



**GRADO EN INGENIERÍA EN TECNOLOGÍAS
INDUSTRIALES**

TRABAJO FIN DE GRADO

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

Autor: Ignacio Ortiz de Zúñiga Mingot

Directores:

Jaime Boal Martín-Larrauri

José Antonio Rodríguez Mondéjar

Madrid

Junio de 2020

Copyright © 2020 Ignacio Ortiz de Zúñiga Mingot

Este trabajo fue escrito con $\text{\LaTeX}$ y compilado en $\text{\TeX}$ maker usando la distribución $\text{\TeX}$ -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 Microsoft Visio[®] y MATLAB[®].

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título "Optimización del sistema de visión artificial de un robot industrial para una aplicación de Pick and Place" en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2019-2020 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.: Ignacio Ortiz de Zúñiga Mingot

Fecha: 20/07/2020

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO



Firmado digitalmente por BOAL

MARTIN LARRAURI JAIME -

05304600H

Fecha: 2020.07.20 12:04:27 +02'00'

Fdo.: Jaime Boal Martín-Larrauri

Fecha:/...../.....

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Firmado por RODRIGUEZ MONDEJAR JOSE ANTONIO
- 29057313X el día 20/07/2020 con un
certificado emitido por AC FNMT Usuarios

Fdo.: José Antonio Rodríguez Mondéjar

Fecha:/...../.....



**GRADO EN INGENIERÍA EN TECNOLOGÍAS
INDUSTRIALES**

TRABAJO FIN DE GRADO

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

Autor: Ignacio Ortiz de Zúñiga Mingot

Directores:

Jaime Boal Martín-Larrauri

José Antonio Rodríguez Mondéjar

Madrid

Junio de 2020

Copyright © 2020 Ignacio Ortiz de Zúñiga Mingot

Este trabajo fue escrito con $\text{\LaTeX}$ y compilado en $\text{\TeX}$ maker usando la distribución $\text{\TeX}$ -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 Microsoft Visio[®] y MATLAB[®].

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

Agradecimientos

En primer lugar me gustaría darle las gracias a Jaime por haberme ayudado y guiado tanto a pesar de las circunstancias. Por haber estado siempre dispuesto a reunirnos, por aclararme las ideas y por tener tanta paciencia. Y gracias por introducirme al mundo del *Deep learning*, gracias a ti y este proyecto he descubierto una rama de la electrónica fascinante y apasionante que me ha dejado con ganas de más.

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. Y en especial, gracias a mis amigos y equipo de laboratorio, Nora y Fidel por hacerme reír y distraerme durante las largas horas de clase y las duras prácticas de laboratorio. 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 agradecerse 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
1.1. Motivación	2
1.2. Objetivos	2
1.3. Herramientas	3
1.4. Arquitectura del sistema	3
1.5. Organización del documento	4
2. Estado del arte	7
2.1. Procesado de la imagen	7
2.1.1. Filtros, detección de borde y detección de formas	7
2.1.2. Redes Neuronales Convolucionales	12
2.2. Robótica Industrial	14
3. Segmentación con mascarar de color	17
3.1. Filtrado de color	18
3.2. Identificación y mejora de la imagen	20
3.3. Extracción de características	23
3.4. Resultados	23
4. Clasificación con redes neuronales	27
4.1. Condiciones de entrenamiento	27
4.2. LEGONet	28
4.2.1. Estructura	29
4.2.2. Entrenamiento	30
4.2.3. Resultados	30
4.3. LEGO16	33
4.3.1. Estructura	33
4.3.2. Entrenamiento	34
4.3.3. Resultados	35
5. Segmentación con redes neuronales	37
5.1. Preparativos para el entrenamiento y evaluación	38
5.1.1. Entrenamiento	38
5.1.2. Evaluación	39
5.2. R-CNN	40

5.2.1. R-CNN basado en LEGONet	41
5.2.2. R-CNN basado en LEGO16	45
5.3. Faster R-CNN	45
5.3.1. Faster R-CNN basado en LEGONet	47
5.3.2. Faster R-CNN basado en LEGO16	51
5.4. YOLO	55
5.4.1. YOLO basado en LEGONet	57
5.4.2. YOLO basado en LEGO16	60
6. Extracción de atributos	65
6.1. Análisis de profundidad	65
6.2. Análisis de la orientación	67
6.2.1. Detección de borde y transformada de Hough	68
6.2.2. Modelo de regresión con redes neuronales	72
7. Análisis de los resultados	79
7.1. Análisis de la segmentación	79
7.1.1. Precisión, Exhaustividad, Tasa de fallos y FPPI	79
7.1.2. Velocidad	81
7.2. Análisis del cálculo de la orientación	82
7.2.1. Precisión	82
7.2.2. Velocidad	84
8. Conclusiones	85
9. Futuros desarrollos	87
A. Comunicación con la cámara	89
B. Interfaz gráfica de usuario	91
B.1. Interfaz	91
B.2. Controles	92
B.3. Medidas de seguridad	93
B.4. Problemas causados por la interfaz	93
C. Objetivos de desarrollo sostenible	95
Bibliografía	97

