y Sobel emplean dos matrices de 3x3. Se puede ver una comparativa de estos en la Figura 2.2.

El principal inconveniente de estos operadores es su forma de representar los bordes ya que están definidas por una franja de pixeles y además se ven bastante afectados por el ruido. Por ello existen algoritmos basados en la segunda derivada (laplaciano) para delimitar más la región de borde y obtener una mayor precisión.

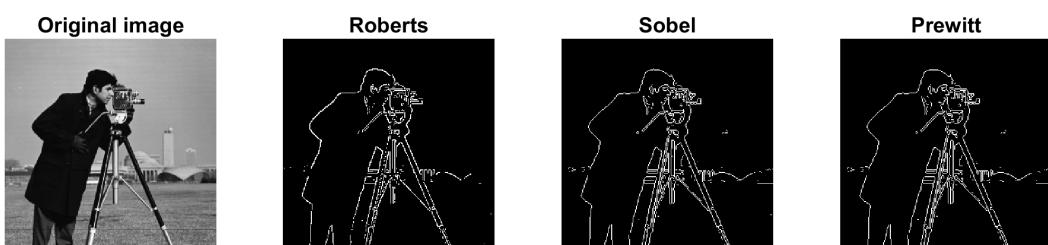


Figura 2.2. Comparativa de algoritmos de detección de borde (Fuente: Ejemplo de MATLAB con imágenes de stock)

- **Operadores basados en laplacianos:** Emplean la segunda derivada para obtener una mejor representación del borde y mayor robustez frente al ruido. Pero a cambio, requieren de una mayor carga computacional [BHM20].

Existen múltiples algoritmos basados en el laplaciano, pero el más importante y usado es Canny. Esto es debido a su alto nivel de precisión y eficiencia. Asimismo, debido a su popularidad, han surgido múltiples variantes del algoritmo de Canny para perfeccionar la detección de borde.

El algoritmo de Canny se realiza en tres etapas. El primer paso es aplicar un filtro gaussiano para suavizar la imagen y evitar falsos bordes producidos por ruido en la imagen. A continuación, se calcula el gradiente de cada punto con su dirección y módulo. En este paso se intenta encontrar todos los bordes posibles de la imagen, pero con una diferencia respecto a los algoritmos anteriores, en este caso solo se queda con los máximos locales, convirtiendo así los bordes en líneas más nítidas y sin el efecto rampa. Por último, se hace un filtrado para eliminar los bordes innecesarios. Para ello se aplica un doble umbral sobre los gradientes y seguido por un rastreo de histéresis. Este elimina todos aquellos posibles puntos que no estén conectados a un borde. Se pueden observar los resultados en la Figura 2.3.



Figura 2.3. Detección de borde por algoritmo de Canny (Fuente: Ejemplo de MATLAB con imágenes de stock)

Detección de formas

Tras procesar la imagen y aplicar los algoritmos de detección de borde, se puede comenzar a analizar la escena. El objetivo de esta etapa es reconocer líneas y formas para poder detectar la posición y orientación del objeto. De nuevo, existen múltiples algoritmos para solventar el problema de localización y determinación a partir de imágenes. En este documento solo se van a analizar dos, la transformada de Hough y RANSAC ya que en múltiples ocasiones han demostrado su robustez y son dos de los algoritmos más extendidos.

- **Transformada de Hough:** Este algoritmo fue diseñado en 1972 por Richard Duda y Peter Hart [DH72] y se ha postulado como una de las mejores alternativas para el reconocimiento de formas geométricas primitivas. Entendiéndose estas como una curva o superficie que pueda ser expresada matemáticamente con tres parámetros [DH72]. Por ejemplo, una línea, círculo, elipse, etc.

Para poder identificar curvas o superficies, el algoritmo analiza cada punto de la imagen y hace pasar por este todos los patrones posibles. Para cada punto se almacena ρ , que representa la distancia mínima entre la recta a analizar y el origen de coordenadas, esta viene dada por una perpendicular a la recta. Y θ que es el ángulo del vector director de esa recta [IK88]. Esto se puede ver gráficamente en la Figura 2.4.

La transformada de Hough ha demostrado numerosas veces su gran robustez para detectar patrones y líneas, pero a pesar de ello, presenta un gran inconveniente que impide que sea usada tan extensamente. Requiere de una alta carga computacional ya que debe analizar cada punto de la imagen y almacenar todas las posibles rectas/formas. Por ello, se han desarrollado multitud de variantes del algoritmo original con el fin de mejorar o reducir su carga. Algunas de las variantes más notables son:

- *Randomized Hough Transform (RHT)*: Se basa en el algoritmo original, pero con la peculiaridad de que no necesita analizar todos los puntos de la imagen. El algoritmo intenta seleccionar de forma aleatoria puntos que puedan formar parte de una posible curva. De esta forma, se reduce bastante la carga computacional.
- *Probabilistic Hough Transform (PHT)*: Parte del esquema de acumulación del algoritmo original, pero difiere en el método de selección del siguiente punto a analizar. Selecciona un subconjunto a analizar en el que se ha incluido puntos previamente detectados como borde. De esta forma se reduce notablemente la carga computacional.
- *Progressive Probabilistic Hough Transform (PPHT)*: Partiendo de la imagen original, se seleccionan puntos de forma aleatoria y se analizan antes de pasar al siguiente. Para cada punto se observa si puede formar parte de una línea o no y se decide en base a los resultados obtenidos por los anteriores puntos. La principal ventaja de este algoritmo es la capacidad de poder ser interrumpido en cualquier momento y aun así dar buenos resultados. Pero es bastante susceptible a dar falsos positivos producidos por ruido en la imagen.

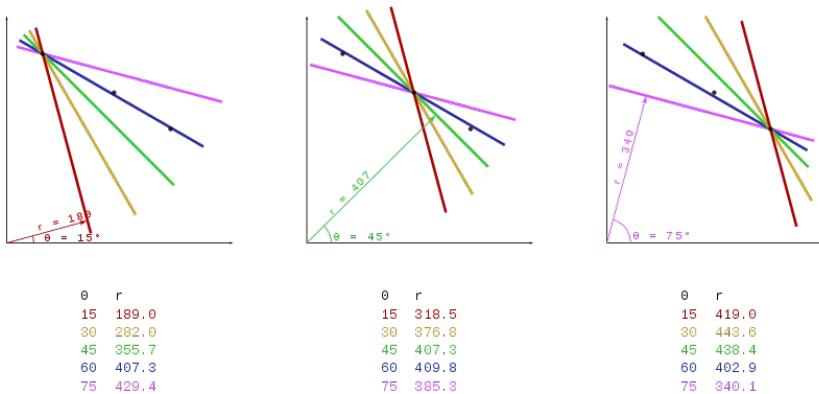


Figura 2.4. Representación de la transformada de Hough (Fuente: https://en.wikipedia.org/wiki/Hough_transform)

- **RANDom SAmple Consensus**: También conocido como RANSAC, se trata de un método matemático iterativo para encontrar los parámetros de un modelo matemático dentro de unos datos. Fue publicado por primera vez en 1981 por Fischler y Bolles y desde entonces se ha convertido en uno de los pilares de la visión artificial [FB81].

El método iterativo se puede separar en varias etapas que se repiten consecutivamente, tantas veces como se desee. Al aumentar el número de iteraciones, aumenta la probabilidad de detectar el objeto. Las etapas de cada iteración son:

- Selección de un subconjunto aleatorio sobre el que trabajar. Se conocen como *inliers hipotéticos*.
- Basándose en el subconjunto, se monta un modelo sobre este.

- Se comprueba la robustez del modelo contrastando contra el resto de los datos.
- Si suficientes puntos se han clasificado como parte del modelo, este es válido.

En la siguiente iteración se partirá de este modelo y se intentará mejorar la detección de este.

Este algoritmo a pesar de ser muy preciso presenta una gran desventaja. Solo es capaz de contrastar la imagen frente a un modelo dado, por ello solo es capaz de detectar un tipo de objeto. Por ello se han desarrollado alternativas como Recursive RANdom SAmple Consensus (R-RANSAC).

2.1.2. Redes neuronales convolucionales

La idea del aprendizaje automatizado/aprendizaje profundo/inteligencia artificial no es novedosa, ya en 1955 IBM formó un grupo centrado en el reconocimiento de patrones supervisado por Nathaniel Rochester. Para ello, decidieron simular el funcionamiento de las neuronas con un ordenador IBM 704 [Vid19]. Y aunque estos conceptos no suenan novedosos, es importante repasar los conceptos principales para poder entender las técnicas de aprendizaje automatizado [Alo+18].

En función del tipo de aprendizaje que se vaya a realizar, los tipos de inteligencias artificiales se pueden separar en 4 categorías:

- Aprendizaje supervisado: la información de la que se va a aprender ha sido preparada y etiquetada. Es un trabajo largo y tedioso, pero le permite a la inteligencia saber qué es lo que debe aprenderse.
- Aprendizaje semi-supervisado: Tal y como su nombre indica, en este caso no toda la información ha sido preparada y etiquetada.
- Aprendizaje no supervisado: La información no ha sido preparada de ninguna forma y es el trabajo de la inteligencia el de entender la información y distinguir qué es lo que se desea aprender.
- Aprendizaje por refuerzo: es un área del aprendizaje automático inspirada en la psicología conductista. La inteligencia debe de decidir que acciones tomar en un entorno y en función de sus decisiones recibirá una recompensa.

Qué tipo de aprendizaje usar depende del tipo de problema que se intente aprender. En este proyecto se va a emplear el aprendizaje supervisado. Para ello se emplearán imágenes previamente etiquetadas para que una red neuronal pueda aprender de ellas.

Desde la década de los cincuenta han surgido múltiples algoritmos para intentar simular el comportamiento de las neuronas y así poder alcanzar el objetivo de aprendizaje automatizado/profundo. Pero no fue hasta 1989 que surgió el concepto de Red Neuronal Convolutacional impulsado por Yann LeCun. Yann desarrolló una de las primeras redes capaces del auto aprendizaje diseñada para reconocer la escritura a mano [Lec+89]. LeCun destacó por el uso de la convolución y fue uno de los primeros en obtener unos resultados positivos con este método.

En 2012 un grupo de investigadores dirigido por Alex Krizhevsky consiguieron obtener el menor error de clasificación visto hasta la fecha sobre las 1.2 millones de fotos de ImageNet con 1000 clases diferentes [KSH17]. Fue gracias a esta hazaña que empezó a crecer un gran

interés por las redes neuronales convolucionales frente al resto de alternativas. Estas nuevas redes se basan en el principio de convolución, el cual ha sido explicado en la Sección 2.1.1. La convolución se caracteriza por ser un proceso completamente lineal, esto es lo que ha permitido el desarrollo de estas redes y que puedan aprender tan fácilmente. En la actualidad este tipo de redes son las más comunes para el análisis de imágenes y su uso está ampliamente extendido.

Dentro del procesado de imágenes por aprendizaje profundo se distinguen dos categorías de redes en función de su objetivo:

- Clasificadores: son capaces de identificar un objeto en una imagen. Pueden ser entrenados para identificar una gran cantidad de objetos, pero tienen la desventaja de que solo pueden detectar un objeto por imagen. Además, solo indican su presencia, no su posición. Uno de los clasificadores más conocidos es AlexNet, la red neuronal diseñada por Alex Krizhevsky y mencionada anteriormente. Se muestra su estructura en Figura 2.5.

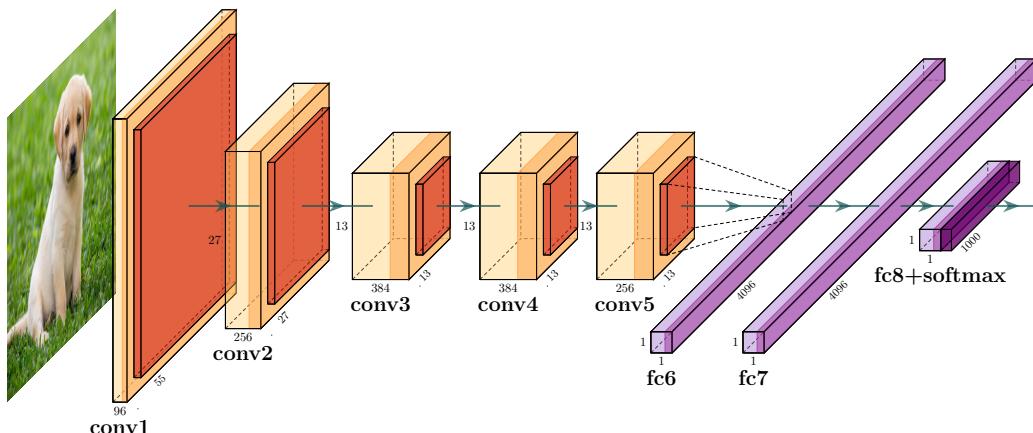


Figura 2.5. Representación de la estructura de AlexNet [KSH17]

En los últimos años los clasificadores han vivido una gran evolución y desarrollo y en parte esto es debido a la competición de ImageNet. Todos los años, investigadores de todo el mundo compiten por intentar obtener el mejor resultado al intentar clasificar millones de imágenes y mil clases. En la actualidad ImageNet cuenta con más de 14 millones de imágenes de alta resolución para ser clasificadas en mil clases. A continuación, se muestra en la Figura 2.6 el avance de las redes en este área analizando el error cometido por estas en ImageNet. Como referencia también se ha añadido el error medio cometido por los humanos. Se puede observar que algunas de las redes más modernas ya son capaces de superar al ojo humano en estas pruebas.

- Detectores de objetos: se basan en los principios de los clasificadores pero por medio de diferentes herramientas son también capaces de localizar el objeto en la imagen. Además, pueden identificar y detectar múltiples objetos en una misma imagen. En la actualidad existe una gran variedad de estructura y métodos para la detección de objetos, pero en este proyecto solo se van a analizar y emplear algunos de los más populares:
 - *Region-proposal Convolutional Neuronal Networks (R-CNN)*: Se basan en el principio de propuesta de regiones frente a sistemas empleados previamente conocidos como ventana flotante. Este sistema consiste en correr un clasificador barriendo todas las posibles secciones de una imagen y con diferentes tamaños. Como este proceso es

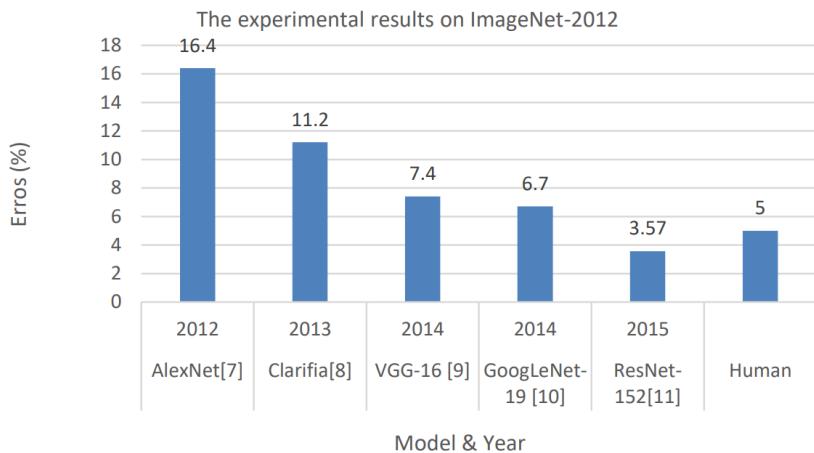


Figura 2.6. Errores cometidos en ImageNet en los últimos años (Fuente: The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches [Alo+18]

muy costoso a nivel computacional y lento, R-CNN propone usar un sistema que proponga un máximo de 2000 regiones a estudiar basándose en colores y formas fácilmente reconocibles. De esta forma se reduce bastante la carga, pero manteniendo un buen nivel de confianza y precisión.

- *Faster Region-proposal Convolutional Neuronal Networks (Faster R-CNN)*: Al igual que R-CNN se basan en el principio de propuesta de regiones pero difiere de este en el método para proponer dichas regiones de interés. En este método, la propia red neuronal se encarga de analizar la escena y decidir qué regiones debe considerar de interés. Esto resulta en una red bastante más rápida, aunque puede ser menos precisa que R-CNN.
- *You Only Look Once (YOLO)*: Es una de las técnicas más actuales y prósperas. Como su nombre indica, se caracteriza porque la imagen solo se pasa una vez por el clasificador y una capa convolucional se encarga de predecir la clase de objeto detectado y su posición. Este tipo de redes se caracterizan por ser extremadamente rápidas, aunque son incapaces de igualar la precisión de R-CNN.

2.2. Robótica industrial

El término robot se remonta hasta 1921 donde por primera vez se introdujo en la obra de teatro R.U.R (Rossum's Universal Robots) obra del novelista y autor checo Karel Čapek. La palabra deriva de *robota* que en checo significa fuerza del trabajo o servidumbre [Oll05]. En la actualidad es un término aceptado y conocido por toda la sociedad, empleado para designar a cualquier “máquina o ingenio electrónico programable que es capaz de manipular objetos y realizar diversas operaciones” (definición de robot según la RAE).

Actualmente los robots representan una gran parte de la fuerza de trabajo industrial. Se presentan en una gran variedad de configuraciones diferentes para cumplir diferentes objetivos, aunque una de las configuraciones más destacadas son los brazos robóticos. Tal y como su nombre indica, se asemejan a brazos de forma que tienen múltiples articulaciones sobre las que rotar para poder moverse en todas las direcciones y orientaciones posibles.

Estos robots se pueden clasificar en función de diversas categorías:

- Por su función. Para ello se han desarrollado diferentes efectores que les permiten realizar tareas tan simples como *Pick and Place* hasta tareas más exigentes como soldar.
- Por su forma. Esta muchas veces se ve determinada por la función que debe desarrollar el brazo, así como el entorno en el que debe trabajar. Aunque a grandes rasgos se pueden distinguir dos tipos, fijos y móviles. Como sus nombres indican, esto depende de la capacidad del robot para moverse libremente.
- Por el sistema de control. Los robots se pueden controlar por control manual, a distancia, mediante programación o con un sistema de aprendizaje por refuerzo.
- Por el modelo de control de la planta. Una planta industrial dotada de numerosos robots tiene dos formas de ser controlada. Se puede tener un sistema de control independiente para cada robot, esto simplifica la creación de la planta, pero dificulta su manejo. O puede contar con un complejo sistema de interconexión entre las diferentes etapas de la planta. Un buen ejemplo de este tipo de sistemas de control es SCADA (Supervisory Control And Data Acquisition) [Pér15].

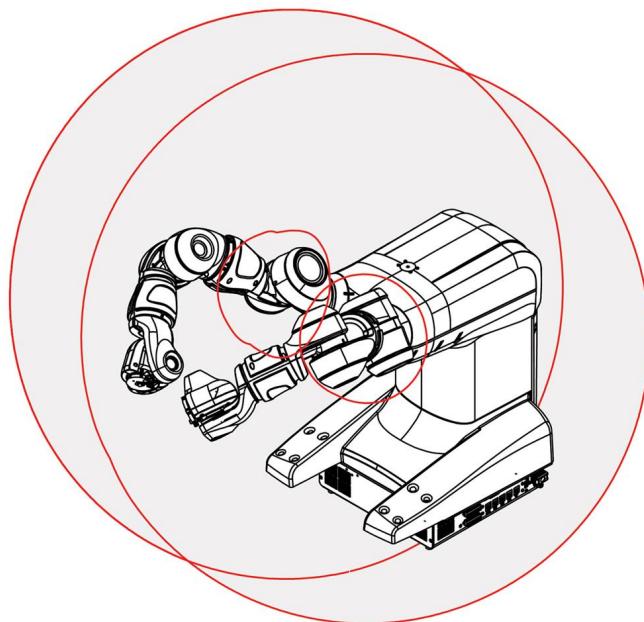


Figura 2.7. Brazo robótico IRB 1400 YuMi empleado en el proyecto con su rango de acción [ABB]

Como ya se ha explicado previamente, el objetivo de este proyecto es el desarrollo de un sistema capaz de localizar y capturar piezas para la preparación de un pedido. Este tipo de sistemas son conocidos como configuraciones *Pick and Place* y son ampliamente usadas en la industria. Estos sistemas suelen estar constituidos por un brazo robot, una cámara y una unidad de procesamiento para analizar las imágenes. En función de los objetos que deben transportar, estos varían su forma de coger los objetos y sus cabezales.