Índice de figuras

Figura	1. Esquema de la arquitectura del sistema	XVI
Figura	2. Comparativa de los métodos de segmentación al detectar piezas rojas . . .	XVIII
Figura	3. Diagrama de cajas: Comparación de la velocidad de diferentes métodos para la segmentación (más arriba es mejor)	XIX
Figura	4. Diagrama de cajas: Comparación de la precisión de diferentes métodos para el cálculo de la orientación (cuanto más centrado respecto al cero y menos disperso mejor)	XX
Figure	5. Architecture of the system	XXII
Figure	6. Comparison of the different segmentation methods to detect red LEGO pieces	XXIV
Figure	7. Box diagrams: comparison of the speed of different object detectors (Higher is better)	XXV
Figure	8. Box diagram: Comparison of the precision of different methods for calculating the orientation of pieces (the closer to the zero the better) . . .	XXV
Figura	1.1. Esquema de la arquitectura del sistema	4
Figura	1.2. Diagrama de las etapas del sistema	5
Figura	2.1. Proceso de convolución en matrices	8
Figura	2.2. Comparativa de algoritmos de detección de borde	9
Figura	2.3. Detección de borde por algoritmo de Canny	10
Figura	2.4. Representación de la transforma de Hough	11
Figura	2.5. Estructura de AlexNet	13
Figura	2.6. Errores cometidos en ImageNet en los últimos años	14
Figura	2.7. Brazo robótico IRB120	15
Figura	3.1. Error por mala calibración del filtrado por color	18
Figura	3.2. Filtrado por color	19
Figura	3.3. Segmentación de las piezas a analizar	21
Figura	3.4. Escalado a gris de las piezas segmentadas	21
Figura	3.5. Primer paso: ajuste de intensidad	21
Figura	3.6. Segundo paso: Resalto de pequeños detalles	21
Figura	3.7. Tercer paso: aplicación del algoritmo de Canny	22
Figura	3.8. Cuarto paso: dilatación y contracción de los bordes	22
Figura	3.9. Quinto paso: Inversión de los colores de la pieza	22
Figura	3.10. Muestra de imágenes para la evaluación del proceso de segmentación . . .	23
Figura	3.11. Estudio de la segmentación por color al detectar piezas amarillas	24
Figura	3.12. Estudio de la segmentación por color al detectar piezas rojas	24
Figura	3.13. Estudio de la segmentación por color al detectar piezas azules	25

Figura 3.14. Error en la evaluación de segmentación por color	25
Figura 4.1. Muestra de imágenes para el entrenamiento de clasificadores	28
Figura 4.2. Funcionamiento de <i>Relu</i> (Rectified Linear Unit)	29
Figura 4.3. Estructura de LEGONet	30
Figura 4.4. Activación primera convolución de LEGONet	30
Figura 4.5. Evaluación de LEGONet: matriz de confusión	32
Figura 4.6. Evaluación de LEGONet: curvas <i>ROC</i>	32
Figura 4.7. Estructura de LEGO16	33
Figura 4.8. Activación de la primera convolución de LEGO16	34
Figura 4.9. Evaluación de LEGO16: matriz de confusión	36
Figura 4.10. Evaluación de LEGO16: curvas <i>ROC</i>	36
Figura 5.1. Imágenes empleadas para el entrenamiento de segmentación con redes neuronales	38
Figura 5.2. Aumento del número de imágenes para el entrenamiento	39
Figura 5.3. Muestra de imágenes para la evaluación de redes neuronales	40
Figura 5.4. Demostración del concepto de ventana flotante (Fuente: [Tsa])	40
Figura 5.5. Estimación de las regiones de interés con R-CNN	41
Figura 5.6. Representación del funcionamiento de R-CNN	41
Figura 5.7. Estructura de R-CNN basado en LEGONet	42
Figura 5.8. Evaluación de R-CNN basado en LEGONet en función de diversos parámetros	43
Figura 5.9. Estudio de la segmentación por R-CNN al detectar piezas amarillas	44
Figura 5.10. Estudio de la segmentación por R-CNN al detectar piezas rojas	44
Figura 5.11. Estudio de la segmentación por R-CNN al detectar piezas azules	44
Figura 5.12. Estructura de R-CNN basado en LEGO16	45
Figura 5.13. Esquema del funcionamiento de Faster R-CNN	46
Figura 5.14. Esquema de la red RPN empleada en Faster R-CNN basado en LEGONet . .	48
Figura 5.15. Estructura de Faster R-CNN basado en LEGONet	48
Figura 5.16. Evaluación de Faster R-CNN basado en LEGONet en función de diversos parámetros	49
Figura 5.17. Estudio de la segmentación por Faster R-CNN basado en LEGONet al detectar piezas amarillas	50
Figura 5.18. Estudio de la segmentación por Faster R-CNN basado en LEGONet al detectar piezas rojas	50
Figura 5.19. Estudio de la segmentación por Faster R-CNN basado en LEGONet al detectar piezas azules	51
Figura 5.20. Esquema de la red RPN empleada en Faster R-CNN basado en LEGO16 . . .	52
Figura 5.21. Estructura de Faster R-CNN basado en LEGO16	52
Figura 5.22. Evaluación de Faster R-CNN basado en LEGO16 en función de diversos parámetros	53
Figura 5.23. Estudio de la segmentación por Faster R-CNN basado en LEGO16 al detectar piezas amarillas	54
Figura 5.24. Estudio de la segmentación por Faster R-CNN basado en LEGO16 al detectar piezas rojas	54
Figura 5.25. Estudio de la segmentación por Faster R-CNN basado en LEGO16 al detectar piezas azules	55

Figura 5.26. Esquema del funcionamiento de YOLO	56
Figura 5.27. Estructura de YOLO basado en LEGONet	57
Figura 5.28. Evaluación de YOLO basado en LEGONet en función de diversos parámetros	58
Figura 5.29. Estudio de la segmentación por YOLO basado en LEGONet al detectar piezas amarillas	59
Figura 5.30. Estudio de la segmentación por YOLO basado en LEGONet al detectar piezas rojas	60
Figura 5.31. Estudio de la segmentación por YOLO basado en LEGONet al detectar piezas azules	60
Figura 5.32. Estructura de YOLO basado en LEGO16	61
Figura 5.33. Evaluación de YOLO basado en LEGO16 en función de diversos parámetros	62
Figura 5.34. Estudio de la segmentación por YOLO basado en LEGO16 al detectar piezas amarillas	63
Figura 5.35. Estudio de la segmentación por YOLO basado en LEGO16 al detectar piezas rojas	63
Figura 5.36. Estudio de la segmentación por YOLO basado en LEGO16 al detectar piezas azules	63
Figura 6.1. Demostración del escalado de imágenes de profundidad	66
Figura 6.2. Mallado de imágenes de profundidad para determinación de altura	67
Figura 6.3. Demostración del proceso de cálculo de profundidad tras aplicar el desescalado	67
Figura 6.4. Filtrado por color para el análisis de la orientación	69
Figura 6.5. Separación y obtención de bordes para el análisis de la orientación	70
Figura 6.6. Análisis de una pieza mediante la transformada de Hough	70
Figura 6.7. Histograma de imágenes rotadas empleadas para evaluación	71
Figura 6.8. Diagrama de cajas de la orientación por la transformada de Hough	71
Figura 6.9. Imágenes para el entrenamiento de los modelos de regresión	72
Figura 6.10. Aumento de imágenes para el entrenamiento de los modelos de regresión	73
Figura 6.11. Histograma de imágenes rotadas empleadas para evaluación	73
Figura 6.12. Estructura de un modelo de regresión basado en LEGONet	74
Figura 6.13. Diagrama de cajas de la orientación mediante LEGONet	75
Figura 6.14. Estructura de un modelo de regresión basado en LEGO16	76
Figura 6.15. Diagrama de cajas de la orientación mediante LEGO16	77
Figura 7.1. Comparativa de los métodos de segmentación al detectar piezas amarillas	80
Figura 7.2. Comparativa de los métodos de segmentación al detectar piezas rojas	80
Figura 7.3. Comparativa de los métodos de segmentación al detectar piezas azules	81
Figura 7.4. Diagrama de barras: Comparación de la velocidad de diferentes métodos para la segmentación (más alto es mejor)	82
Figura 7.5. Diagrama de cajas: Comparación de la velocidad de diferentes métodos para la segmentación (más alto es mejor)	82
Figura 7.6. Diagrama de cajas: Comparación de la precisión de diferentes métodos para el cálculo de la orientación (cuanto más centrado respecto al cero y menos disperso mejor)	83
Figura 7.7. Diagrama de barras: Comparación de la velocidad de diferentes métodos para el cálculo de la orientación (más alto es mejor)	84

Figura 7.8. Diagrama de cajas: Comparación de la velocidad de diferentes métodos para el cálculo de la orientación (más a la derecha es mejor)	84
Figura A.1. Cámara Intel Realsense D435	89
Figura A.2. Alineamiento de imágenes de color y profundidad	90
Figura B.1. Desarrollo de una interfaz gráfica con App Designer	91
Figura B.2. Interfaz gráfica desarrollada con MATLAB	92
Figura C.1. Objetivos de desarrollo sostenible aprobados en 2015	95