Un buen ejemplo de la importancia de estos sistemas es *Amazon Robotics Challenge*. Con el fin de mejorar los sistemas actuales y de buscar a los mejores en el sector, Amazon ha creado una competición entre alumnos universitarios para desarrollar el mejor sistema *Pick and Place* [Mor+17]. Estos sistemas se basan en redes neuronales muy avanzadas y desarrolladas tales como RefineNet [Mor+17].

3

Metodología de trabajo

Para poder desarrollar e innovar es necesario primero definir una estructura base sobre la que edificar. Y se debe determinar un plan a seguir que sirva de guía.

3.1. Motivación

La industria avanza a un ritmo constante y cada día es más necesario la implantación de sistemas robóticos para poder llevar a cabo tareas que los humanos no podemos desarrollar o que presentan una elevada tasa de errores humanos o tienen un elevado nivel de peligrosidad. Al dotar a estos sistemas de inteligencia y un sistema de visión artificial, se consigue que se pueda adaptar mejor al entorno y se evite tener que re-programar los robots con cada cambio de las condiciones de operación. Avanzamos hacia una sociedad en la que los robots serán la principal mano de obra y para llegar a ese objetivo es necesario invertir y desarrollar más los sistemas actuales. Por ello se ha propuesto como proyecto la integración de un sistema de visión artificial en un robot industrial. Con este proyecto se podrá modernizar las instalaciones del Grupo Antolín® y aumentar las capacidades de la línea de montaje y ensamblaje actual.

3.2. Objetivos

Este trabajo parte de un proyecto actualmente en desarrollo con un sistema de visión artificial ya desarrollado e implantado. Desgraciadamente la implantación actual presenta limitaciones así como una tasa de error superior a los requisitos planteados por el Grupo Antolín.

Para poder superar las limitaciones actuales se ha planteado el desarrollo de un nuevo sistema que debe de mejorar las capacidades de identificación y detección de las piezas así como poder determinar el punto de agarre óptimo. El sistema debe de ser modular de forma que se pueda adaptar fácilmente a nuevas piezas. Y con el fin de que el robot tenga la mayor posibilidad de coger la pieza, también se debe de determinar el vector normal al punto de agarre. Por último, se pide que sea independiente del resto del proyecto de forma que pueda ser fácilmente manejado, modificado y actualizado.

Con el fin de cumplir dichos requisitos se deben plantear unos objetivos de corto, medio y largo alcance a seguir durante todo el desarrollo del trabajo:

- Desarrollo de una base de datos sintética:
 - Investigación y prueba de diferentes sistemas/herramientas (bpi, zpi Zumo labs y Blenderproc).
 - Obtención de primeras imágenes RGB y de profundidad.
 - Obtención de imágenes de profundidad, mapa de normales, identificación de las piezas, mapas de segmentación, etc.
 - Introducción de aleatoriedad y repetibilidad al sistema.
 - Generación de un *Pipeline* para la automatización del proceso.
 - Introducción de ruido en las imágenes para mejorar la robustez.
- Desarrollo de una base de datos real complementaria tanto para la fase de entrenamiento como evaluación del sistema.
- Desarrollo de las nuevas redes neuronales:
 - Investigación y pruebas de diferentes sistemas/herramientas (YOLO, MobilNet, redes regresoras, etc).
 - Planteamiento y definición de la estructura modular a desarrollar. Una primera red para la detección de piezas, seguida de una red para la detección de zonas de intereses con probabilidad de ser el punto de agarre óptimo. Y finalmente una red neuronal del tipo regresor para la determinación del punto de agarre.
 - Primeros desarrollos independientes de las partes modulares del sistema
 - Desarrollo y corroboración del conjunto modular
 - Creación del sistema de evaluación tanto del conjunto como de las partes modulares del sistema
 - Entrenamiento y evaluación de todo el sistema con diferentes configuraciones y bases de datos.
 - Comparativa y determinación del sistema óptimo.
 - Implantación del nuevo sistema de visión artificial

3.3. Arquitectura del sistema

El sistema se compone de tres elementos claves: un robot industrial para poder interactuar con las piezas, una cámara RGB-D para la captura de imágenes y Python para el procesamiento de las imágenes y la toma de decisiones. Es necesario que estos tres elementos funcionen correctamente y estén comunicados entre sí. La arquitectura básica se puede ver en Figura 3.1

A continuación, se va a detallar todo el proceso desde el arranque del sistema hasta el final del mismo y se analizarán cada una de las etapas que constituyen el proceso. Este se puede ver gráficamente en Figura 3.2. El sistema debe componerse con los sistemas actuales los cuales diseñan los pedidos que se deben de preparar y ensamblar. Estos pedidos se reciben y se

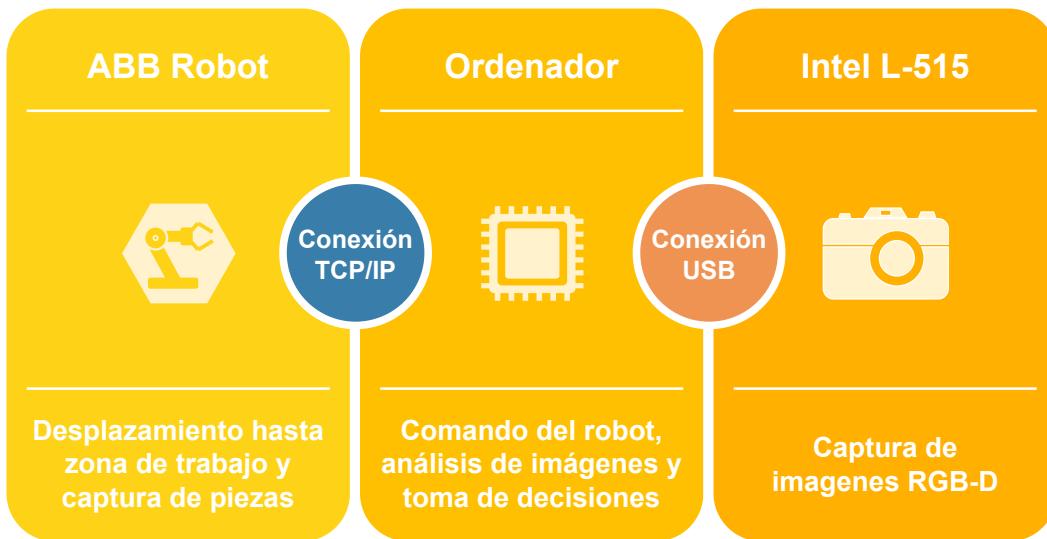


Figura 3.1. Esquema de la arquitectura del sistema

procesan para establecer qué piezas se desean y en qué orden deben de ser recogidas por medio de un proceso cíclico.

Una vez procesado el pedido y establecida la primera pieza a recoger se debe de desplazar el robot a la zona de trabajo y prepararse para la captura de dicha pieza. Una vez reubicado el robot, un sistema de cámaras capturará la escena para determinar la posición de la pieza/piezas que se desean capturar frente a la posición del robot. Esta imagen es analizada por el sistema de visión artificial que se encargará de .^{en} primera instancia identificar las piezas para entre ellas poder centrarse solo en las que se desea capturar. A continuación, si se trata de una pieza de dimensiones grandes se analizará más en profundidad para determinar los posibles puntos de agarre. Y una vez determinados estos puntos de agarre un regresor determinará el centro exacto de dichos puntos de agarre así como el vector normal a dicho punto (dirección que debe emplear la muñeca del robot para capturar la pieza). Si por el contrario se trata de una pieza de dimensiones reducida se empleará el centro de la pieza como punto de agarre.

Esta información es transmitida al robot permitiéndole así poder capturar la pieza y depositarla en la cesta del pedido. Para ello este se deberá aproximar a la pieza seleccionada siguiendo en la medida de lo posible la trayectoria normal determinada. Al aproximarse a la pieza también deberá reducir la velocidad de operación para evitar colisionar.



Figura 3.2. Diagrama de las etapas del sistema

3.4. Herramientas

- Cámara Intel RealSense L515 [Inta] y Wrapper de Python para Intel Realsense [Intb].

- Ordenador con los siguientes SO/paquetes/programas:
 - Ubuntu 20.04 LTS
 - Blender 2.93.5
 - BlenderProc 2.0 (Blender API)
 - PyTorch 1.10.1
 - Tensorflow 2.7.0
 - CUDA Toolkit
 - Python 3.8 o superior
 - Conda ó miniconda

3.5. Cronograma

La planificación del proyecto permite establecer un orden a seguir. No solo da estructura al proyecto, también ayuda a los desarrolladores a estructurar sus ideas. Es por ello que es vital y necesario establecer guías, objetivos e hitos a seguir.

Tabla 3.1. Cronograma del proyecto

Descripción del hito	Categoría	Progreso	Inicio	Días
MVP				
Estado de la Cuestión	Hito	100 %	01/06/2021	14
Base de datos sintéticas v0.1	Objetivo	100 %	15/06/2021	14
Detección puntos de agarre	Hito	100 %	29/06/2021	7
Driver Intel RealSense v0.1	Objetivo	100 %	06/07/2021	7
Regresor - Tensorflow v0.1	Objetivo	100 %	13/07/2021	7
Evaluación regresor	Objetivo	100 %	20/07/2021	3
Base de datos sintética v1.0				
Introducción de Arucos	Objetivo	100 %	01/09/2021	21
Extracción y lectura de normales	Hito	100 %	22/09/2021	7
Preparación de bases de datos	Objetivo	100 %	29/09/2021	21
Actualización a Blenderproc 2.0	Hito	100 %	20/10/2021	7
Replicabilidad	Objetivo	100 %	27/10/2021	7
Riqueza de escenarios	Hito	15 %	03/11/2021	14
Generación de base de datos real	Objetivo	0 %	17/11/2021	7
Visión artificial v1.0				
Estado de la Cuestión	Hito	100 %	01/12/2021	10
Arquitectura del sistema	Objetivo	100 %	11/12/2021	14
YOLO	Objetivo	100 %	25/12/2021	7
TINY YOLO	Objetivo	100 %	01/01/2022	7
Regresor	Objetivo	15 %	08/01/2022	30
Evaluación				
Tamaño del dataset sintético	Objetivo	0 %	14/02/2022	30
Comparativa con dataset real	Objetivo	0 %	16/03/2022	10
Redacción				
Anexo B	Objetivo	100 %	01/12/2021	20
Memoria	Objetivo	15 %	01/04/2022	50

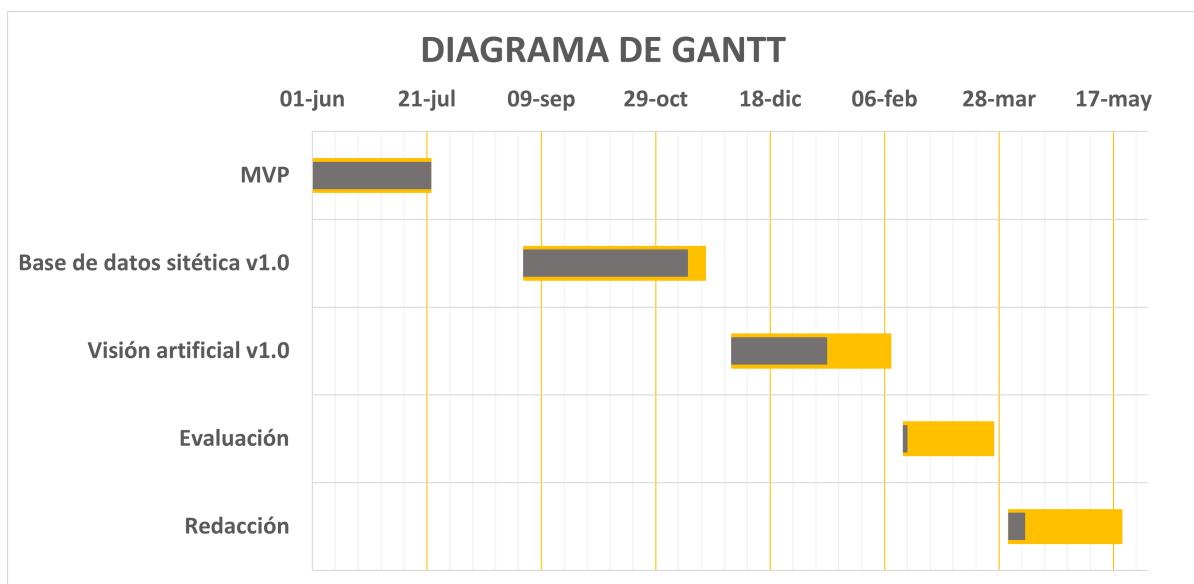


Figura 3.3. Diagrama de gantt del proyecto

4

Generación de un dataset

Un buen sistema de visión artificial requiere de un buen *dataset*. En este capítulo se desarrollará y justificará la creación de un *dataset* sintético que permita expandir las capacidades del sistema de visión artificial.

Se entiende por *dataset* a un conjunto de datos normalmente tabulados/organizados empleados en la ejecución de programas o algoritmos. El término *dataset* es muy amplio y su funcionalidad y formato varía dependiendo del campo de desarrollo pero en este trabajo nos centraremos en una de las áreas donde mas se ha desarrollado el concepto, *Machine Learning*. La capacidad de aprender de un modelo depende en primera instancia de la calidad del *dataset*. Si este no tiene un formato correcto o no es capaz de asegurar la integridad de los datos entonces independientemente del modelo empleado el resultado sera desfavorable. Por lo tanto, la creación del *dataset* es de vital importancia y debe de ser uno de los primeros pasos en el desarrollo de todo sistema basado en *Machine Learning*.

Uno de los métodos mas comunes es la generación de forma manual del *dataset*. Esto implica que un ser humano debe de categorizar, estructurar y definir el valor del dato. Se caracteriza por ser sencillo y rápido para *datasets* pequeños pero presenta numerosos problemas cuando se requiere de datos complejos o de un tamaño elevado. Este proyecto entra dentro de esta categoría y es por ello que se ha creado un sistema de generación de datos que ha permitido automatizar y agilizar el proceso. A este tipo de *datasets* creados por ordenador se les define como sintéticos. Sin embargo no se recomienda depender solo de *datasets* sintéticos ya que estos no reflejan con total precisión la realidad. Es recomendable desarrollar un sistema que mezcle datos reales y sintéticos durante la fase de entrenamientos. Y emplear datos reales para la fase de validación ya que solo de esta forma se puede determinar la verdadera capacidad del modelo.

Pero antes de analizar los datos y las herramientas creadas para obtenerlos, es necesario entender que datos se desean obtener. Para ello se debe de entender el objetivo/problema del proyecto (3.2), el agarre de piezas industriales de diferentes formas y tamaños. Este problema se puede dividir en etapas:

1. Generación del pedido: en función de la demanda y de los requisitos del cliente se debe de generar una lista con todos los componentes necesarios para cada pedido.
2. Estructuración del pedido: Las piezas necesarias se encuentran distribuidas en diferentes secciones y es por ello que se debe de crear un que determine el orden de recolección de piezas óptimo que reduzca el tiempo recolección.
3. Recolección de piezas: Se trata de un proceso iterativo que se debe de realizar para cada una de las piezas que constituyen el pedido.
 - Desplazamiento hasta la pieza. Dependiendo de la configuración del robot este sistema variará, pero el objetivo siempre será el mismo. Trasladar el robot hasta la región donde se encuentra la pieza a recolectar con el fin de poder capturarla y situarla dentro de la región de alcance.
 - Detección de la pieza: aplicando algoritmos de detección se identificará la pieza que se desea recolectar. Dentro de una misma zona se detectarán numerosas instancias de una misma pieza. Se debe de escoger la pieza/piezas mejor ubicadas y con más probabilidad de éxito.
 - Punto de agarre: tras detectar la pieza se debe de determinar como se debe de agarrar la pieza. Para ello se debe de determinar el punto de agarre óptimo y el vector normal a dicho. Para las piezas pequeñas se puede emplear el centro de la pieza como punto de agarre. Sin embargo, para piezas más grandes este método no funciona ya que se trata de piezas irregulares que debido a su gran tamaño requieren de un buen agarre capaz de levantar el peso de la pieza. Por ello se debe de buscar zonas lisas sobre las que se pueda emplear una ventosa o superficies más complejas que permitan el uso de un cabezal *soft-robotics*. Esto implica que la salida final del sistema debe de ser un punto de agarre y el vector normal a dicho punto que debe de seguir el brazo robótico.
 - Recolección de la pieza: se transfiere la información necesaria al robot para que este pueda recolectar la pieza a través del punto de agarre definido. El sistema de agarre a emplear dependerá del tipo de pieza.
 - Deposición y control de calidad: por último se debe de depositar la pieza dentro de la cesta que constituye el pedido. Y mediante el sistema de visión artificial se debe de comprobar que la pieza deseada ha sido correctamente depositada en la cesta.
4. Control de calidad y trazabilidad: antes de dar por finalizado el pedido se analiza por última vez para comprobar que todas las piezas necesarias se encuentran dentro de la cesta. Se registra en el sistema el pedido y toda la información necesaria para futura trazabilidad.
5. Traslado del pedido: una vez se da por finalizado el pedido este se debe de trasladar hasta la zona de ensamblaje para comenzar el proceso de montado.

4.1. Estructura del dataset

Una vez entendido el problema se puede determinar el tipo de datos que se necesitan y para este problema se puede observar que dependen en gran medida de la pieza. Por ello se

debe de distinguir entre las piezas grandes y las piezas pequeñas. A continuación, se muestra la estructura estándar que deben de seguir las piezas pequeñas y grandes:

Piezas pequeñas

- Imagen en formato ".png" de toda la escena.
- Archivo ".txt" para identificar y detectar las piezas presentes en la imagen. Se emplea una fila para cada pieza y en este se debe de mostrar:
 - Categoría: identifica el tipo de pieza.
 - Centro x: coordenada horizontal en píxeles (normalizados) del centro del rectángulo que engloba la pieza en la imagen.
 - Centro y: coordenada vertical en píxeles (normalizados) del centro del rectángulo que engloba la pieza en la imagen.
 - Ancho: dimensiones en píxeles (normalizados) del ancho del rectángulo.
 - Largo: dimensiones en píxeles (normalizados) del largo del rectángulo.

Piezas grandes

- Imagen en formato ".png" de toda la escena.
- Archivo ".txt" para identificar y detectar las piezas presentes en la imagen. Idéntico al archivo ".txt" de las piezas pequeñas.
- Imágenes recortadas de cada una de las piezas presentes en la imagen global.
- Archivo ".txt" para identificar y detectar en cada una las imágenes recortadas las regiones de interés donde hay presentes zonas de agarre. Idéntico al archivo ".txt" anterior pero en este caso en lugar de piezas, lo que se detecta son regiones. Y todas las dimensiones se refieren a la imagen recortada.
- Archivo ".txt" para determinar el punto de agarre de cada una de las regiones de detectadas.
 - Nombre: identifica el tipo de región.
 - Centro x: coordenada horizontal en píxeles (normalizados) del centro del punto de agarre respecto a la imagen recortada.
 - Centro y: coordenada vertical en píxeles (normalizados) del centro del punto de agarre respecto a la imagen recortada.
 - u: coordenada normalizada respecto al eje x del vector normal.
 - v: coordenada normalizada respecto al eje y del vector normal.
 - w: coordenada normalizada respecto al eje z del vector normal.

4.2. Herramientas

La generación sintética de imágenes es una tarea compleja que requiere de avanzados motores y simuladores que permitan representar un entorno realista. Actualmente se trata de un campo todavía en desarrollo y por lo tanto no se dispone de herramientas específicamente diseñadas para cumplir ese objetivo. Sin embargo, si que existen sistemas/plugins que permiten adaptar sistemas/herramientas ya existentes para permitir la generación de datasets sintéticos. Para el desarrollo de este proyecto se ha escogido un sistema de esta categoría, BlenderProc. Es una *Application Programming Interface (API)* que permite controlar el programa Blender para la generación de datasets.

4.2.1. Blender

Blender es una aplicación gratuita y *open source* bajo una licencia GNU GLP que permite desarrollar proyectos 3D por completo. Cuenta con sistemas todos los sistemas necesarios para proyectos 3D: modelado, *rigging*, animación, simulación, renderizado, composición, *motion tracking*, edición de video y desarrollo de videojuegos. También permite un uso avanzado gracias a la existencia de una API basada en Python llamada *BPy*.

Es gracias a la existencia de esta API que la comunidad puede desarrollar extensiones para Blender y así expandir las capacidades de este. Esto es de vital importancia para el desarrollo de este proyecto ya que para el desarrollo del *dataset* sintético se ha empleado un *pipeline* que facilita y automatiza la creación de *datasets*.

4.2.2. BlenderProc

BlenderProc es un *pipeline* desarrollado por German Aerospace Center (DLR) - Institute of Robotics and Mechatronics (RM) (DLR-RM) con el fin simplificar el proceso de generación de *datasets* centrados en imágenes para entrenar redes convolucionales [Den+19]. Se caracteriza por centrarse en la modularidad al dotar a blender de un sistema con las herramientas necesarias para generar imágenes foto realistas pero que manteniendo una estructura modular que permite adaptarse al problema.

Algunas de las utilidades de este *pipeline* son: cargar, modificar, eliminar escenas y objetos, variar la iluminación, la composición de la escena, aplicar simuladores de físicas para obtener escenas realistas, renderizar imágenes de color, profundidad, distancia y normales. Y todo esto se realiza mediante *scripts* que llaman a la API de BlenderProc y esta se encarga de cargar y controlar Blender.

4.2.3. Aruco

Empleando Blender se puede renderizar imágenes foto realistas pero se sigue requiriendo del punto de agarre y del vector normal a dicho punto. Para ello se empleara un sistema de posprocesamiento basado en Aruco [AVA20] que permite la extracción de la información restante. Aruco es una librería basada en OpenCV diseñada para aplicaciones de realidad aumentada cuya principal ventaja y objetivo es la simplicidad. Permite detectar QR con un simple linea de código, el altamente eficiente y permite trabajar con múltiples diccionarios.

4.3. Arquitectura del generador de datasets

El objetivo del generador de imágenes es obtener un número elevado de muestras con las que poder entrenar varios modelos de redes neuronales. Y tal como se ha definido en la Sección 4.1, las muestras deben de estar constituidas por varias imagen así como información respecta a la posición de las piezas en la imagen e información respectos a los puntos de agarre. Desgraciadamente, no se ha podido determinar un único sistema que soporte la generación de las múltiples imágenes así como la definición de los puntos de agarre y por ello se ha tenido que crear una capa de posprocesamiento que permita extraer la información adicional necesaria.

Este posprocesado requiere de la generación de una segunda imagen con una versión modificada de las piezas. estas piezas presentan QR (Aruco) en las regiones de interés donde hay



Figura 4.1. Esquema de la arquitectura del sistema

presente un punto de agarre. Gracias a la combinación de ambas imágenes se puede obtener todos los datos necesarios. El proceso se desarrolla en detalle en la Sección 4.4

Como se puede observar en la Figura 4.1, se trata de un proceso iterativo constituido por varias etapas que se ejecutan en serie para la generación de cada nueva imagen. A excepción de la primera etapa que consiste en la generación de los modelos 3D, esta etapa no forma parte del generador pero este depende de la existencia de un modelo 3D con el que poder trabajar.

En esta etapa se han diseñado múltiples piezas de diferentes tamaños con la ayuda de Blender así como múltiples entornos sobre los que más adelante se dispondrán las piezas. Para las piezas se parte de un archivos .stl que contienen la información dimensional de las piezas. Estos han sido cargados en Blender donde se ha modificado el número de triángulos para mejorar el rendimiento y la suavidad de las piezas. Y se ha generado las texturas necesarias para las piezas las cuales incluyen defectos y patrones que deben de variar para cada instancia de estas añadiendo así aleatoriedad. Los entornos se han desarrollado por completo en Blender y se caracterizan por ser sencillos y planos. Su objetivo es contener a las piezas y no formar 0 confundirse con estas. Para los entornos se empleará una textura aleatoria para cada imagen de una librería *open source* CCTextures y constituida por más de mil seiscientas texturas de alta calidad.

Las piezas grandes requieren del uso de códigos QR que permitan en la etapa de posprocesado (4.4) extraer la información necesaria. Es por ello que en la etapa de modelado se debe de crear duplicados de las piezas grandes a los que se les insertará códigos QR en las zonas con posibles puntos de agarre.

Partiendo de los modelos y texturas necesarios, el sistema desarrollado puede empezar a generar imágenes:

1. Escena: Carga del entorno, de las piezas, la iluminación y la cámara

- Entorno: Se escoge de forma aleatoria uno de los entornos disponibles y se carga dentro de blender. A continuación, se escoge una de las texturas disponibles y se aplica en la totalidad del entorno. Por último, se centra el entorno y se deshabilita el cuerpo dentro del motor de físicas. De esta forma se fija la posición del entorno y este pasa a ser inalterable.
- Piezas: Se escoge de forma aleatoria el número de piezas que estarán presentes en la escena (se establecen mínimos y máximos dependiendo del tamaño de la pieza) y se cargan dentro de blender. Por último, se define una propiedad *category_id* que más adelante será usada para identificar las distintas piezas.

- Iluminación: Se parte de la base de que el entorno de trabajo real estará bien iluminado, por ello que se han definido unas condiciones de iluminación fijas y óptimas que permitan reducir las sombras. Se planteo emplear una iluminación aleatoria pero esta daba lugar a demasiadas sombras y dificultaba ver el contorno de las piezas oscuras.
- Cámara: Para asemejarse a la realidad se ha fijado una cámara cenital a la escena y se ha implementado los parámetros intrínsecos de la cámara real que se va a emplear dentro de blender.

2. Simulación: activación de las piezas, posicionamiento de las piezas y simulación física.

- Activación de las piezas: se activa el motor de físicas para todas las piezas de forma que se vea afectadas por la gravedad y el entorno.
- Posicionamiento de las piezas: con la ayuda de la API de BlenderProc se posicionan todas las piezas de forma aleatoria dentro de un volumen de trabajo. El volumen de trabajo se fija en base a las dimensiones del entorno y la posición y orientación de la pieza se determina de forma aleatoria con el motor de aleatoriedad de numpy y BlenderProc.
- Simulación física: se ejecuta una simulación física en la que se deja caer las piezas sobre el entorno/zona de trabajo. Se ejecuta el motor hasta que todas las piezas se vuelvan estáticas o se alcance el máximo tiempo de simulación de cuatro segundos.

3. Renderizado: configuración del motor de renderizado, renderizado y escritura de la salida de blender.

- Configuración: dependiendo del tamaño de la pieza se fija la configuración necesaria.
 - Piezas grandes: resolución 3840x2160, 50 muestras y se activa el mapa de normales y de profundidad.
 - Piezas pequeñas: resolución 1920x1080, 25 muestras.
- Renderizado: se activa el proceso o procesos de renderizado dependiendo del tamaño de la pieza.
- Salida: se transfiere el resultado de los renderizados del archivo temporal de trabajo de blender al lugar de salida deseado.

4.4. Posprocesado

El *dataset* deseado debe de contener para cada imagen una lista de todas las piezas presentes así como su posición y los puntos de agarres disponibles. Esta información no se puede obtener de forma directa con Blender y BlenderProc, es por ello que se ha tenido que desarrollar varios sistemas de posprocesado específicos para cada uno de los componentes de la red neuronal.

- Posición (*YOLO*): el primer procesado se caracteriza por ser una simple transformación de la información generada por Blender y BlenderProc a un formato que la red neuronal pueda entender. El generador de imágenes es capaz de crear un archivo de texto contenido la posición de las piezas en la imagen y el mapa de segmentación. Sin embargo, estas se encuentran en un formato distinto al empleado por la red neuronal. La

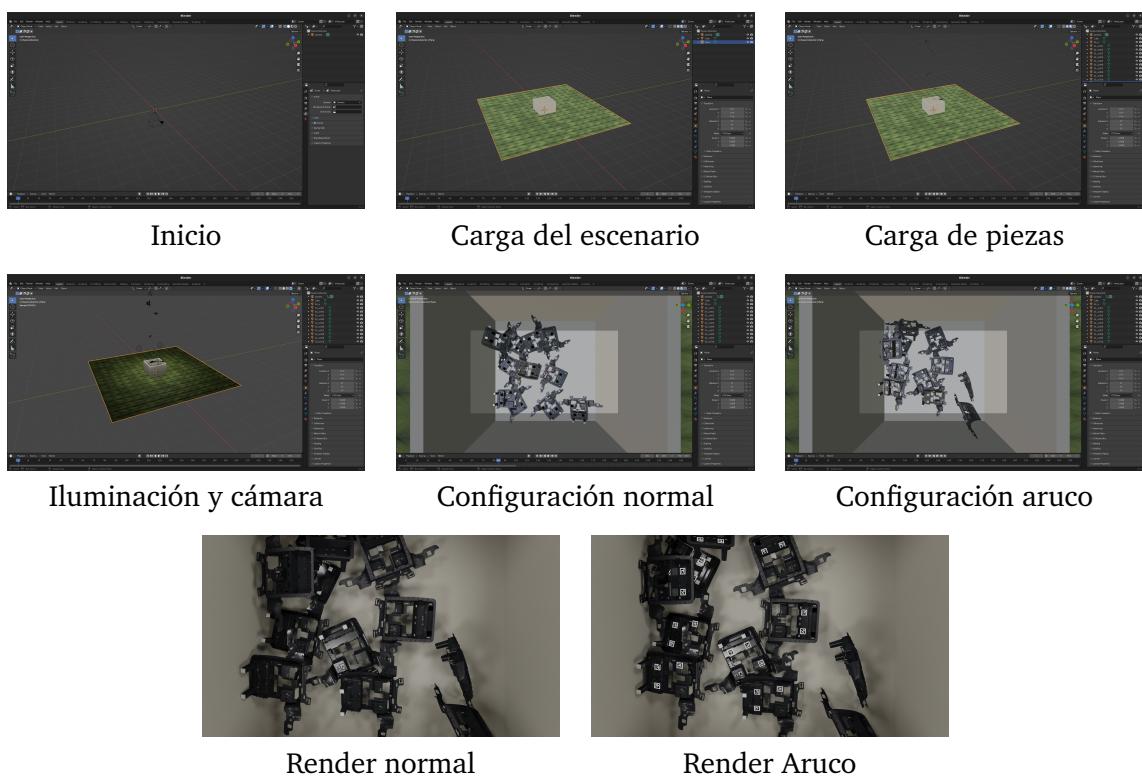


Figura 4.2. Proceso de generación de imágenes

transformación empleada consiste en un cambio de origen de la posición de las piezas y en la eliminación del resto de información que no es relevante. El proceso a seguir se muestra a continuación:

```

1 # Obtain everything inside input_paths
2 for folder in normal_path:
3     # Obtain all files inside each folder
4     image = data_loading(normal_path, folder)
5
6     # Load and treatment of Coco Annotations
7     bboxes, categories = coco_treatment(normal_path, folder)
8
9     # Saving of images and data
10    data_saving(num_images, save_path, image, bboxes, categories)
11
12    # Increase the number of processed images
13    num_images += 1

```

- Zonas de interés (*Tiny YOLO*): el segundo procesado debe de determinar para cada piezas las posibles zonas de agarre. Para ello se debe de obtener para cada imagen un recorte de cada pieza presente en la imagen. Y la posición de las zonas de interés de cada recorte. Para llevar a cabo esta tarea se va a emplear la librería Aruco que se caracteriza por poder detectar códigos QR con precisión, rapidez y eficiencia. El proceso a seguir se muestra a continuación:

```

1 for folder in normal_path:
2     # Obtain all files inside each folder

```

```

3     image, normals, _, _ = data_loading(normal_path, folder)
4     image_aruco, _, _, _ = data_loading(aruco_path, folder)
5
6     # Load and treatment of Coco Annotations
7     bboxes, _ = coco_treatment(normal_path, folder)
8
9     # Analyze and detect indexes of interest
10    id_category = PIECE
11    indexes = [index in categories if id_category]
12
13    # Analysis of each piece
14    for _, j in enumerate(indexes):
15        # Creation of crops and category
16        bbox = bboxes[j]
17        image_crop = crop_creation(image, bbox)
18        image_aruco_crop = crop_creation(image_aruco, bbox)
19        image_normals_crop = crop_creation(normals, bbox)
20
21        # Aruco detection
22        corners, ids, Point_names = detect_markers(image_aruco_crop,
23                                                    matrix_flag=True,
24                                                    distortion_flag=True)
25
26        # Saving of images and data
27        data_saving(num_images, save_path, image_crop,
28                    corners, Point_names)
29
30        # Increase the number of processed images
31        num_images += 1

```

- Vectores normales (Regresor): el objetivo es obtener los vectores normales y el punto de agarre para cada pieza. Por ello la salida en este caso será una imagen individual para cada pieza y un archivo de texto para cada punto de la pieza. En este archivo se almacenarán los centros de cada punto y las coordenadas de su vector normal. El proceso para la obtención de la imagen de cada pieza individual es idéntico al de las zonas de interés. Este posprocesado se caracteriza por el tratamiento de la imagen de normales y la generación de los archivos de texto con la información de cada punto de agarre. El proceso a seguir se muestra a continuación:

```

1 for folder in normal_path:
2     # Obtain all files inside each folder
3     image, normals, _, instance_segmap = data_loading(normal_path, folder)
4     image_aruco, _, _, _ = data_loading(aruco_path, folder)
5
6     # Load and treatment of Coco Annotations
7     bboxes, _ = coco_treatment(normal_path, folder)
8
9     # Analyze and detect indexes of interest
10    id_category = PIECE
11    indexes = [index in categories if id_category]
12
13    # Analysis of each piece
14    for _, j in enumerate(indexes):
15        # Creation of crops and category
16        bbox = bboxes[j]

```

```

17     image_crop = crop_creation(image, bbox)
18     image_aruco_crop = crop_creation(image_aruco, bbox)
19     image_normals_crop = crop_creation(normals, bbox)
20
21     # Aruco detection
22     corners, ids, Point_names = detect_markers(image_aruco_crop,
23                                                 matrix_flag=True,
24                                                 distortion_flag=True)
25
26     # Exclusion of points from other pieces
27     corners_clean, ids_clean, Point_names_clean = segmentation_detect(
28         bbox, corners, total_padding, instance_segmap)
29
30     # Extraction of normals
31     normal_coords = normal_extraction(
32         image_normals_crop, corners_clean)
33
34     # Saving of images and data
35     data_saving(num_images, save_path, image_crop,
36                 corners_clean, Point_names_clean)
37
38     # Increase the number of processed images
39     num_images += 1

```

4.5. Aleatoriedad

Uno de los factores más importantes durante el desarrollo de este sistema es el sesgado. Se debe de evitar que el sistema genere un *dataset* sesgado lo cual implica comprobar y asegurar la aleatoriedad de los resultados obtenidos. Para ello se empleará siempre que sea posible motores de aleatoriedad. Estos se caracterizan por ser capaces de obtener distribuciones uniformes (la mayor expresión de la aleatoriedad ya que todos los resultados son equitativamente probables). Con la excepción del generador de rotaciones aleatorias, ya que debido a la naturaleza de los ángulos de euler la distribución no puede ser uniforme. Durante el proceso de generación de imágenes se emplea un total de tres motores de aleatoriedad. Para asegurar la veracidad de estos tres motores, se ha realizado una serie de pruebas con diferentes semillas y múltiples muestras. A continuación, se muestran los resultados de todos los motores de aleatoriedad empleados.

El primer motor empleado se trata del generador de una distribución uniforme de numpy. Como su nombre indica, se encarga de generar un valor entre los máximos y mínimos establecidos de forma que todos los valores presenten la misma probabilidad. Este método es empleado para determinar el número de piezas que se deben de cargar así como su posición dentro de la zona de trabajo.

El segundo motor empleado es *choice* del paquete *random* de Python. Este se encarga de escoger de forma aleatoria un elemento de una lista. Se debe de asegurar de que todos los elementos presenten la misma probabilidad de ser escogidos. Este método es empleado para determinar que tipo de escenas, piezas y texturas se deben de emplear.

El último motor empleado es *UniformSO3* de BlenderProc. Este se encarga de determinar una rotación aleatoria para las piezas. Esta rotación está definida por ángulos de euler, es por ello que las distribuciones no deben de ser uniformes. Esto se debe a que una misma posición puede ser alcanzada por diferentes combinaciones de ángulos.

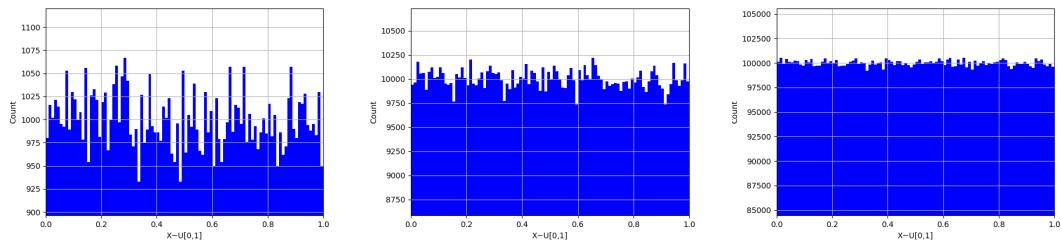


Figura 4.2. Aleatoriedad de una distribución uniforma generada por numpy

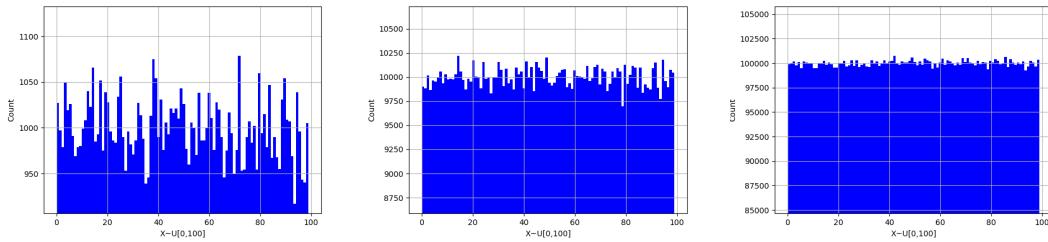


Figura 4.3. Aleatoriedad del comando *choice* del paquete *random*

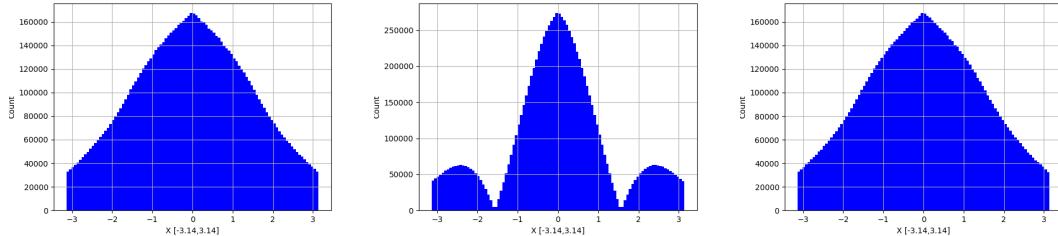


Figura 4.4. Aleatoriedad de *uniformSO3* del paquete BlenderProc

4.6. Resultados

Gracias a la implantación de un generador de imágenes se ha conseguido obtener un elevado número de instancias con las que poder entrenar la red neuronal que reflejan en gran medida la realidad. A continuación se muestran algunas de las imágenes obtenidas por es ambos sistemas.