Índice de tablas

Tabla 1.	Comparación del error medio, error máximo, precisión, desviación típica y velocidad de diferentes métodos para el cálculo de la orientación	XIX
Table 2.	Comparative of the mean error, the maximum error, precision, the typical deviation and speed of each method for calculating the orientation of pieces .	xxv
Tabla 4.1.	Base de datos para el entrenamiento de clasificadores	28
Tabla 4.2.	Opciones de entrenamiento de LEGONet	31
Tabla 4.3.	Evaluación de LEGONet: áreas encerradas debajo de la curva ROC para cada clase	31
Tabla 4.4.	Opciones de entrenamiento de LEGO16	34
Tabla 4.5.	Evaluación de LEGO16: áreas encerradas debajo de la curva ROC para cada clase	35
Tabla 5.1.	Base de datos para el entrenamiento de detectores de objetos	39
Tabla 5.2.	Opciones de entrenamiento de R-CNN basado en LEGONet	42
Tabla 5.3.	Opciones de entrenamiento de Faster R-CNN basado en LEGONet	49
Tabla 5.4.	Opciones de entrenamiento de Faster R-CNN basado en LEGO16	52
Tabla 5.5.	Opciones de entrenamiento de YOLO basado en LEGONet	58
Tabla 5.6.	Opciones de entrenamiento de YOLO basado en LEGO16	61
Tabla 6.1.	Resultados de la transformada de Hough	69
Tabla 6.2.	Opciones de entrenamiento del modelo de regresión basado en LEGONet . . .	74
Tabla 6.3.	Opciones de entrenamiento del modelo de regresión basado en LEGO16 . . .	76
Tabla 7.1.	Comparación del error medio, error máximo, precisión y desviación típica de diferentes métodos para el cálculo de la orientación	83

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 ha 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 está constituido por tres elementos que deben cooperar entre sí 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 procesamiento 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 conteniendo 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 Figura 1

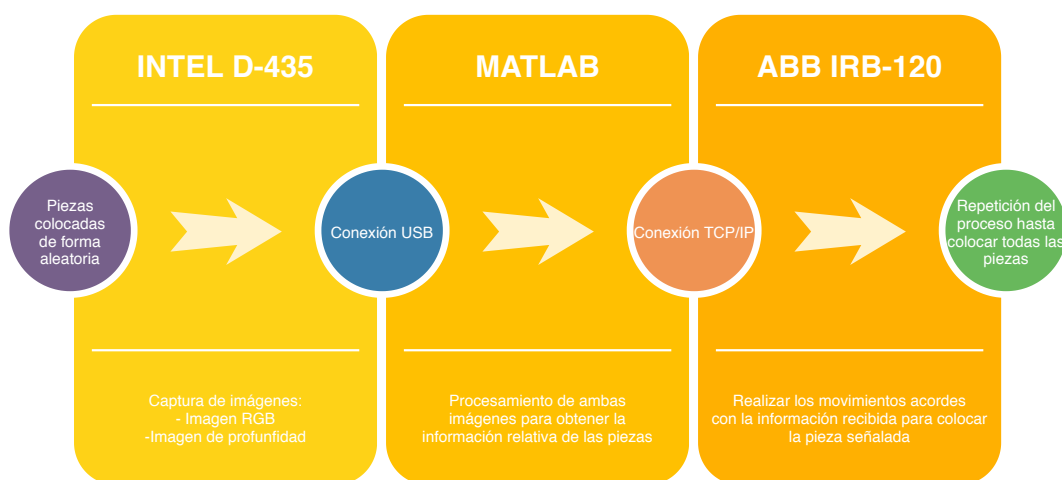


Figura 1. Esquema de la arquitectura del sistema

El primer paso en el desarrollo de este proyecto ha sido la mejora del 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 resultados al evaluar las piezas LEGO de color rojo en la Figura 2.

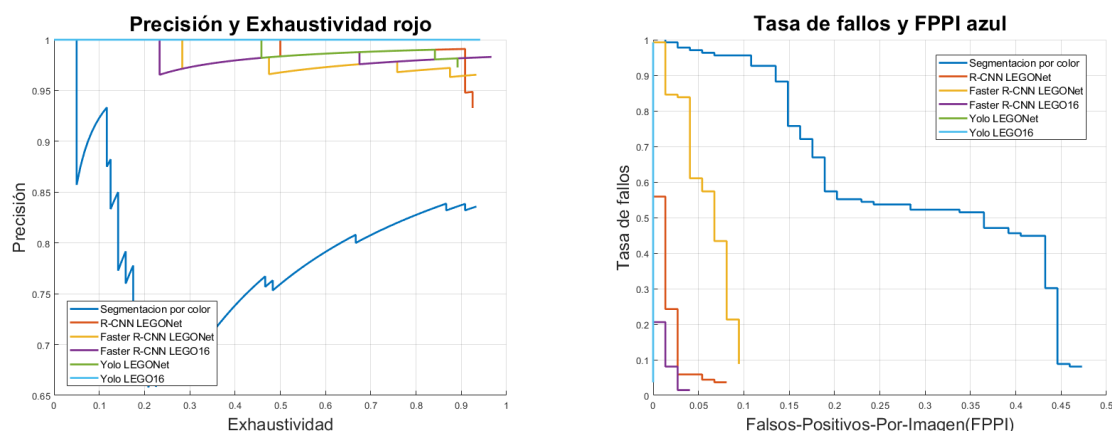


Figura 2. Comparativa de los métodos de segmentación al detectar piezas rojas

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.