Figura 4.4. Muestra de imágenes obtenidas mediante el generador de imágenes

Como se ha mencionado previamente, es importante entrenar modelos de redes neuronales con *datasets* no sesgados. Esto implica que antes de entrenar un modelo se debe de analizar los datos para comprobar que se cumplen dichos requisitos. Para ello se han desarrollado una serie de *scripts* en Python que permitirán mostrar el verdadero estado del *dataset*.

Como para el desarrollo de este proyecto se deben de desarrollar tres modelos de redes neuronales, se debe de analizar los datos necesarios correspondiente a cada uno de estos modelos:

- *YOLO*: se trata de la primera red neuronal empleada y desarrollada. Se encargará de detectar las piezas dentro de la zona de trabajo y se empleará como base para decidir que pieza se debe de recolectar. La salida de este modelo debe de ser las piezas presentes en la imagen y su posición. Por ello, para poder ser entrenada correctamente se debe de disponer de un número de instancias similares para cada pieza que se desea detectar.
- *Tiny YOLO*: una red neuronal menor y menos potente pero con una mayor especialización. Esta red se encargará de determinar regiones con posibles puntos de agarre dentro de las piezas (previamente identificadas por *YOLO*). Esta segunda capa es específica para cada pieza y solo se aplicará a aquellas piezas de elevado volumen. Es por ello que para el análisis de los datos de entrenamiento nos debemos de fijar de forma individual e independiente en cada pieza. En este caso nos vamos a centrar en las piezas G1 y G3 y en el número de instancias de cada zona de interés presente en estas.
- Regresor: se trata de la última capa del sistema de visión artificial. Se basa en la salida del *Tiny YOLO* y determina para cada una de las posibles regiones de interés el punto de agarre óptimo y su vector normal. Esta última capa es específica para cada pieza y solo se aplicará a aquellas piezas de elevado volumen. Es por ello que para el análisis de los datos de entrenamiento nos debemos de fijar de forma individual e independiente en cada pieza. Además, al tratarse de un regresor es importante fijarse en la distribución de muestras con el fin de evitar emplear un *dataset* sesgado. En este caso nos vamos a centrar en la pieza G1_a.

4.7. Conclusiones y futuros desarrollos

Gracias a los resultados mostrados en el capítulo anterior se puede observar el verdadero potencial del generador de imágenes, pero también se aprecia los fallos y puntos de mejora. El sistema ha demostrado ser una opción viable capaz de representar la realidad de forma fiel. Capaz de mantener una constante aleatoriedad a la vez que asegura la replicación. Y ha demostrado ser un sistema adaptable y escalable capaz de trabajar con cualquier tipo de pieza.

Pero a su vez ha mostrado varios fallos de diseño que se deben a error humano y no del sistema. Desgraciadamente el *dataset* generado no es perfecto ya que presenta un elevado sesgo a pesar de representar en gran medida la realidad. A la hora de desarrollar el sistema se enfoco el proyecto en recrear la realidad en lugar de pensar en que tipo de *dataset* se requiere para entrenar la red neuronal. Y debido a la naturaleza de las piezas la realidad es un escenario sesgado en donde la mayoría de las piezas se encuentran en una posición horizontal.

Para evitar este problema en futuros desarrollos se recomienda la introducción de nuevos escenarios con diferentes geometrías que permitan obtener instancias de las piezas con mayor riqueza y en diferentes posiciones. Este tipo de escenarios no deben de presentar un suelo plano sino que deberán de presentar diferentes elevaciones de forma que al apoyarse las piezas no queden en posición horizontal.

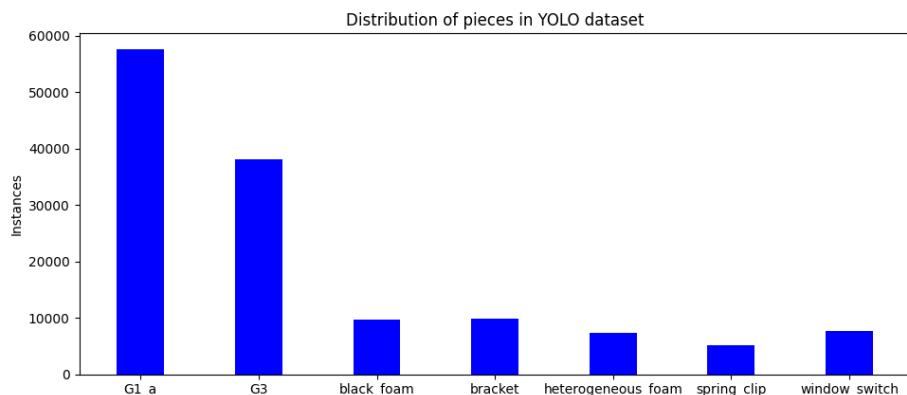


Figura 4.4. Instancias de cada pieza empleadas para entrenar el modelo basado en YOLO

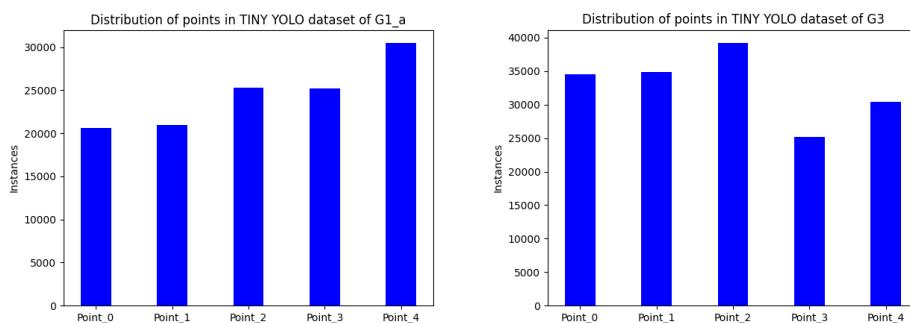


Figura 4.5. Instancias de cada zona de interés empleadas para entrenar el modelo basado en Tiny YOLO

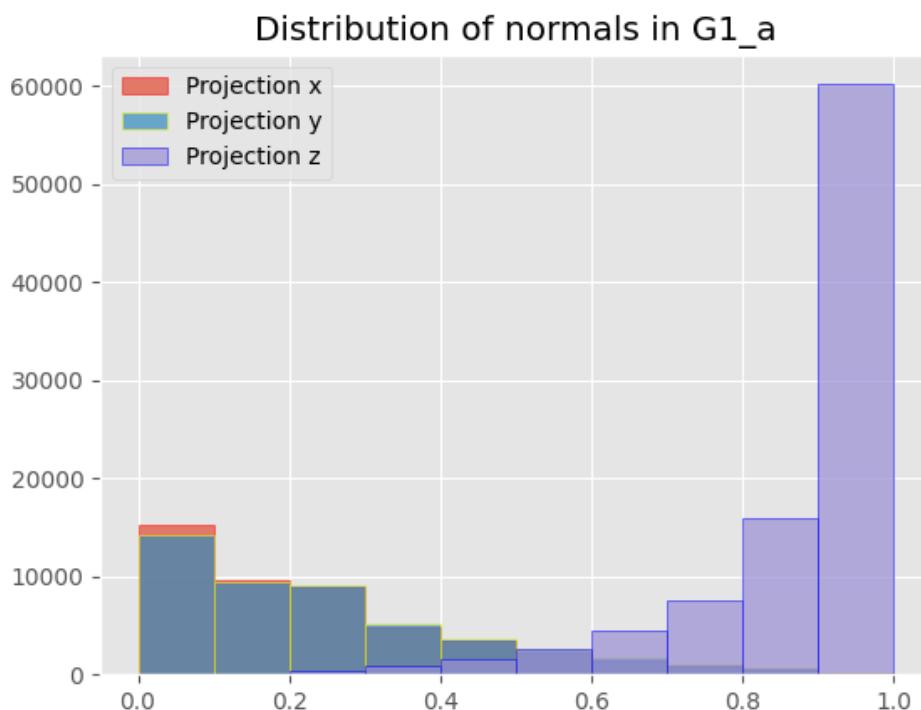


Figura 4.6. Distribución de las proyecciones normales de los puntos de agarre de G1

5

Sistema de visión artificial

En este capítulo se van a desarrollar los tres componentes que constituyen el sistema de visión artificial. Una primera red para identificar las piezas, una red intermedia para distinguir regiones de interés y una última capa para determinar los puntos de agarre.

El fin de este proyecto es la generación de un nuevo sistema de visión artificial para el reconocimiento de piezas de uso industrial. Este sistema se implantará dentro de una cadena de suministro del Grupo Antolín® que a su vez alimentará a una línea de montaje y ensamblaje. El sistema deberá de poder identificar múltiples piezas y determinar el punto de agarre óptimo de cada pieza. Este se ve definido por sus coordenadas así como el vector normal a la superficie. De esta forma un brazo robótico con un sistema de agarre por aspiración, ventosa o *soft-robotics* (varía en función de la pieza a coger) podrá recolectarlas.

La multitud de herramientas de agarre así como de la necesidad de un sistema de determinación de puntos de agarre se debe a la gran variedad de piezas existentes con formas y tamaños de gran variedad (1-30 cm). Esto ha obligado el desarrollo de un sistema modular capaz de adaptarse dependiendo del tamaño de la pieza. Se ha definido un primer proceso común para todas las piezas basado en YOLOv5 que permite la identificación de las piezas. Para las piezas pequeñas se empleará el punto medio de la pieza identificada como punto de agarre. Sin embargo, esta lógica no se puede emplear para las piezas grandes debido a su complejidad y a las superficies irregulares. Por ello, se ha definido un proceso adicional que parte de la salida de YOLO y es capaz de determinar el punto de agarre óptimo. El proceso está a su vez constituido por dos etapas.

- YOLO: consiste en una red convolucional del tipo YOLOv5 de gran tamaño. Se encarga de identificar y detectar las piezas presentes en el entorno de trabajo. Para recolectar las piezas pequeñas se empleará el punto medio de la pieza y un vector perpendicular a la superficie de trabajo. Sin embargo, para las piezas grandes se requiere de un proceso más avanzado basado en Tiny YOLO y un regresor.
- Tiny YOLO: como su nombre indica, se trata de una red convolucional del tipo YOLO pero de menor tamaño capaz de determinar para una pieza sus regiones con posibles puntos de agarre.

- Regresor: se trata de una red neuronal convolucional del tipo regresor capaz de determinar para las regiones de interés de cada pieza el punto de agarre. Este está definido por las coordenadas así como el vector normal a dicho punto.

El uso de una capa intermedia (Tiny YOLO) se debe a la naturaleza de un regresor. Este tipo de red es capaz de estimar para una pieza los diferentes puntos de agarre. Pero no dispone ningún mecanismo que nos permita calcular la probabilidad de que ese punto haya sido determinado correctamente. Una forma clara de visualizar este concepto es suponer un escenario muy común en la práctica. Supongamos que una pieza está dada la vuelta de forma que el punto de agarre se encuentre apuntando al suelo. En la práctica esta pieza no puede ser recolectada por el brazo robótico ya que no tiene acceso al punto de agarre. Por lo que la salida del sistema debería de ser nula. Sin embargo, debido a la naturaleza de los regresores, este intentará determinar el punto de agarre aunque no pueda verlo y este no sea accesible. Es debido a esto que se debe de emplear una capa intermedia que determine si el punto que se quiere agarrar es accesible por el robot o no.

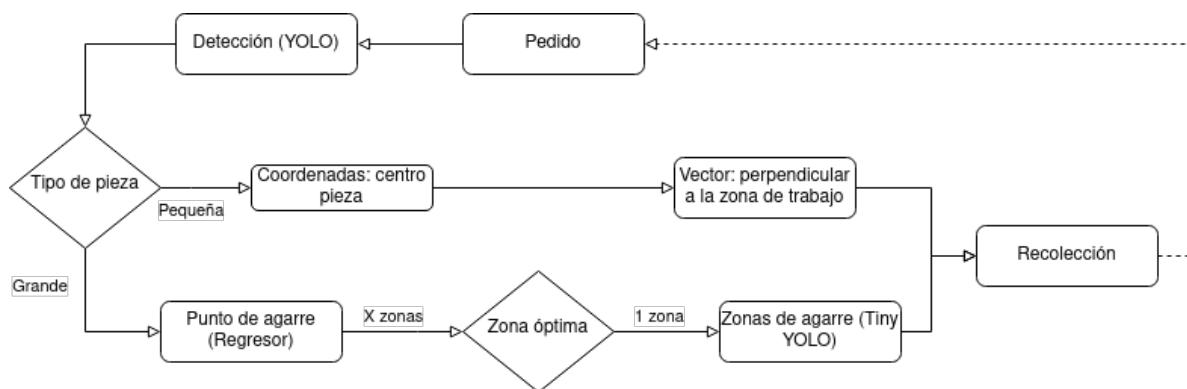


Figura 5.1. Esquema de la arquitectura del sistema

SO	Ubuntu 18.04
Kernel	5.15.0-40-generic
Procesador	Intel Core i5-8250U @ 3.40GHz
Nucleos/Hilos	4/8
Ram	16 GB 2666 MHz
GPU	Intel UHD 620

Tabla 5.1. Ordenador empleado para el entrenamiento

5.1. YOLO

En 2015 se produjo una revolución que cambió la forma de trabajar con las redes neuronales y nuestro entendimiento de estas. En 2015 cuatro investigadores de la universidad de Washington desarrollaron un nuevo tipo de red neuronal bautizada como YOLO (You Only Look Once) [Red+15]. Esta nueva red se caracteriza porque tal y como su nombre indica, la imagen solo se analiza una vez. Esto puede parecer similar a redes como Faster R-CNN pero difiere bastante. En Faster R-CNN la imagen es inicialmente procesada por las capas de convolución para a continuación determinar regiones de interés con la ayuda de una red tipo RPN. Finalmente, se analiza cada una de las posibles regiones de interés por separado. Esto implica que aunque inicialmente la imagen pase completamente por las capas de convolución, después se analiza por secciones por lo tanto es necesario analizar la misma imagen varias veces.

Sin embargo, esto no sucede en YOLO por la forma interna en la que trabaja. YOLO no fue diseñada con la idea de reutilizar un clasificador ya existente para detectar objetos, se construyó desde cero y con el único objetivo de detectar objetos. Es decir, para detectar objetos no recurre a la técnica de correr un clasificador por diferentes regiones de la imagen y así detectar un objeto. YOLO emplea un método completamente diferente, este comienza por la división de la imagen en secciones, en el artículo original, se divide la imagen en una 7×7 secciones, aunque el número de secciones se pueden modificar. Para cada una de las secciones una capa convolucional predice un número de posibles *bounding boxes* (área de la imagen que puede contener un objeto) así como la posible clase de cada *bounding box* y su probabilidad de ser ese objeto. Por último, se analizan en conjunto todos los *bounding boxes* definidos así como su clase y su probabilidad y se determina cuáles son los objetos en la imagen y donde están.

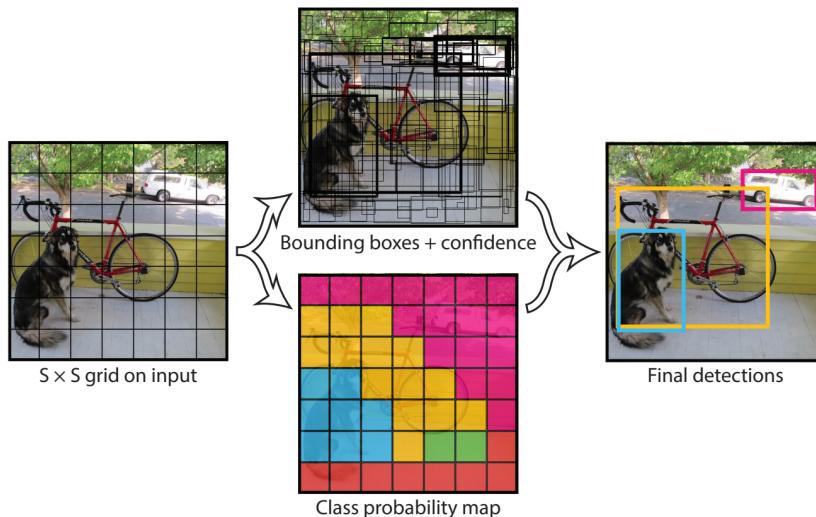


Figura 5.2. Esquema del funcionamiento de YOLO (Fuente: publicación original de YOLO [Red+15])

Este método presenta numerosas ventajas frente a R-CNN y Faster R-CNN, en primer lugar, destaca por su rapidez. La primera red diseñada por los desarrolladores de YOLO cuenta con 24 capas de convolución y era capaz de correr a 45 imágenes por segundo (gracias a avances en computación actualmente es capaz de correr con una mayor tasa de refresco). Esta red también se caracteriza por tener una estructura más propia de las Fully Convolutional Neuronal Network (FCNN), es decir, se caracteriza por ser mayoritariamente convolucional. Otra de sus grandes ventajas, es que al mirar a la imagen entera, es capaz de ver las diferentes conexiones entre *bounding boxes* y determinar de forma rápida y precisa la posición del objeto. Esto no ocurre en métodos como la ventana flotante. A continuación, se muestra de forma breve y gráfica el funcionamiento de YOLO en la Figura 5.2

5.1.1. Estructura

Debido al gran interés y las capacidades de YOLO, este ha sido desarrollado bastante en los últimos años. En la actualidad se han publicado cinco versiones de YOLO que mejoran su rendimiento y le dotan de más capacidades. Estas nuevas versiones han traído consigo cambios en la estructura de la red principal de YOLO para mejorar sus capacidades y rendimiento. A la vez se han desarrollado variantes de la red principal de diferentes tamaños con el fin de poder implantar estos sistemas en sistemas con capacidades computacionales menores. Para el desarrollo de este proyecto nos basaremos en la última versión disponible, la versión

cinco que trae consigo un total de diez modelos de diferentes tamaños y resoluciones (ver Tabla 5.2).

La primera red neuronal del sistema de visión artificial se basará en el modelo YOLOv5l que se caracteriza por mantener una buena relación entre potencia y capacidades. Este será el modelo capaz de distinguir e identificar todas las piezas y es por ello que requiere de una mayor capacidad. Se puede observar la estructura interna de la red en Apéndice A.

Model	Size (pixels)	mAP^{val} 0.5:0.95	mAP^{val} 0.5	Speed V100 b1 (ms)	Speed V100 b32 (ms)	Params (M)	FLOPs @640 (B)
YOLOv5n	640	28.0	45.7	6.3	0.6	1.9	4.5
YOLOv5s	640	37.4	56.8	6.4	0.9	7.2	16.5
YOLOv5m	640	45.4	64.1	8.2	1.7	21.2	49.0
YOLOv5l	640	49.0	67.3	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	12.1	4.8	86.7	205.7
YOLOv5n6	1280	36.0	54.4	8.1	2.1	3.2	4.6
YOLOv5s6	1280	44.8	63.7	8.2	3.6	12.6	16.8
YOLOv5m6	1280	51.3	69.3	11.1	6.8	35.7	50.0
YOLOv5l6	1280	53.7	71.3	15.8	10.5	76.8	111.4
YOLOv5x6 + TTA	1280 + 1536	55.0 55.8	72.7 72.7	26.2 -	19.4 -	140.7 -	209.8 -

Tabla 5.2. Modelos de YOLOv5

5.1.2. Entrenamiento

Una de las mejoras introducidas en YOLOv5 es la evolución de los hiperparámetros. Este nuevo sistema permite ejecutar numerosos entrenamientos bajo diferentes hiperparámetros con el objetivo de optimizar estos de forma automática. Para el desarrollo de esta red se ha empleado esta nueva función de forma que se ha podido reducir notablemente el tiempo de entrenamiento gracias a la reducción del proceso de aprendizaje en base a prueba y error.

Con el fin de obtener los mejores resultados posibles se ha optado por emplear el proceso evolutivo de YOLO. Se trata de un proceso iterativo en el que en base a una función objetivo se van seleccionando los hiperparámetros de entrenamiento con el fin de optimizar el entrenamiento. Estos nuevos hiperparámetros se obtienen por medio de un algoritmo genético, de forma que los mejores resultados de la anterior iteración se emplearán como base para la creación de nuevos hiperparámetros. La función objetivo empleada se caracteriza por ser una combinación de $mAP@0.5$ con una contribución del 10 % y $mAP@0.5 : 0.95$ con una contribución del 90 %.

```

1 def fitness(x):
2     # Model fitness as a weighted combination of metrics
3     w = [0.0, 0.0, 0.1, 0.9]  # weights for [P, R, mAP@0.5, mAP@0.5:0.95]
4     return (x[:, :4] * w).sum(1)

```

Se recomienda realizar un mínimo de 300 iteraciones para poder obtener unos resultados óptimos. Sin embargo, debido a límites de tiempo y potencia se ha optado por reducir a 100 iteraciones y cada una con un total de 10 epochs. En la Tabla 5.3 se muestran los hiperparámetros empleados durante todo el proceso.

Opciones de entrenamiento		
Parametro	Base	Evolución
Optimizador	SGD momentum/Adam betal	
Initial learn rate	0.01	0.00769
Learn rate decay	0.01	0.01464
Momentum	0.937	0.98
Weight decay	0.0005	0.00059
Warmup epochs	3.0	2.961
Warmup momentum	0.8	0.85061
Box loss gain	0.05	0.05199
CLS loss gain	0.5	0.4489
IoU threshold	0.2	0.2
Anchor threshold	4	4.6172
Epochs	100	100
Batch Size	16	16

Tabla 5.3. Opciones de entrenamiento de YOLO

5.1.3. Resultados

Con la ayuda de YOLO se ha podido realizar un último análisis del *dataset* empleado con el fin de determinar las instancias, la ubicación y forma de los diferentes objetos presentes. Y determinar si existe alguna correlación entre estos. Analizando los resultados lo primero que se puede observar es la notable diferencia de instancias de piezas G1 y G3 respecto al resto de piezas pequeñas. El motivo principal de semejante diferencia es falta de tiempo y de potencia computacional para renderizar un mayor número de imágenes. Las pruebas iniciales del proyecto se realizaron con dichas piezas y es por ello que se dispone de un mayor número de instancias.

Analizando la posición de las piezas se observa que se cubre la totalidad del área de trabajo pero se observa un claro patrón cuadrático. Esto es debido a que los escenarios empleados presentan forma de cajas de tres tamaños, lo cual queda reflejado en la distribución de las piezas. Se observa también formas cuadradas dispersas por el interior de las cajas, estas son una adición al escenario con el fin de evitar que las piezas queden paralelas a la superficie de trabajo y aumentar así la riqueza del *dataset*.

Si se analiza la forma de todas las instancias se comprueba que un elevado grado de dispersión pero siguiendo una ligera distribución lineal de forma que existe correlación entre el ancho y el alto de las instancias. Esto se debe a que todas las piezas empleadas presentan una forma rectangular con lados semejantes. Por último, se puede observar el tamaño de las instancias dividido en alto y ancho. Se puede observar en ambos histogramas la presencia de dos categorías de piezas, piezas grandes y piezas pequeñas. Las piezas pequeñas se muestran al inicio de ambos histogramas y presentan todos un tamaño muy similar. Por otro lado, las piezas grandes se encuentran distribuidas por el resto del histograma y presentan una mayor varianza.

Como se ha mencionado en la sección de entrenamiento, se ha empleado un algoritmo genético para la estimación de los hiperparámetros con el fin de mejorar los resultados de la red. Es por ello que en los resultados se comparará dos redes. Ambas han sido entrenadas el mismo número de *epochs* pero con diferentes hiperparámetros. Se compararán los hiperparámetros

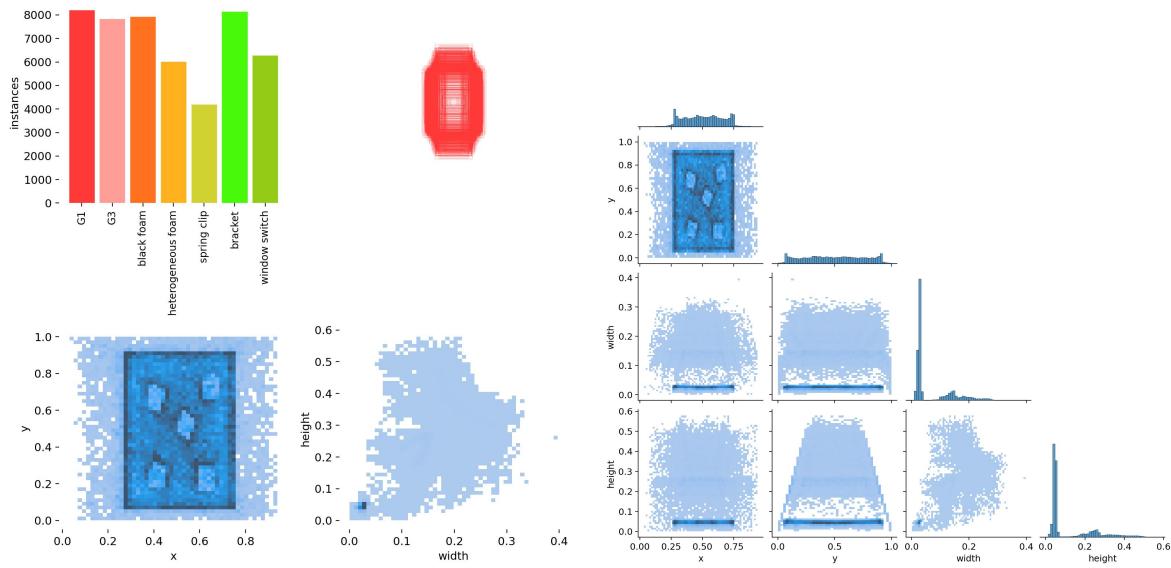


Figura 5.3. Evaluación de instancias, forma y ubicación del *dataset* empleado en YOLO

definidos por defecto por los desarrolladores de YOLO y los definidos por el algoritmo evolutivo.

Los resultados del proceso evolutivo se dividen en dos secciones, en primer lugar se muestra en la Figura 5.4 el proceso de evolución de los hiperparámetros y como estos han ido cambiando entre las diferentes evoluciones. Y en segundo lugar, se muestra en la Figura 5.5 una comparativa de los resultados de cada red entrenada durante el proceso evolutivo. Analizando estas dos gráficas se puede llegar a la conclusión de que el sistema no ha sido capaz de optimizar los hiperparámetros debido a la aparente aleatoriedad de la distribución de los datos. Sin embargo, al comparar ambos modelos se observa grandes mejoras respecto al caso base tanto en entrenamiento como en validación. Es probable que la aparente aleatoriedad de los resultados se deba a una falta de *epochs*. Con un número tan reducido es difícil hacer asunciones sobre los hiperparámetros. Si se desea obtener los resultados óptimos también se debe de considerar un aumento del número de iteraciones.

Una vez completado el proceso evolutivo se han entrenado las dos versiones de YOLO con diferentes hiperparámetros. En ambos casos se observa como la red neuronal es capaz de generalizar y extraer la información y características de las piezas. Pero también se observa la falta de entrenamiento ya que las métricas siguen mejorando cuando se detuvo el entrenamiento. Pero a pesar de ello se puede observar el gran rendimiento y capacidades de YOLO. También se observa una clara mejora al emplear los hiperparámetros obtenidos por el algoritmo evolutivo tanto en entrenamiento como en validación. Por último, una vez realizado el entrenamiento se ha analizado una nueva sección del *dataset* que no ha sido empleada para entrenamiento ni para validación durante el entrenamiento. Con este nuevo *dataset* se han obtenido unos resultados prometedores (ver Figura 5.9) caracterizados por una tasa de fallos reducida y una precisión elevada para niveles de confianza del 50 %. Los resultados son prometedores pero esto no implica que sean perfectos. La red presenta una clara ligera tendencia a dar falsos positivos a la hora de detectar algunas de las piezas. Esto se debe a varios factores:

- Texturas: al emplear una librería de texturas no se tiene un control total sobre estas. Algunas de las texturas empleadas son de colores oscuros lo que dificulta en gran medida

la detección de las piezas negras. Las piezas grandes no se ven tan afectadas por la presencia de mayor relieve.

- Tamaño: Las piezas pequeñas se encuentran a una distancia elevada de la cámara y en ocasiones el área de la pieza se puede medir en decenas de píxeles.

Se recomienda replantear la configuración del generador de imágenes para las piezas pequeñas. Como primera medida se debe de reducir la distancia a las piezas para obtener mayor detalle de las piezas. Si es posible se recomienda no modificar las texturas para evitar sesgar el *dataset*.

Por último, se ha obtenido la precisión y el *recall* de validación (ver Figura 5.10). Con la precisión se observa como con un nivel de confianza del 40 % se pueden obtener resultados con una precisión elevada. Y con *recall* se observa como ha medida que aumentamos el nivel de confianza requerido aumenta el numero de falsos negativos (piezas no detectadas). En la curva F1 se enfrentan ambas métricas permitiendo ver de forma visual como afecta el nivel de confianza a la identificación de piezas. Se observa como el modelo final debe de trabajar con un nivel de confianza entre el 40 % y 80 % para obtener resultados óptimos.

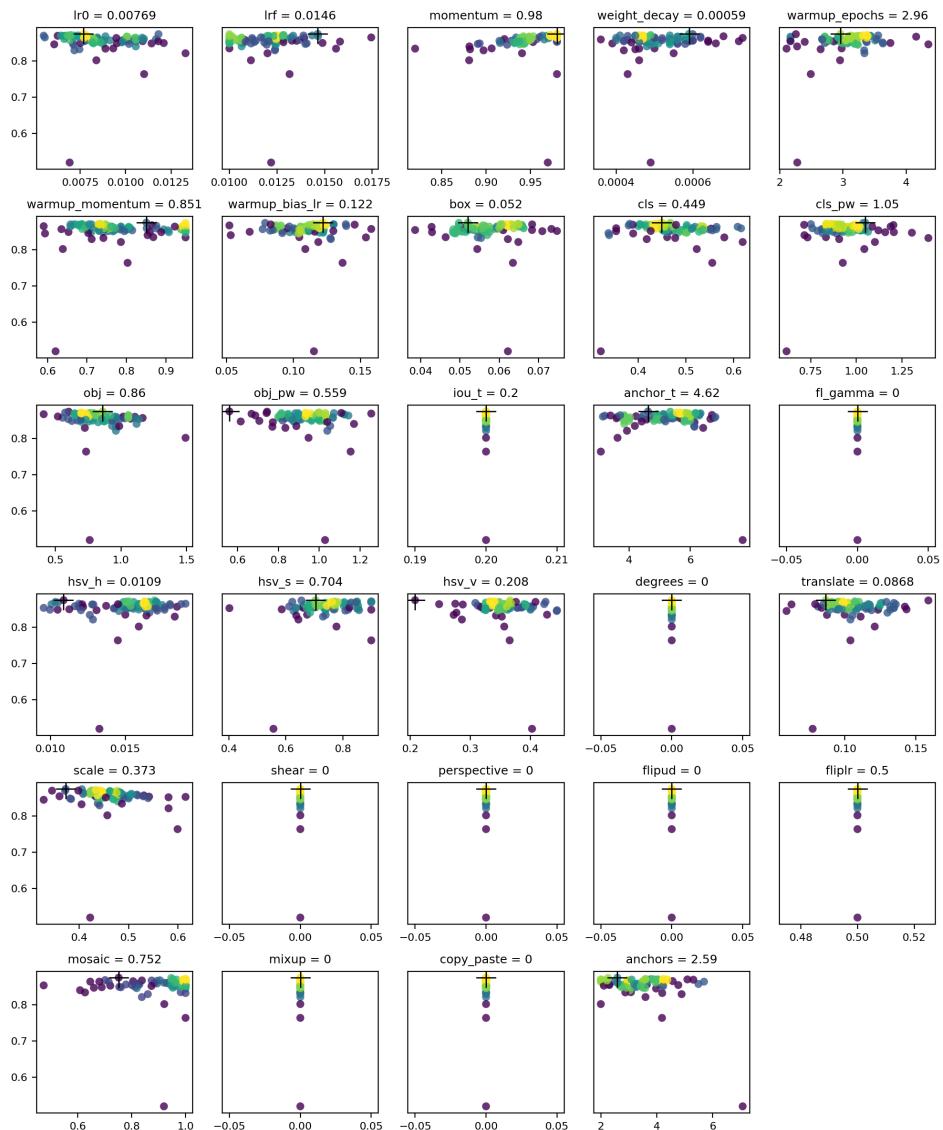


Figura 5.4. Proceso de evolución de los hiperparámetros de YOLO

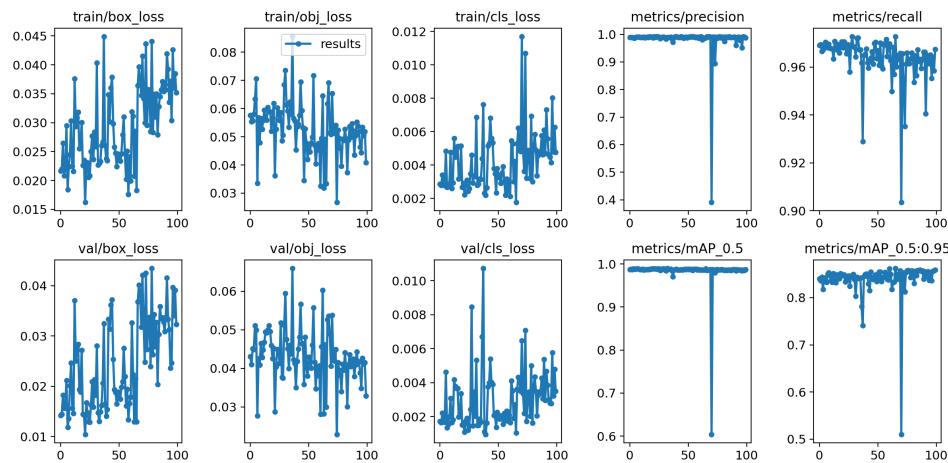


Figura 5.5. Resultados de la evolución de los hiperparámetros de YOLO

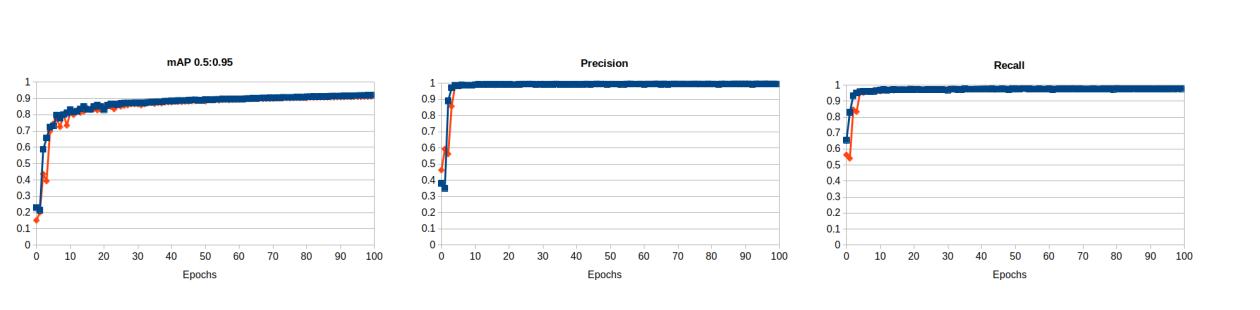


Figura 5.6. Métricas de la red neuronal YOLO (azul evolutivo y rojo base)



Figura 5.7. Entrenamiento de la red neuronal YOLO (azul evolutivo y rojo base)

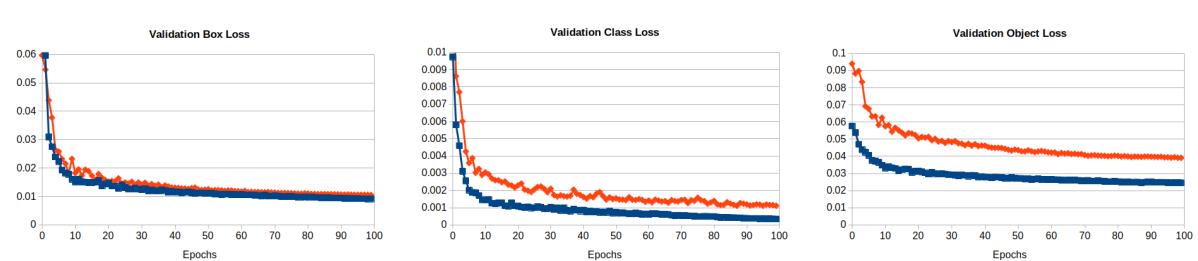


Figura 5.8. Validación de la red neuronal YOLO (azul evolutivo y rojo base)