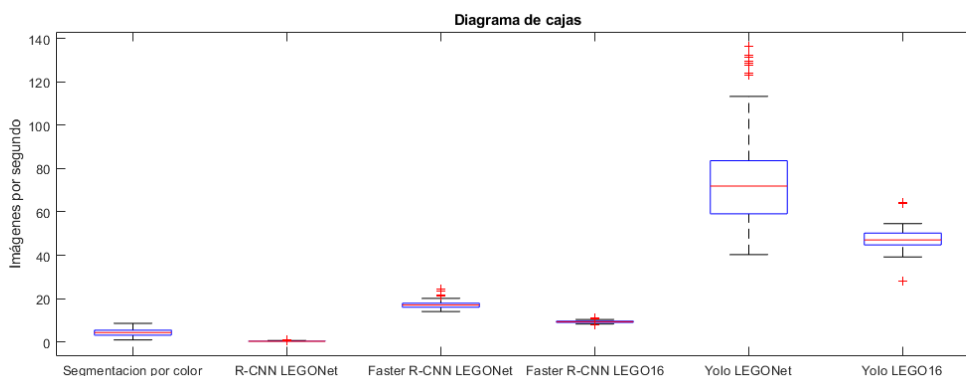


Figura 3. Diagrama de cajas: Comparación de la velocidad de diferentes métodos para la segmentación (más arriba es mejor)

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.

	Transformada de Hough	LEGONet	LEGO16
Error medio	0.98°	0.33°	0.71°
Error máximo	26°	1.3°	16°
Precisión	83.67 %	97.67 %	79.33 %
Desviación típica	1.82°	0.43°	1.23°
Piezas por segundo	89.7	74.6	178.5

Tabla 1. Comparación del error medio, error máximo, precisión, desviación típica y velocidad de diferentes métodos para el cálculo de la orientación

4. Conclusiones

El objetivo final de este proyecto es el perfeccionamiento del sistema previamente desarrollado por Ana Berjón Valles [Ber] 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.

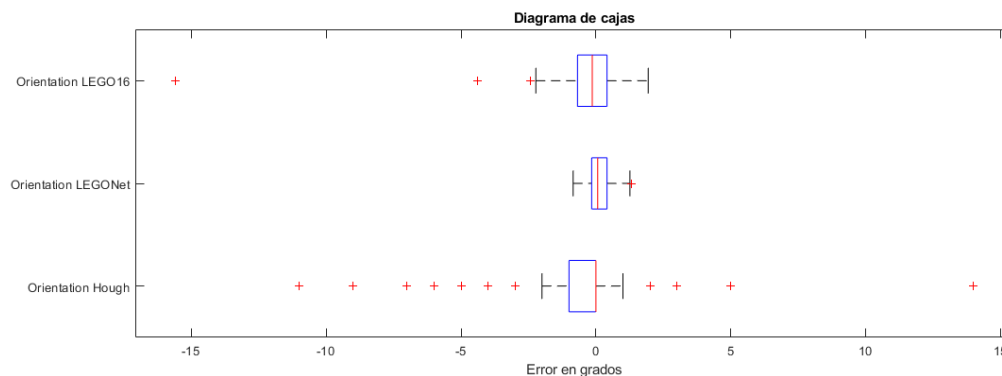


Figura 4. Diagrama de cajas: Comparación de la precisión de diferentes métodos para el cálculo de la orientación (cuanto más centrado respecto al cero y menos disperso mejor)

Uno de los principales objetivos consiste en subsidiar 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. This robots are now capable of completing task previously impossible to humans and at a faster and more efficient rate. Because of this advances Comillas ICAI has decided to implant this new systems to it's industrial robots. This robots have to be capable of identifying and recollecting LEGO pieces randomly set at a work bench. To do so this project takes upon the previous work of Ana Berjón Valles to improve it. It will also create new systems base 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 it's robots. This robots have to be capable of identifying and recollecting LEGO pieces randomly set at a work bench. 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 base on colour masking, edge detection and the Hough transform. Also the systems designed to analyse the depth and rotation of the pieces has been redesigned and upgraded. However, despite this upgrades our system is still incapable to compete with the modern systems already well implanted in the industry. This new systems are based on convolutional neuronal networks. As it's name indicates this networks are based on the principles of convolution and deep learning. Because of it we have decided to upgrade the robots with this new technologies and improve it's 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

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 Figure 5.