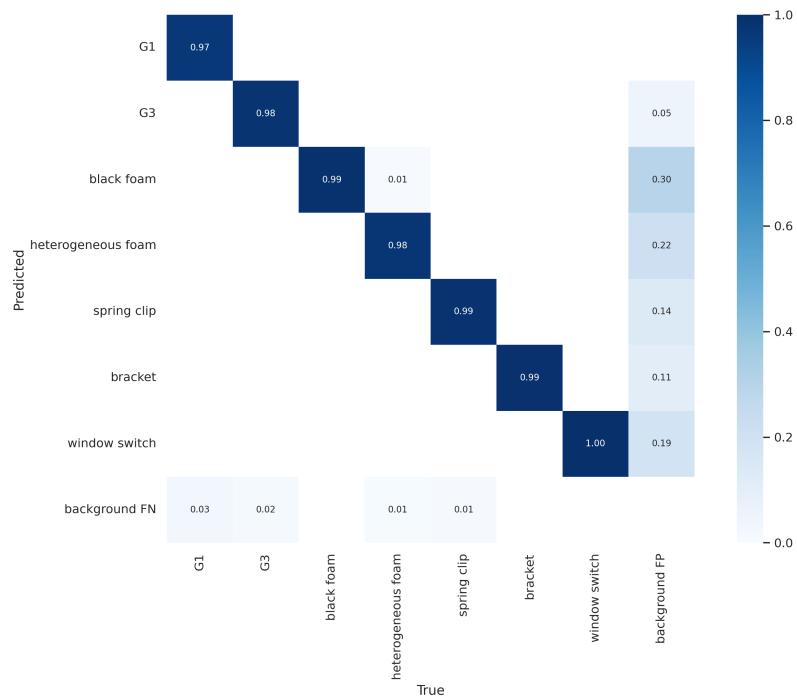


Figura 5.9. Matriz de confusión en validación de YOLO

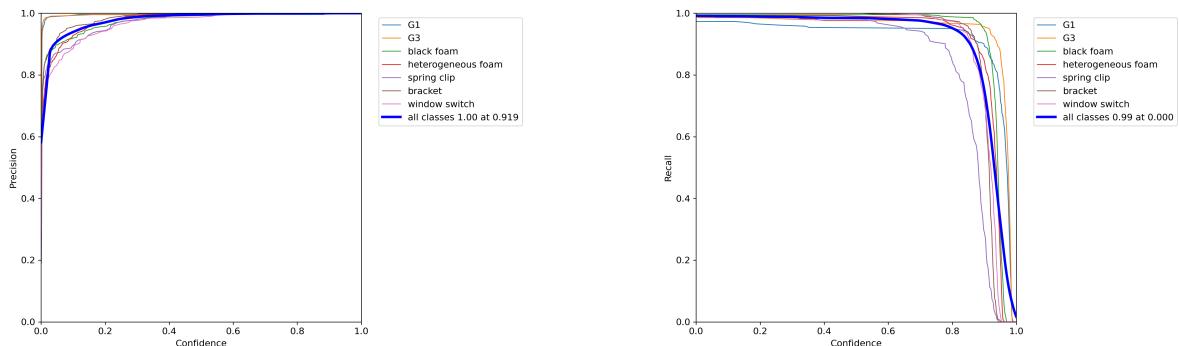


Figura 5.10. Precision y Recall en validación de YOLO

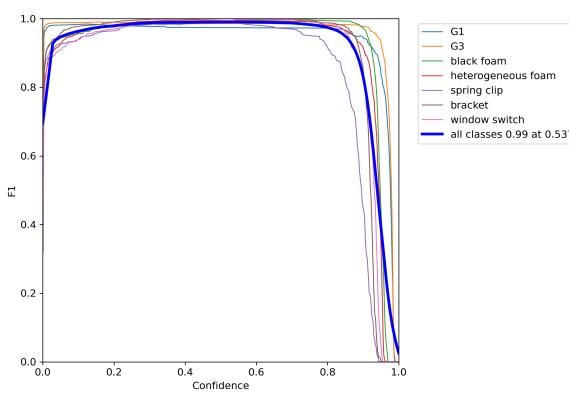


Figura 5.11. Curva F1 en validación de YOLO

5.2. Tiny YOLO

Con el fin de determinar los posibles puntos de agarre disponibles en la pieza se ha desarrollado una segunda red neuronal basado en YOLO capaz de detectar dichas regiones. Empleará como entrada un recorte de la pieza detectada por la red neuronal principal y sobre está identificará y detectará aquellos puntos presentes y accesibles. Se emplea una segunda red neuronal debido a que YOLO puede presentar problemas al intentar detectar objetos dentro de objetos debido la proximidad de los *bounding boxes*. Se deberá de crear y entrenar una red por cada pieza que se desee introducir en el sistema. Esta solución implica un aumento de complejidad al tener que disponer de un gran número de redes pero a su vez permite la creación de un sistema modular capaz de adaptarse a una nueva pieza con rapidez y sin afectar al resto del sistema.

5.2.1. Estructura

Esta segunda red neuronal se deberá de entrenar tantas veces como piezas de desee introducir al sistema. Es por ello que la carga computacional debe de ser reducida para evitar crear un sistema lento y poco eficiente. Por suerte, debido a la gran especialización de la red, solo debe de detectar regiones para un tipo de pieza, que nos podemos permitir emplear la versión de YOLO de menos tamaño. La principal ventaja de YOLOv5 es el uso de recursos tanto a nivel computacional como de almacenamiento. La red ocupa 4 MB y cuenta con un total de 1.9 millones de neuronas. Pero a pesar de su reducido tamaño es capaz de lograr resultados sorprendentes capaces de competir con potentes y pesadas redes basadas en Fast-RCNN.

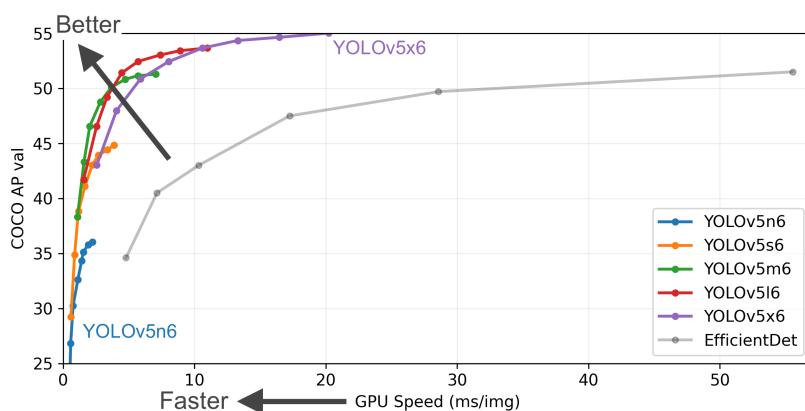


Figura 5.12. Comparativa de los diferentes modelos de YOLO

5.2.2. Entrenamiento

Como se ha visto anteriormente, una de las mejoras introducidas en YOLOv5 consiste en la optimización de los hiperparámetros de forma autónoma empleando un algoritmo genético. Sin embargo, este tipo de entrenamiento aumenta significativamente la carga computacional necesaria ya que se debe de entrenar múltiples veces la misma red para determinar la configuración óptima. En el caso de la red principal este es un gasto asumible ya que solo se deberá de entrenar una red. Para la red secundaria se debe de entrenar un modelo para cada pieza. Es por ello que con el fin de reducir los tiempos de entrenamiento se empleará los hiperparámetros obtenidos por el proceso evolutivo de la red YOLO.

Opciones de entrenamiento	
Parametro	Valor
Optimizador	SGD momentum/Adam betal
Initial learn rate	0.00769
Learn rate decay	0.01464
Momentum	0.98
Weight decay	0.00059
Warmup epochs	2.961
Warmup momentum	0.85061
Box loss gain	0.05199
CLS loss gain	0.4489
IoU threshold	0.2
Anchor threshold	4.6172
Epochs	250
Batch Size	96

Tabla 5.4. Opciones de entrenamiento de TINY YOLO

5.2.3. Resultados

Tiny YOLO debe de ser entrenado tantas veces como piezas grandes haya presentes en el sistema. Por lo cual se ha tenido que entrenar múltiples veces y se dispone de numerosas redes y resultados. pero con el fin de simplificar este documento y para evitar redundancia solo se va a mostrar el proceso de entrenamiento y los resultados de una pieza. La pieza seleccionada es G1 y consta de un total de 5 posibles puntos de agarre.

Con la ayuda de YOLO se ha podido realizar un último análisis del dataset empleado con el fin de determinar las instancias, la ubicación y forma de los diferentes puntos presentes. Y determinar si existe alguna correlación entre estos y patrones en la distribución de piezas. Al analizar los resultados mostrados en la Figura 5.13 se observa a simple vista múltiples patrones tanto en distribución como en forma de las piezas a detectar. Para entender estos patrones es necesario entender que se está intentando detectar, se desea identificar regiones con puntos de agarre dentro de una pieza y en este caso nos estamos centrando en la pieza G1. Estas regiones se han definido con la ayuda de códigos QR basados en la tecnología Aruco implantados dentro de la geometría 3D de la pieza. Esto implica que en todas las piezas los códigos se encuentran siempre en las mismas posiciones y dimensiones relativas a la pieza. Por ello se pueden observar los siguientes patrones:

- Instancias: Se observa una mayor número ed instancias del punto 5. Esto se debe al comportamiento de la pieza G1 y su tendencia a acabar en posiciones determinadas.
- Posición: se observa un patrón con forma de estrella que representa las zonas con un mayor número de instancias. Esta geometría se debe al elevado tamaño e la pieza G1 respecto a la zona de trabajo. Debido a las dimensiones esta pieza presenta posiciones con una alta ocurrencia.
- Tamaño: al enfrentar el ancho y alto de las instancias destaca una clara proporción 1:1 entre estas. Esto se debe a que los códigos QR empleados son cuadrados.

Se recomiendo replantear la configuración del generador de imágenes para adaptar el entorno de trabajo a la pieza con el objetivo de evitar la presencia de posiciones frecuentes.

Esto se puede realizar con la inclusión de nuevos entornos de trabajo que añadan riqueza y aleatoriedad.

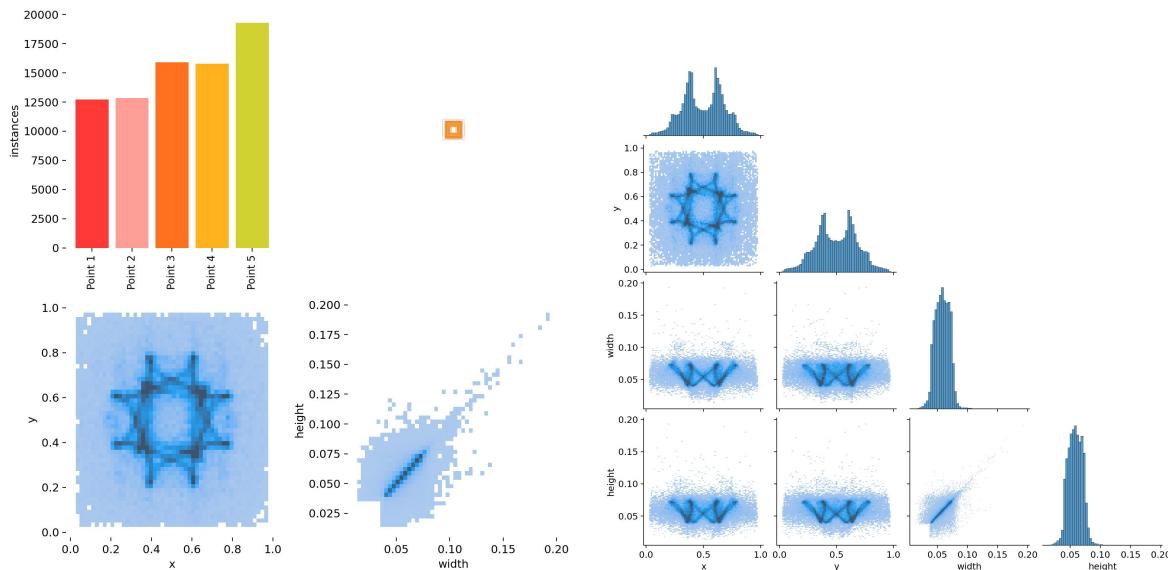


Figura 5.13. Evaluación de instancias, forma y ubicación del *dataset* basado en G1 empleado en TINY YOLO

Tras entrenar durante un total de 250 *epochs* se puede observar que la red neuronal ha conseguido aprender, extraer y generalizar la información disponible en el *dataset*. Con lo que se ha podido reducir drásticamente las perdidas (*object*, *class* & *box*) tanto en entrenamiento como en validación. En este entrenamiento se puede apreciar que a pesar de ser un caso más simple, con una red de menor tamaño y un mayor número de *epochs*, la red todavía presenta un margen de mejora. El error en validación todavía presenta una tendencia decreciente cuando se detuvo el entrenamiento. El motivo de no continuar con el entrenamiento son limitaciones de tiempo y carga computacional.

Por último, se ha obtenido la *precision* y el *recall* de validación (ver Figura 5.18). Con la *precision* se observa como con un nivel de confianza del 80 % se pueden obtener resultados con una *precision* elevada y similares a la red YOLO con un nivel de confianza del 40 %. Esta reducción de *precision* frente a YOLO se puede deber a varios factores:

- Red: al emplear la versión yolov5n se reduce de forma considerable las capacidades de la red.
- Tipo de pieza a detectar: en este caso se desea detectar regiones de interés dentro de una pieza. Es decir, estas no presentan un borde claro que se pueda distinguir frente al fondo. Esto puede afectar en gran medida a la red que se ve incapaz de determinar los límites del objeto a detectar.
- Tamaño de la pieza: todas las instancias empleadas en este entrenamiento han sido diseñadas por ordenador y presentan unas dimensiones exactas. Esto puede sesgar las capacidades del sistema y puede llevarlo a realizar asunciones.

Y con *recall* se observa como ha medida que aumentamos el nivel de confianza requerido aumenta el número de falsos negativos (piezas no detectadas). En la curva F1 se enfrentan ambas métricas permitiendo ver de forma visual como afecta el nivel de confianza a la identificación

de piezas. Se observa como el modelo final debe de trabajar con un nivel de confianza entre el 40 % y 80 % para obtener resultados óptimos.

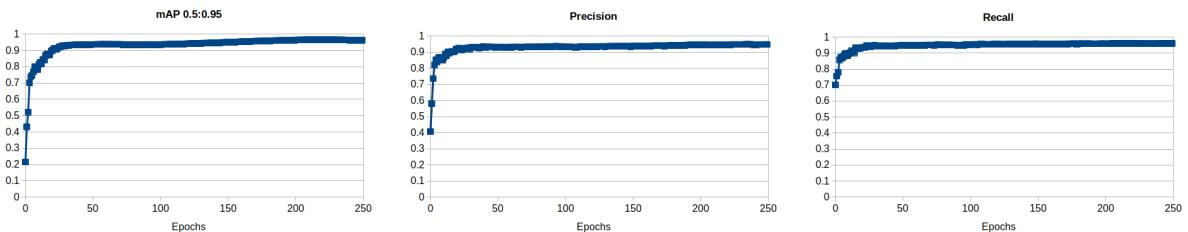


Figura 5.14. Métricas de la red neuronal TINY YOLO (azul evolutivo y rojo base)

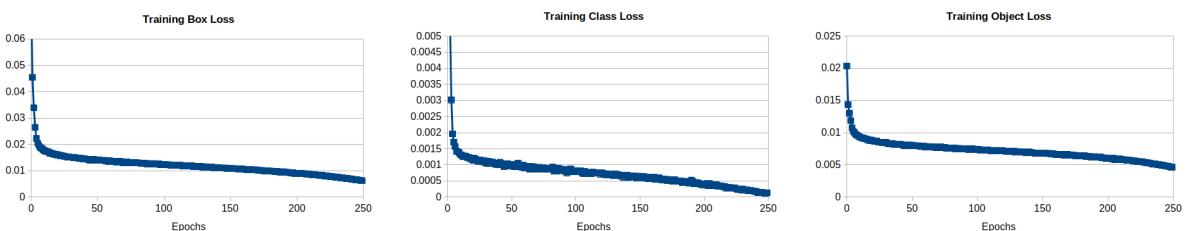


Figura 5.15. Entrenamiento de la red neuronal TINY YOLO (azul evolutivo y rojo base)

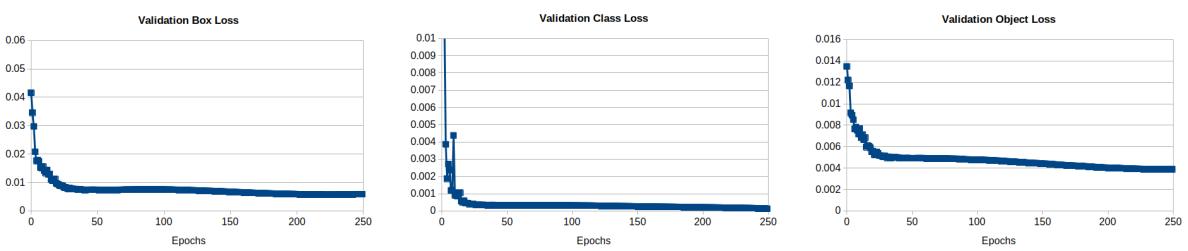


Figura 5.16. Validación de la red neuronal TINY YOLO (azul evolutivo y rojo base)

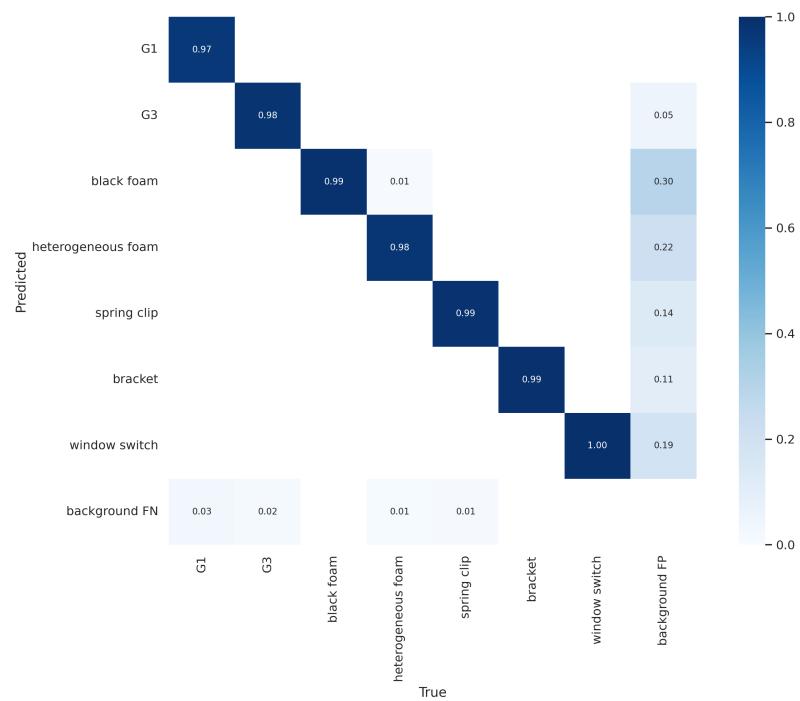


Figura 5.17. Matriz de confusión en validación de TINY YOLO

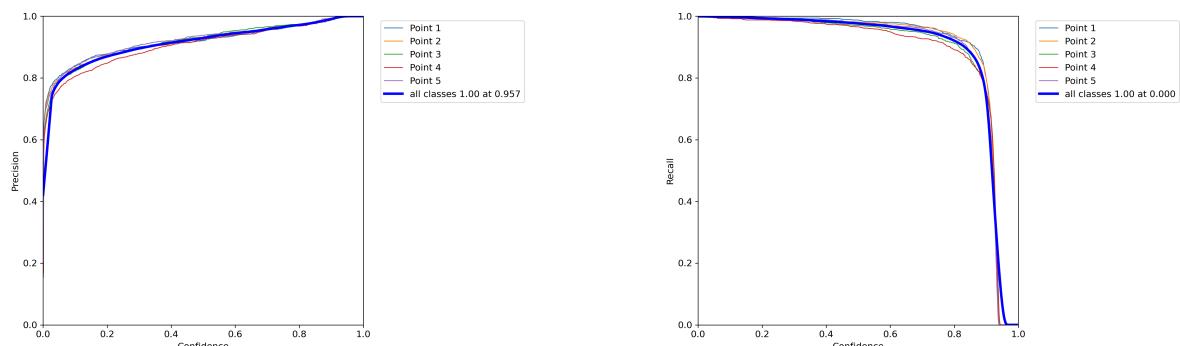


Figura 5.18. Precision y Recall en validación de TINY YOLO

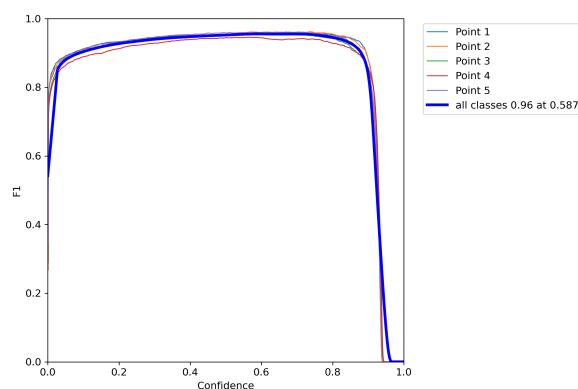


Figura 5.19. Curva F1 en validación de TINY YOLO

5.3. Regresor

Los últimos elementos necesarios para poder agarrar la pieza deseada es la determinación del punto de agarre óptimo y el vector normal a dicho punto. Gracias a la red neuronal TINY YOLO es posible determinar el punto óptimo pero se todavía se desconoce con precisión sus coordenadas. Por ello se ha introducido una tercera red neuronal encargada de determinar las cinco coordenadas necesarias (x, y, u, v y w).

Para simplificar el proceso lo máximo posible, esta última capa se entrenará de forma individual para cada posible punto de agarre. Esto implica que se dispondrán de tantos pesos como puntos de agarre. Esta red empleará como entrada un recorte de la pieza detectada por la red neuronal principal. Esta solución implica un aumento de complejidad al tener que disponer de un gran número de redes pero a su vez permite la creación de un sistema modular capaz de adaptarse a un cambio en la pieza con rapidez y sin afectar al resto del sistema. Además se permite reducir la carga sobre la red al simplificar el proceso lo máximo posible.

5.3.1. Estructura

Esta tercera red neuronal se deberá de entrenar tantas veces como puntos de desee introducir al sistema. Es por ello que la carga computacional debe de ser reducida para evitar crear un sistema lento y poco eficiente. Pero tampoco se desea crear un sistema incapaz de extraer las características de la pieza y determinar los puntos de agarre. Teniendo esto en cuenta se ha desarrollado una red neuronal dividida en dos secciones, una primera sección basada en la convolución y encargada de extraer las características de la pieza. Y una segunda sección del tipo *fully connected* basada en perceptrones que se encargarán de analizar las características extraídas por las primeras capas. En base a estas estimarán las coordenadas del punto de agarre.

El desarrollo de una red neuronal es un proceso laborioso que requiere de múltiples pruebas e intentos con el fin de conseguir desarrollar un sistema que aprenda y de forma optimizada. Es por ello por lo que se han tenido que desarrollar y entrenar múltiples redes para dar con un sistema óptimo. El primer paso en el desarrollo de la red ha sido comparar el impacto de diferentes tamaños de redes. Se han comparado redes con tres, cinco y siete capas convolucionales para determinar el número de capas necesarias para poder extraer las características. Como se verá en las secciones de entrenamientos y resultados, de estas pruebas se ha determinado que un número de capas óptimo en base a la fase de entrenamiento es cinco. Sin embargo, estos resultados varían drásticamente cuando se analiza los resultados de validación. En esta última fase se observa mejores resultados con la red neuronal del el máximo número de capas de convolución. Se puede plantear un aumento del número de capas para mejorar los resultados pero esto a su vez implica un aumento de carga computacional. Por ello se ha optado por fijar el número de capas convolucionales en siete. Las capas están constituidas por un conjunto de capa convolucional seguida por una capa de normalización y finalmente una capa de activación tipo SiLu.

A la vez que se comprobó el impacto del número de capas de convolución, se comprobó el impacto del tamaño del regresor. Y se obtuvieron resultados similares a las capas de convolución. Se desarrollaron redes con dos, tres y cuatro redes de convolución y se obtuvo que mejores resultados con un sistema constituido por cinco capas del tipo *fully connected*.

Una vez establecido el número de capas se determinó el impacto de diferentes funciones de activación (ReLU frente a SiLu) y se comparó diferentes criterios para la determinación del error

(L1Loss frente a MSELoss) y diferentes optimizadores (Adam frente a SGD). De todas estas pruebas (visibles en el apartado de entrenamiento) se determinó que los mejores resultados se obtuvieron con la capa de activación SiLu, el criterio L1Loss y el optimizador Adam.

Finalmente se comprobó el impacto de una capa del tipo *Dropout* debido a que se observaba una gran diferencia de resultados entre entrenamiento y validación. Se comprobó el impacto de una capa entre el proceso de convolución y el regresor. Y también se comprobó el impacto de una capa *Dropout* dentro del regresor. En ambos casos se entrenó para diferentes valores de *Dropout* (0.1, 0.3 y 0.5) y en los seis entrenamientos se obtuvo peores resultados frente a la red sin capa *Dropout*. Es por ello por lo que se excluyó de la red.

Tras analizar todos los resultados se determinó que la tercera red del proceso en un sistema constituido por siete capas de convolución y cuatro del tipo *fully connected*. Estas capas a su vez son activadas por la función SiLu y no se empleará ninguna capa del tipo *Dropout*.

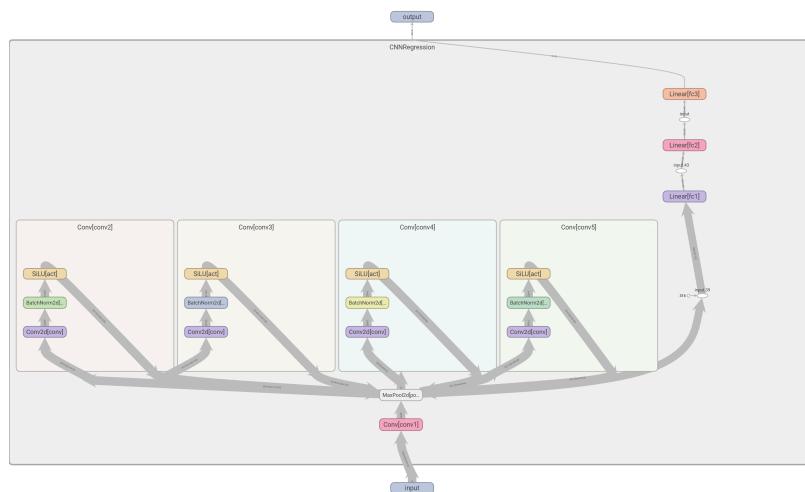


Figura 5.20. Estructura del regresor

5.3.2. Entrenamiento

Debido al gran número de veces que se deberá de entrenar esta red neuronal, tantas como puntos de agarre por cada pieza. Se ha fijado como objetivo reducir la carga computacional en la medida de lo posible. Esto implica que el tamaño de la red debe de ser reducido y el número de *epochs* debe de ser limitado a 100. Con este objetivo en mente se ha desarrollado la siguiente configuración para el entrenamiento.

Opciones de entrenamiento	
Parametro	Valor
Optimizador	Adam
Criterio	L1loss
Initial learn rate	1e-04
Learn rate decay	0.5
Learn rate step	15
Epochs	100
Batch Size	32

Tabla 5.5. Opciones de entrenamiento de Regresor

5.3.3. Resultados

El regresor debe de ser tantas veces como puntos en cada pieza grande haya. Es por esto que la red y el entrenamiento se han visto limitados. A pesar de esta limitación se han conseguido resultados que demuestran las capacidades del sistema a la hora de determinar los puntos de agarre. En estos resultados se muestra el impacto de las diferentes tipos de redes desarrolladas.

Destaca con los mejores resultados la red con el mayor número de capas de convolución y sin capas *Dropout*. Y con este tipo de red se consigue obtener un error entorno al 5 % en validación para los vectores normales y entorno al 1-2 % para la determinación del punto de agarre. También destaca en esta red los errores de validación de los vectores normales donde a pesar del entrenamiento se dan situaciones en donde los resultados obtenidos empeoran al aumentar el entrenamiento. Esto se debe a una combinación de sobreaprendizaje junto un la utilización de un *dataset* sesgado. Esto conlleva a la red a asumir las posiciones de los vectores normales y es por ello por lo que aumenta el error total a pesar del entrenamiento. Se intentó solventar el problema con la utilización de una capa *dropout* pero esta afectó negativamente a rendimiento del sistema ya que el problema reside en una mayor medida en el *dataset*. Si se desean mejorar los resultados se debe de evitar el sesgado antes de intentar mejorar la red.

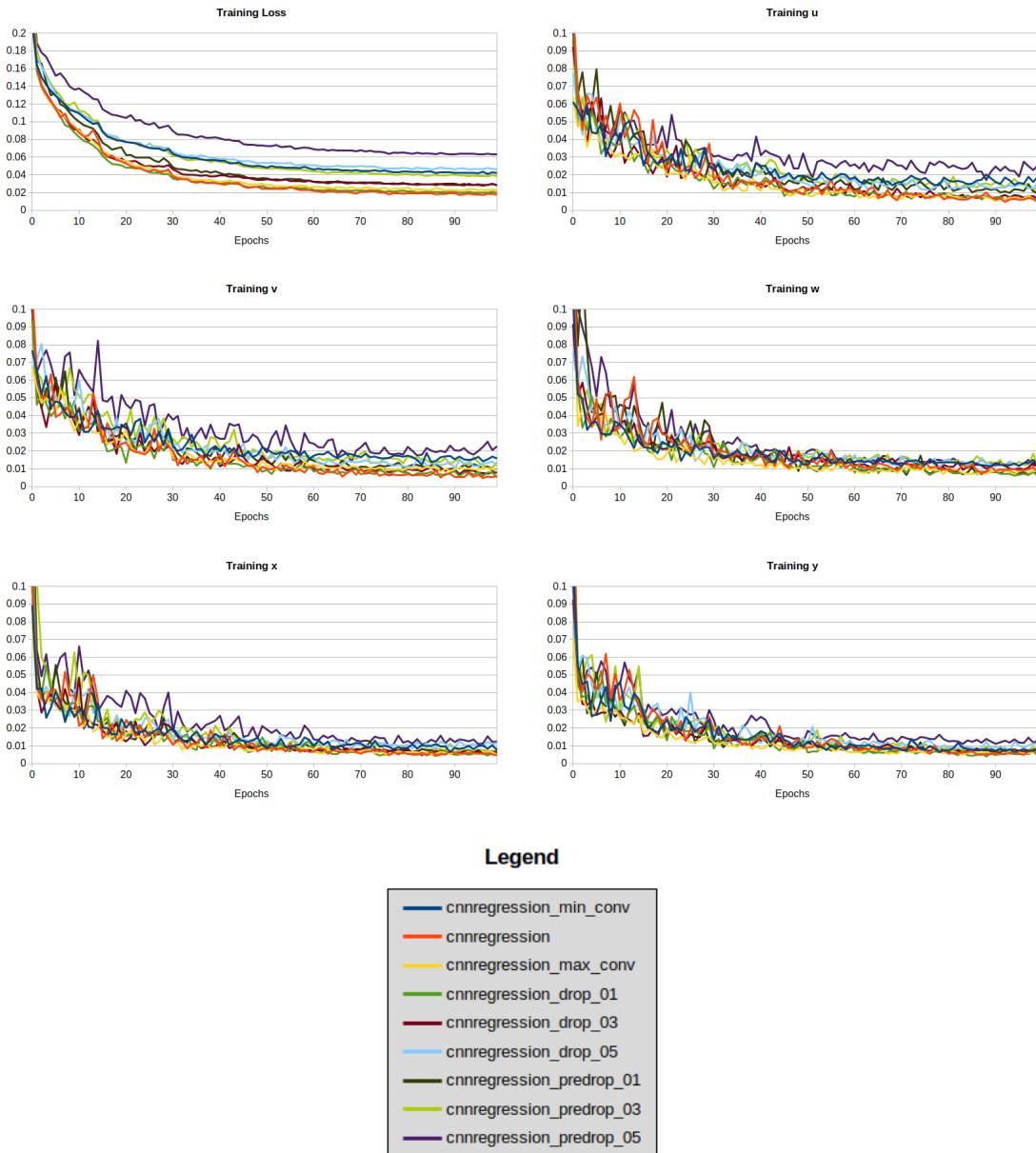


Figura 5.21. Comparativa de las fases de entrenamiento del regresor

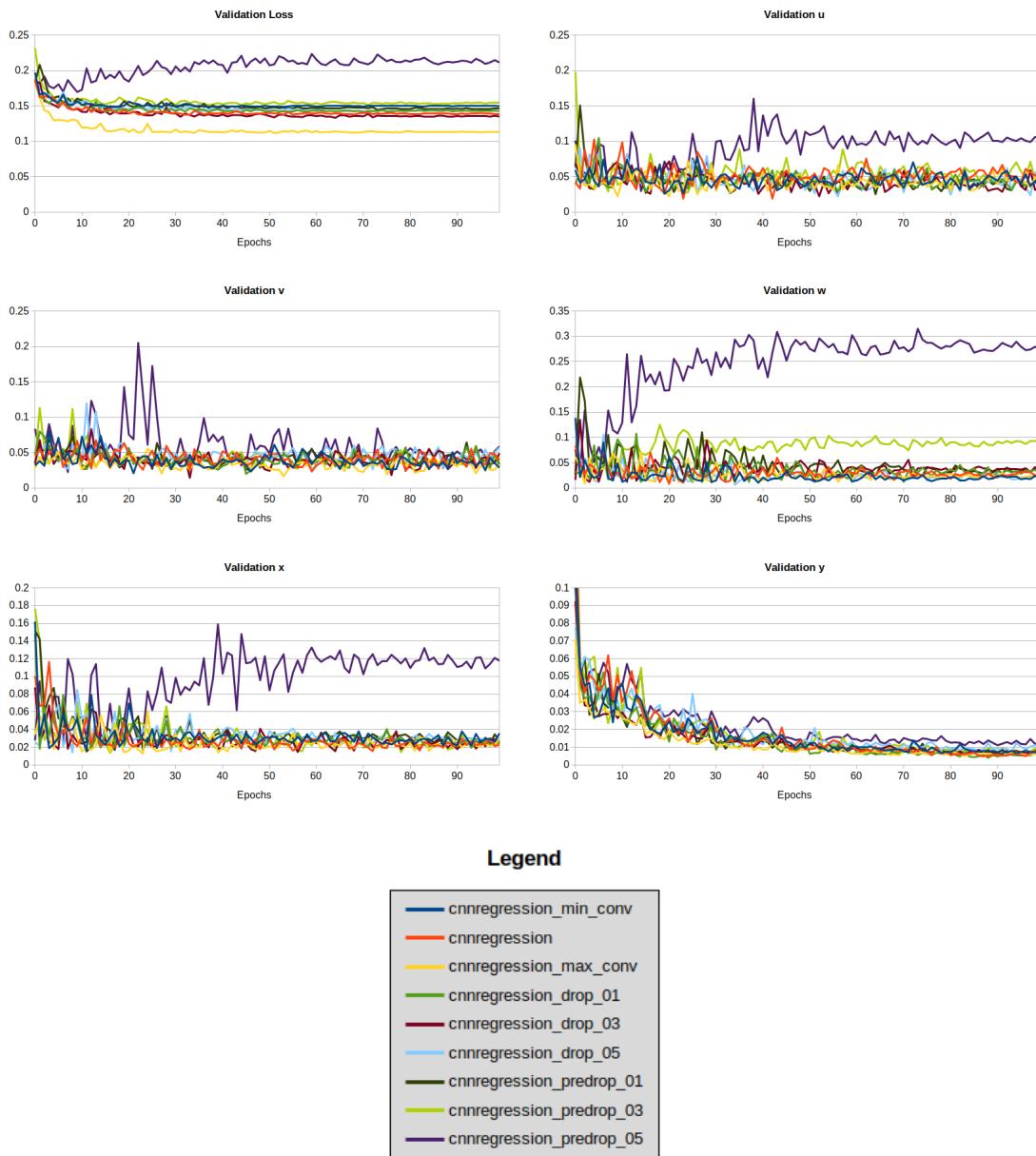


Figura 5.22. Comparativa de las fases de validación del regresor

6

Conclusiones y futuros desarrollos

El desarrollo de este proyecto se caracteriza por cumplir dos grandes objetivos/metas. En primer lugar, su desarrollo ha permitido la creación de un sistema basado en redes neuronales capaz de identificar piezas y sus puntos de agarre óptimos. Y en segundo lugar ha permitido comprobar las capacidades de aprendizaje de una red neuronal al emplear un *dataset* sintético. Es por ello que se debe de analizar el sistema desde las dos perspectivas.

6.1. Sistema de detección de puntos de agarre

El objetivo final del proyecto es la identificación de puntos de agarre de forma que las piezas puedan ser manipuladas con un robot industrial con el fin de automatizar una linea de ensamblado. La linea de ensamblaje se caracteriza por emplear una elevada diversidad de piezas que presentan diferentes formas y tamaños. Es debido a esta elevada diversidad de piezas que se ha tenido que desarrollar un sistema modular capaz de adaptarse a las diferentes tipos de piezas. El nuevo sistema consta de diferentes módulos/redes que se activan u desactivan en función del tipo de pieza que se deseé manipular.

Gracias a este sistema se ha conseguido obtener el punto de agarre óptimo para las piezas grandes y se ha asumido el centro de las piezas pequeñas como un buen punto de agarre. En base a los resultados obtenidos en la fase de validación, el sistema se muestra capaz con errores pequeños y pocos *outliers*. Desgraciadamente el comportamiento de las redes solo se ha podido analizar frente a un *dataset* sintético pero no frente a una situación real. Para determinar con precisión el alcance y la capacidad del sistema se requiere de un *dataset* real.

Pero esta falta de un *dataset* real no impide que se pueda analizar la estructura del sistema y se planteen puntos de mejora. La red presenta grandes puntos fuertes como su modularidad. La cual permite la inclusión de nuevas piezas sin afectar al rendimiento de las ya implementadas. Y sin la necesidad de entrenar toda la red frente a un *dataset* con todas las piezas. Esto permite reducir los tiempos de entrenamiento de forma sustancial durante el proceso de inclusión de nuevas piezas. Pero a su vez afecta al rendimiento global del sistema ya que implica que se debe de analizar y extraer las características de la pieza varias veces durante su análisis. Por ello se plantea la posibilidad del desarrollo de una única red capaz de aprovechar la

extracción de características durante todo el proceso. Este tipo de red aumenta la complejidad del sistema pero permitirá mejorar la eficiencia. Y las últimas capas del sistema deberán de ser intercambiadas dependiendo del tipo de pieza que se vaya a identificar. De esta forma se mantiene la modularidad del sistema pero sin perder la eficiencia propia de estos sistemas.

6.2. Dataset sintético

Para el desarrollo del sistema de identificación de puntos de agarre se ha tenido que desarrollar un *dataset* sintético que refleje la realidad y permita aumentar el número de imágenes para el entrenamiento. Este *dataset* se ha realizado con blender y con la ayuda de blenderproc de forma que se ha conseguido generar miles de imágenes fotorealistas que representan y reflejan la zona de trabajo sobre la que se implantará el sistema.

Durante el desarrollo de este sistema se presta especial atención en representar lo mejor posible la realidad y el entorno de trabajo. De esta forma varios de los escenarios y de las configuraciones usadas se limitan a representar los escenarios observados en la planta de ensamblaje. Esta decisión desgraciadamente ha llevado la generación de un *dataset* sesgado en donde la mayoría de las piezas se encuentran en una posición predefinida por el escenario. Esto ha llevado a que la red neuronal tienda a asumir que las piezas se encuentran en una de esas posiciones predefinidas y empeore los resultados del sistema.

Es por ello por lo que se recomienda mejorar la aleatoriedad del sistema introduciendo más escenarios que a pesar de que no reflejen la realidad si permitan la obtención de imágenes con mayor riqueza y eviten el sesgado de la red. Este tipo de escenarios se deben de caracterizar por no presentar un suelo plano sino un suelo con irregularidades que fuercen a las piezas a nuevas posiciones.

7

Objetivos de desarrollo sostenible

Análisis comparativo del proyecto frente a los objetivos de desarrollo sostenible. Desde un punto de visto social, ecológico y económico

El 25 de septiembre de 2015, los líderes mundiales adoptaron un conjunto de objetivos globales para erradicar la pobreza, proteger el planeta y asegurar la prosperidad para todos como parte de una nueva agenda de desarrollo sostenible. Cada objetivo tiene metas específicas que deben alcanzarse en los próximos 15 años.