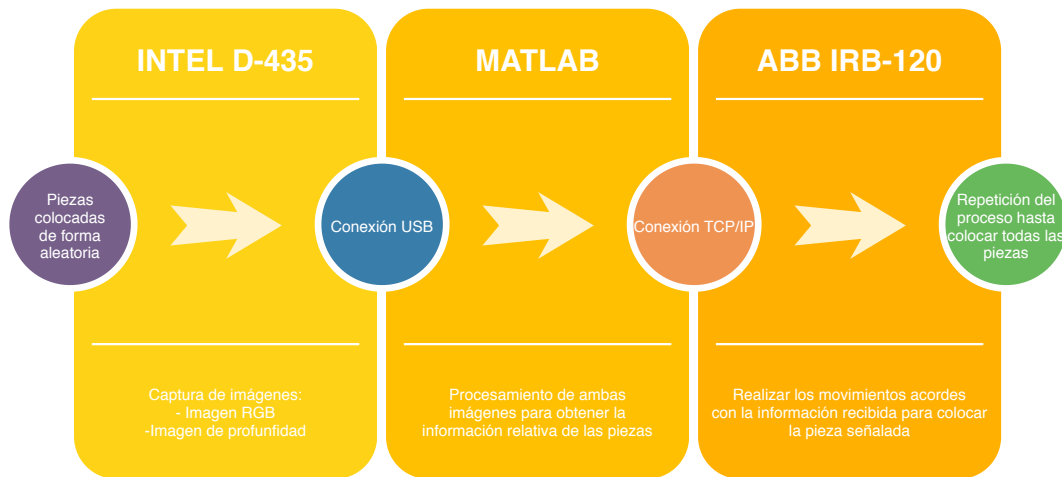


Figure 5. Architecture of the system

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 develop 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 use on the creation of object detectors. That's the reason why we have decided to use AlexNet ang VGG-16. We have decided to use these network so that we could evaluate the impact of different networks size and to demonstrate the evolution of neuronal networks. AlexNet was created on 2012 while VGG-16 is from 2014 and currently is still a very modern and capable network. But because this networks where 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 networks 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 improve compared to the Sliding windows systems but it still is a slow process compared to newer systems. In exchange for it's poor performance this system is characterize for having good precision. During the development of this project two networks base on LEGONet and LEGO16 have been created. This 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.

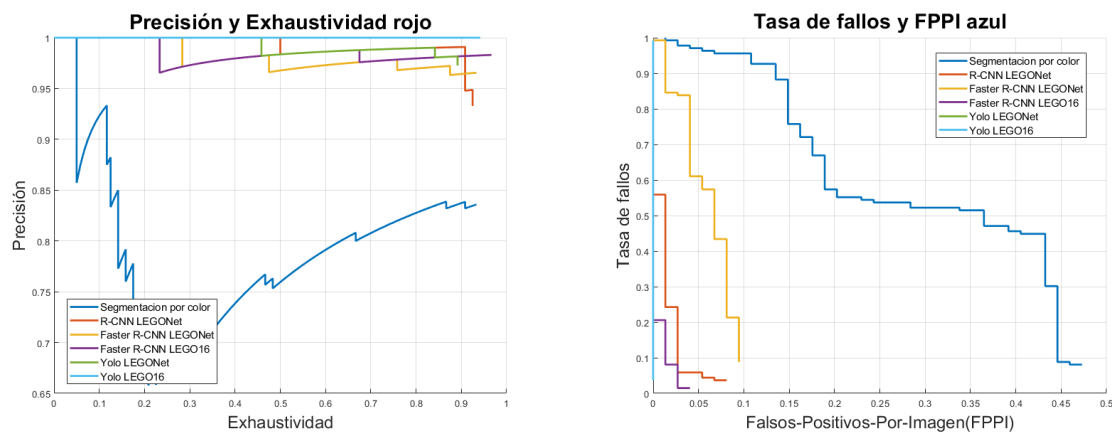


Figure 6. Comparison of the different segmentation methods to detect red LEGO pieces

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

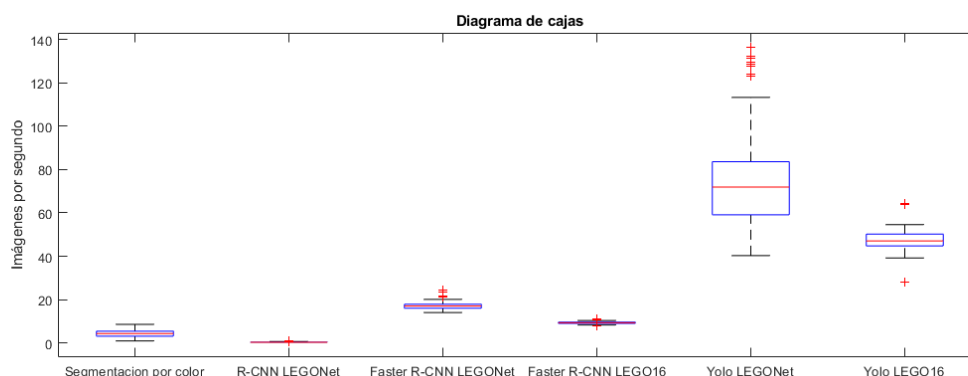


Figure 7. Box diagrams: comparison of the speed of different object detectors (Higher is better)

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

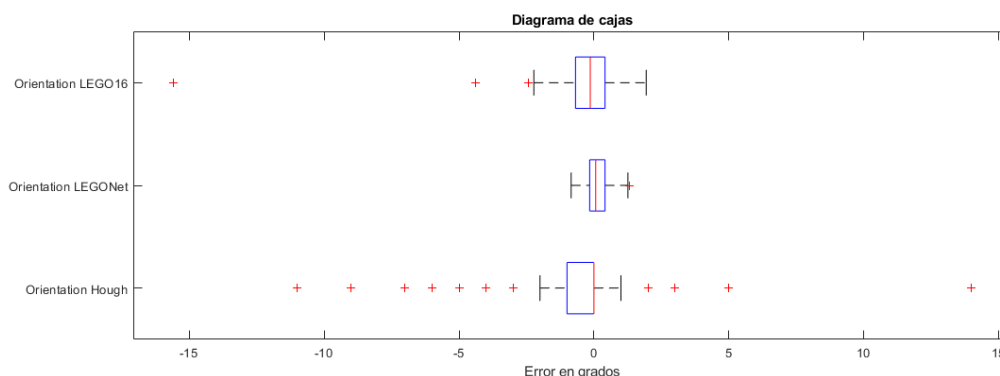


Figure 8. Box diagram: Comparison of the precision of different methods for calculating the orientation of pieces (the closer to the zero the better)

4. Conclusions

The objective of this project is the perfection of a previous system developed by Ana Berjón Valles [Ber]. All the objectives set during the development of this project have been successfully completed. Bellow 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 it's 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 una gran parte 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 [Lab]

Gracias al desarrollo de nuevas tecnologías en el sector fotográfico, se desarrollaron nuevos métodos para la captura de imágenes que permitían mejorar la calidad de estas, así como la captura de nueva información como la dimensión del objeto y su distancia respecto a la cámara. También se consiguió un incremento significativo de la capacidad computacional de los nuevos ordenadores. Y es debido a estos dos avances que se ha permitido el desarrollo de nuevos y más sofisticados sistemas de reconocimiento de objetos. Estos nuevos sistemas tienen una gran variedad de usos como conducción autónoma [WMZ12], sistemas de seguridad autónomos [YC05], control forestal [ART15], etc.

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 más simples empezados

a desarrollar en 1960 y que han sido 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. Además, en espacios controlados pueden llegar a dar mejores resultados que sistemas más complejos y avanzados [Cas+13]. Por ello se van a emplear estos sistemas para el desarrollo de este proyecto [Kat19].

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. Pero debido a la capacidad de cálculo necesaria para entrenarlos, se escapan del principal objetivo de este proyecto.

El fin de este proyecto es la mejora de un sistema de reconocimiento y recogida de piezas LEGO con un brazo robótico. Para ello se parte de un proyecto previo desarrollado por Ana Berjón [Ber] y un robot dotado con una cámara RGB-D. El objetivo es mejorar 3 sistema actual y hacerlo más robustos ante cambios en la escena (iluminación, disposición de las piezas, etc.).

1.1. 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. 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 reprogramar 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 brazo robótico. Con este proyecto se podrá modernizar las instalaciones de ICAI y aumentar las capacidades de los robots actuales, así como instruir a futuros alumnos sobre la visión artificial.

1.2. Objetivos

Este proyecto parte del trabajo previo de Ana Berjón Valles y Lucia Díaz García y el objetivo es mejorar el sistema actual y dotarlo de nuevas capacidades. Para ello se ha mejorado la respuesta del sistema ante cambios de iluminación y de escena mediante el uso de redes neuronales. Gracias al uso de estas redes también se ha mejorada la robustez y rapidez del sistema.

El anterior sistema presentaba problemas de conexión y por ello en ocasiones la cámara dejaba de responder o las imágenes eran corrompidas. Como objetivo de este proyecto se ha decidido mejorar la conexión con la cámara. Para ello se ha diseñado desde cero un nuevo *driver* de la cámara que permite una conexión más rápida y estable sin imágenes corrompidas. Este mismo *driver* se emplea también para la adquisición de la imagen de profundidad. Además, se ha implantado un método para corregir la distorsión y diferencia de enfoque entre la imagen de color y la de profundidad permitiendo así realizar medidas más precisas.

El análisis de la profundidad ha sido perfeccionado con el fin de mitigar el efecto causado por los reflejos. Para ello se ha diseñado una nueva forma de trabajo y análisis así como la opción de calibración para la profundidad.

Mejora del sistema de cálculo de la orientación de una pieza. Se ha mejorado el anterior sistema de forma que siempre pueda detectar el ángulo y se ha reducido el error cometido. Para ello se realiza un análisis mas exhaustivo de la pieza. Además, se han desarrollado redes neuronales para llevar a cabo esta tarea.

Para mejorar las instalaciones de la universidad y dotar de más capacidad al resto de robots del laboratorio se han generalizado en todo lo posible los algoritmos empleados para permitir así su implantación en otros sistemas.

Se han planteado más objetivos adicionales para realizar en el transcurso de este proyecto pero debido a una pandemia no se ha podido tener suficiente acceso al robot y por lo tanto han tenido que ser retrasados. Estos son el uso de piezas de LEGO dobles y el establecimiento de una comunicación bidireccional entre el brazo robótico y MATLAB. En su lugar, se ha desarrollado en mayor profundidad las redes neuronales.