Figura 7.1. Objetivos de desarrollo sostenible aprobados en 2015

En total se han planteado 17 objetivos para llevar a cabo, este proyecto se enmarca dentro de tres de ellos y a continuación se van a desarrollar de forma individual.

- Objetivo 8 - Trabajo decente y crecimiento económico: El proceso de automatización de la industria supone la creación de nuevos puestos de trabajo más especializados y con mejores condiciones laborales. Implica un crecimiento de la producción sin un aumento notable de costes y por lo tanto implica un desarrollo económico. Pero para ello es importante controlar este desarrollo y evitar el sobre uso de estas máquinas y con ello el reemplazo del ser humano en el sector industrial.
- Objetivo 9 - Industria, innovación e infraestructura: Gracias al empleo de las nuevas tecnologías para la automatización de los procesos industriales se ha logrado un crecimiento y desarrollo económico en la industria. Una de las metas de este objetivo es la globalización de estas tecnologías y el facilitar el acceso a estas para los países en vías de desarrollo. Con este proyecto se ha desmontado y creado un sistema reentrenable y usable en el proceso de automatización y esto sin suponer un gran coste de desarrollo. La tecnología desarrollada en este proyecto es de acceso público y modificable para implantar en diferentes sistemas.
- Objetivo 12 - Producción y consumo responsable: Con la implantación de los robots en la industria no solo se obtiene un incremento en la producción. También, se obtiene una reducción en el número de errores cometidos durante el proceso de fabricación. Esto implica que menos recursos deben de ser usados en la producción. Además, los robots son capaces de desarrollar tareas imposibles para un ser humano y con diferentes materiales. Gracias a sus capacidades, se pueden desarrollar productos más complejos o innovadores centrados en mejorar la sostenibilidad y con un empleo más ecológico de los recursos.

Es por todo esto que se considera que este proyecto puede conllevar mejoras para la sociedad a nivel económico, ecológico y social. Siempre y cuando la implantación de estos sistemas se realice correctamente y teniendo en cuenta a la mano de obra humana ya existente. Estos sistemas no deben de reemplazar sino ayudar a este sector.

A

Estructura YOLOv5

Debido al gran interés y las capacidades de YOLO, este ha sido desarrollado bastante en los últimos años. En la actualidad se han publicado cinco versiones de YOLO que mejoran su rendimiento y le dotan de más capacidades. Estas nuevas versiones han traído consigo cambios en la estructura de principal de YOLO para mejorar sus capacidades y rendimiento. A la vez se han desarrollado variantes de la red principal de diferentes tamaños con el fin de poder implantar estos sistemas en sistemas con capacidades computacionales menores. Para el desarrollo de este proyecto nos basaremos en la última versión disponible, la versión cinco que trae consigo un total de diez modelos de diferentes tamaños y resoluciones (ver Tabla 5.2).

Esta nueva versión trae consigo grandes cambios en la arquitectura empleada la cual se puede dividir en tres secciones:

- Columna vertebral: Se basa en CSP-Darknet53. Una estructura diseñada con el objetivo de ser empleada para la detección de objetos. Y cuya principal característica es el uso de una estrategia de división y unión que permite mejorar el flujo del gradiente a lo largo de la red.
- Cuello: Se emplea la nueva función SPPF en lugar de SPP la reduce los tiempos de ejecución a más de la mitad. Se ha desarrollado también un nuevo CSP-PAN.
- Cabeza: Se vuelve a emplear la cabeza desarrollada en YOLOv3.

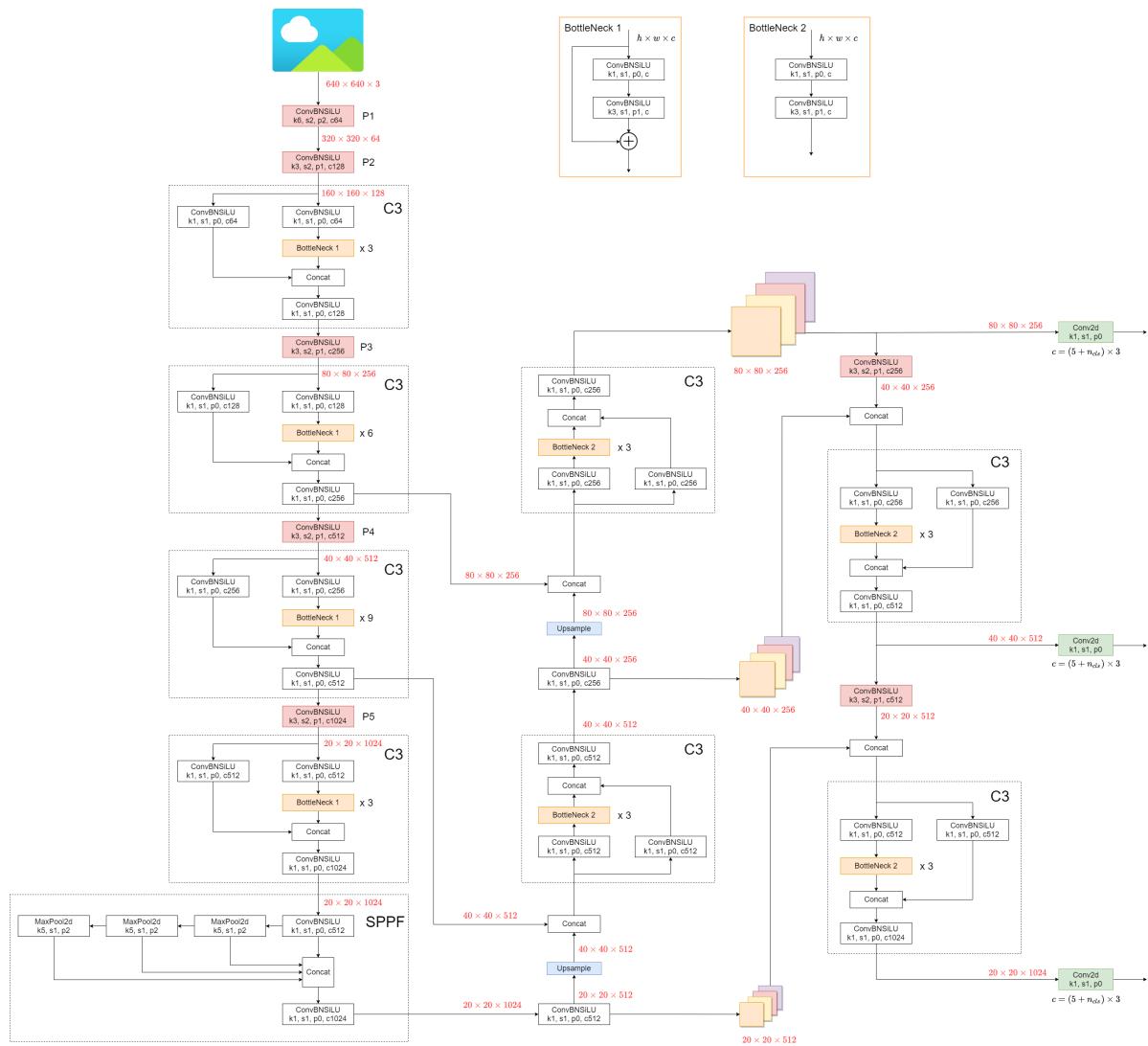


Figura A.1. Estructura de YOLOv51

B

Camara RGBD

En este anexo se desarrollará la conexión con la cámara RGBD Intel realsense L515 y las funcionalidades del driver desarrollado.

Para el desarrollo de este proyecto se ha decidido emplear la cámara Intel Realsense L515 [Inta] ya que dispone de todos los sensores necesarios, buenas prestaciones y un reducido coste. Para controlar la cámara se ha desarrollado un *driver* para permitir la conexión directa a través de Python y el *wrapper* desarrollado por Intel.

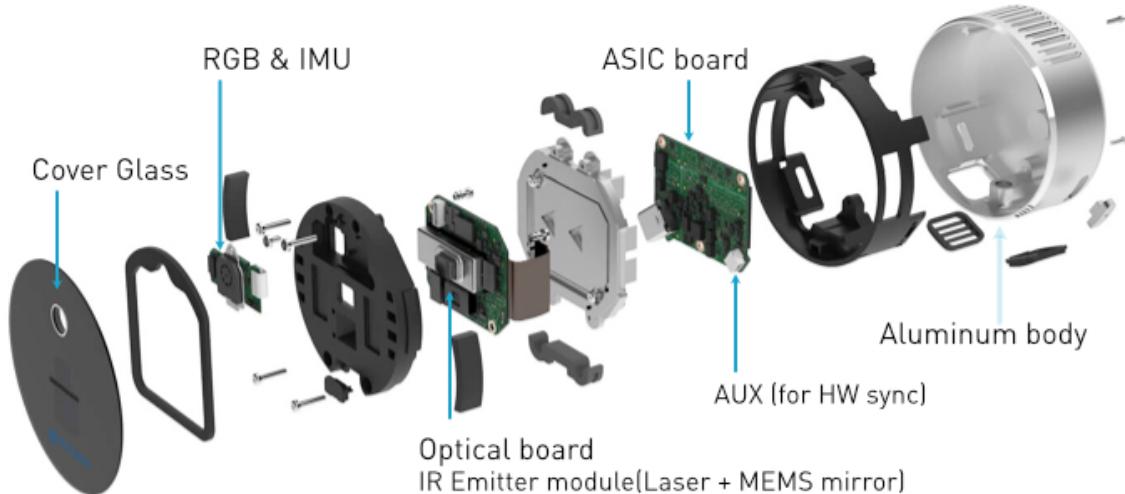


Figura B.1. Cámara Intel Realsense L515 [Inta]

Para la conexión con la cámara se ha desarrollado una clase en Python de forma que el usuario pueda acceder a todas las funciones necesarias de forma cómoda y simple sin necesidad de entender como funciona internamente. La clase se encarga de establecer la conexión con la cámara y al crear nuestra clase en base al *wrapper* desarrollado por Intel, nos permite tener acceso a todas las configuraciones y capacidades de la cámara y basarnos en estas para desarrollar una clase más simple e intuitiva de cara al usuario.

El *driver* se ha desarrollado con varias funcionalidades en mente:

- Abstracción de la cámara: se desea que durante el desarrollo de las diferentes partes del proyecto no se deba de tener en cuenta la cámara y la configuración de la misma.
- Independencia del *hardware*: al emplear el mismo *driver* en todas las partes del proyecto, se puede plantear en un futuro un cambio de cámara y el impacto en el proyecto será mínimo.
- Calibración: el *driver* debe de ser capaz de extraer todo el potencial de la cámara. Esto implica que se debe de añadir la funcionalidad de calibración tanto de las imágenes de color como de profundidad.
- Alineación: La cámara L515 esta a su vez constituida por diferentes sensores con diferentes parámetros y configuraciones. Con el objetivo de poder relacionar las diferentes imágenes se debe de poder alinear correctamente todas las imágenes.

Uno de los puntos más importantes al desarrollar el *driver* ha sido la capacidad de calibración. Al tratarse de diferentes sensores estos deben de estar bien calibrados con sus parámetros intrísecos bien definidos pero a su vez también se deben de definir la relación entre estos sensores con los parámetros extrínsecos. Para la calibración de la cámara de color se pueden emplear varios métodos pero se ha decidido optar por uno de los más extendidos basado en la detección de patrones (tablero de ajedrez) con una distancia entre patrones predefinida. Por el contrario, la imagen de profundidad ha sido un mayor obstáculo debido al tipo de tecnología empleada. Al detectar distancias no es capaz de detectar el patrón y por lo tanto en una primera instancia no se puede emplear este método. Este obstáculo se ha conseguido solucionar empleando la imagen de infrarrojos. Esta se obtiene a través del mismo sensor (ambos emplean el sensor *LiDAR*) y es capaz de distinguir el patrón.



Figura B.2. Comparativa de imágenes de color, infrarrojos y profundidad con Realsense L515

Tras solucionar los problemas de calibración se ha podido desarrollar el *driver* que tomará el control de la cámara y permitirá abstraer el resto del proyecto del *hardware*. A continuación se muestra la funcionalidad del *driver* desarrollado:

- Constructor: Se ha desarrollado un constructor personalizado que permite establecer la conexión con la cámara y establecer las opciones de funcionamiento.
- Destructor: El destructor se ha desarrollado para que durante el proceso de destrucción de la instancia primero se corte de forma gradual la conexión con la cámara. De esta forma se evita futuros errores al intentar reconectarse a la cámara. Si no se realiza una desconexión progresiva no se podrá volver a abrir una conexión con la cámara y será necesario desconectar la cámara o reiniciar el *driver*.

- `setCalibration`: Carga y aplica los parámetros definidos por el proceso de calibración.
- `getColour`: función para la captura solo de la imagen de color. Devuelve la imagen de color con una resolución de 1920x1080x3 píxeles
- `getDepth`: función para la captura solo de la imagen de profundidad. Devuelve la imagen de con o sin alineamiento con la de color y con una resolución de 1024x768 píxeles.
- `getIR`: función para la captura solo de la imagen de infrarrojos. Devuelve la imagen de con o sin alineamiento con la de color y con una resolución de 1024x768 píxeles.
- `getImages`: Combina las anteriores funciones para obtener la salida de todos los sensores con una sola llamada.
- `saveImages`: captura y guarda las imágenes obtenidas por todos los sensores.
- `transformRGBToDepth`: relaciona puntos de la imagen de color con la imagen de profundidad (imagen no alineada).
- `getPosition`: relaciona un punto en pixeles con sus coordenadas en el mundo real.

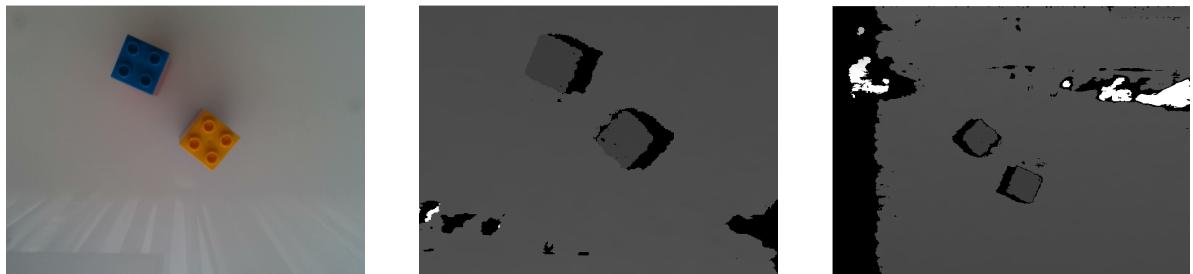


Figura B.3. Alineamiento de las imágenes de color y profundidad tomadas con la cámara Realsense L515

Bibliografía

- [ABB] ABB. «IRB 1400 YuMi», dirección: <https://new.abb.com/products/robotics/es/robots-colaborativos/yumi> (visitado 16-11-2021).
- [Aca] K. Academy. «El gradiente», dirección: <https://es.khanacademy.org/math/multivariable-calculus/multivariable-derivatives/partial-derivative-and-gradient-articles/a/the-gradient> (visitado 18-06-2020).
- [Alo+18] M. Z. Alom y col., *The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches*, 2018. arXiv: [1803.01164 \[cs.CV\]](https://arxiv.org/abs/1803.01164).
- [AMS18] I. Ahmad, I. Moon y S. J. Shin, «Color-to-grayscale algorithms effect on edge detection — A comparative study», en *2018 International Conference on Electronics, Information, and Communication (ICEIC)*, 2018, págs. 1-4.
- [AT13] A. Andreopoulos y J. Tsotsos, «50 Years of object recognition: Directions forward», *Computer Vision and Image Understanding*, vol. 117, págs. 827-891, ago. de 2013.
- [AVA20] AVA. «ArUco: a minimal library for Augmented Reality applications based on OpenCV». (2020), dirección: <https://www.uco.es/investiga/grupos/ava/node/26>.
- [BHM20] J. Burnham, J. Hardy y K. Meadors, «Comparison of the Roberts, Sobel, Robinson, Canny, and Hough Image Detection Algorithms», jun. de 2020.
- [Cas+13] R. Casanelles y col., «Towards high performance robotic solutions in press automation - An ABB view», en *IEEE ISR 2013*, 2013, págs. 1-3.
- [Den+19] M. Denninger y col., «BlenderProc», *arXiv preprint arXiv:1911.01911*, 2019.
- [DH72] R. O. Duda y P. E. Hart, «Use of the Hough Transformation to Detect Lines and Curves in Pictures», *Commun. ACM*, vol. 15, n.º 1, págs. 11-15, ene. de 1972. dirección: <https://doi.org/10.1145/361237.361242>.
- [FB81] M. A. Fischler y R. C. Bolles, «Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography», *Commun. ACM*, vol. 24, págs. 381-395, 1981.
- [GSA16] F. Giménez Palomares, J. Serrá y E. Alemany, «Aplicación de la convolución de matrices al filtrado de imágenes», *Modelling in Science Education and Learning*, vol. 9, pág. 97, ene. de 2016.
- [IK88] J. Illingworth y J. Kittler, «A Survey of the Hough Transform», *Comput. Vision Graph. Image Process.*, vol. 44, n.º 1, págs. 87-116, ago. de 1988. dirección: [https://doi.org/10.1016/S0734-189X\(88\)80033-1](https://doi.org/10.1016/S0734-189X(88)80033-1).
- [Inta] Intel. «Realsense L515», dirección: <https://www.intelrealsense.com/lidar-camera-l515/> (visitado 10-11-2021).
- [Intb] Intel. «SDK for Intel Realsense», dirección: <https://github.com/IntelRealSense/librealsense/releases> (visitado 10-11-2021).

Bibliografía

- [KSH17] A. Krizhevsky, I. Sutskever y G. E. Hinton, «ImageNet Classification with Deep Convolutional Neural Networks», *Commun. ACM*, vol. 60, n.º 6, págs. 84-90, mayo de 2017. dirección: <https://doi.org/10.1145/3065386>.
- [Lab] H. Labs. «History (1961-1980)». Castellano.
- [Lec+89] Y. Lecun y col., «Backpropagation Applied to Handwritten Zip Code Recognition», *Neural Computation*, vol. 1, págs. 541-551, dic. de 1989.
- [LR90] M. E. Lee y R. A. Redner, «A note on the use of nonlinear filtering in computer graphics», *IEEE Computer Graphics and Applications*, vol. 10, n.º 3, págs. 23-29, 1990.
- [Mor+17] D. Morrison y col., *Cartman: The low-cost Cartesian Manipulator that won the Amazon Robotics Challenge*, 2017. arXiv: [1709.06283 \[cs.RO\]](https://arxiv.org/abs/1709.06283).
- [Oll05] A. Ollero Baturone, *ROBOTICA. Manipuladores y Robots Móviles*, Castellano, S. MARCOMBO, ed. 2005, 464 págs.
- [Pér15] E. Pérez-López, «Los sistemas SCADA en la automatización industrial», *Revista Tecnología en Marcha*, vol. 28, págs. 3-14, dic. de 2015.
- [Red+15] J. Redmon, S. Divvala, R. Girshick y A. Farhadi, *You Only Look Once: Unified, Real-Time Object Detection*, 2015. dirección: <https://arxiv.org/abs/1506.02640>.
- [Sze22] R. Szeliski, *Computer Vision: Algorithms and Applications* (Texts in Computer Science). Springer International Publishing, 2022. dirección: <https://books.google.ie/books?id=A34ZygEACAAJ>.
- [Vid19] A. Vidhya. «Brief History of Neural Networks». (2019), dirección: <https://medium.com/analytics-vidhya/brief-history-of-neural-networks-44c2bf72eec>.