1.3. Herramientas

Para el desarrollo del proyecto se requiere de:

- Brazo robótico ABB IRB120 [ABBa] y RobotStudio 2019.5 [ABBb].
- Cámara Intel RealSense D435 [Intc] y Wrapper de MATLAB para Intel Realsense [Intb].
- MATLAB 2019 B con los siguientes paquetes:
 - Deep Learning Toolbox Model for AlexNet Network.
 - Deep Learning Toolbox Model for VGG-16 Network.
 - Instrument Control Toolbox.
 - Deep Learning Toolbox.
 - Bioinformatics Toolbox.
 - Computer Vision Toolbox.
 - Parallel Computing Toolbox.
 - Image Processing Toolbox.
 - Matlab Support Package for USB Webcams.
 - Statistics and Machine Learning Toolbox.

1.4. Arquitectura del sistema

El sistema se compone de tres elementos claves: un brazo robótico para poder interactuar con las piezas, una cámara montada sobre el brazo robótico para la captura de imágenes y MATLAB para el procesamiento de las imágenes y la toma de decisiones. Es necesario que estos tres elementos funcionen correctamente y estén comunicados entre sí. Como ya se ha mencionado, todas las comunicaciones y decisiones son tomadas por el ordenador y transmitidas a la cámara por conexión USB y al brazo robótico mediante un socket TCP/IP.

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

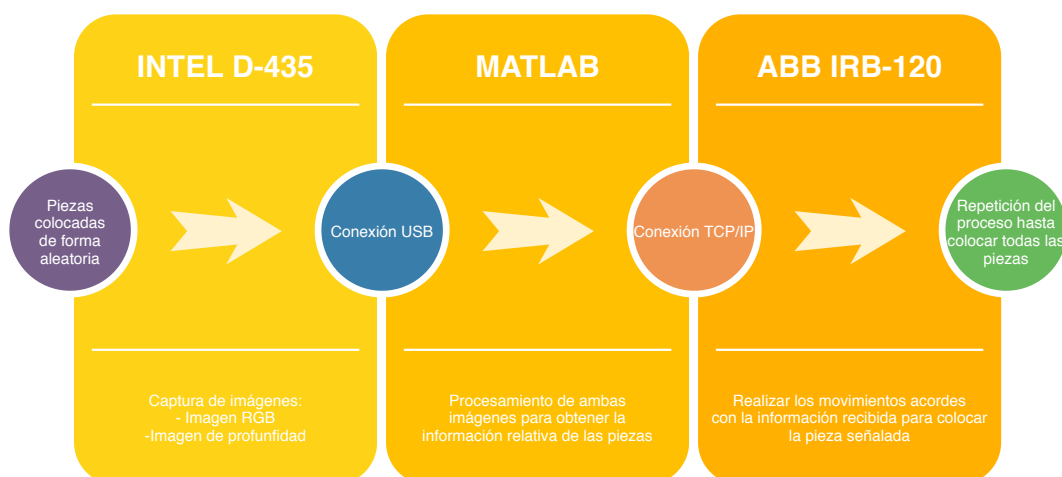


Figura 1.1. Esquema de la arquitectura del sistema

ver gráficamente en Figura 1.2. Al arrancar el programa, el primero proceso es la calibración del sistema. Para ello se inicia la conexión de la cámara y se le da tiempo para permitir que se ajuste a las condiciones lumínicas del momento. Acto seguido, se tomará una imagen de profundidad que se usará como referencia de suelo para calcular alturas. Por último, se carga en memoria la red neuronal a emplear para la detección de las piezas.

Una vez iniciado el sistema se puede comenzar a identificar piezas para su futura colecta. El primer paso es la captura de una imagen de color y otra de profundidad, para ello es necesario que el robot se encuentre en su posición de captura de imágenes. Una vez situado en la posición de captura, se toma la imagen y comienza el procesamiento de las imágenes. Para ello existen diferentes sistemas de detección que se explicarán más adelante, pero el objetivo de todos ellos es el mismo, identificar la pieza, su orientación y su altura. Todo este proceso se lleva a cabo con MATLAB y se realiza por etapas. La primera etapa consiste en la detección de la pieza en la imagen. Para ello como se desarrollará más adelante, se van a emplear diversos detectores de objetos. Una vez esta ha sido detectada se deben de extraer sus características. Estas consisten en la extracción de su orientación y de su altura. Para la extracción de la orientación existen diversos métodos que se desarrollaran más adelante pero el objetivo es obtener el ángulo de la pieza respecto a un eje del robot para así estimar cuanto debe de rotar este. El análisis de la altura se hace con la ayuda de la imagen de profundidad. Al comparar la altura de la pieza frente al nivel del suelo de la calibración se puede estimar la altura de la pieza y el número de piezas de LEGO apiladas.

Tras obtener las coordenadas de la pieza en la imagen, su orientación y altura, se transforman estas coordenadas en píxeles a coordenadas respecto al robot. Una vez obtenidas todas las coordenadas, estas son mandadas al robot para la recolección de las piezas y su disposición en una región de recolección determinada. Una vez terminado este proceso, el robot volverá a la posición de captura y se volverá a iniciar todo el proceso hasta que se haya recolectado todas las piezas.

1.5. Organización del documento

La estructura de este documento se asemeja a la seguida durante el desarrollo de este proyecto. De esta forma se asegura una buena conexión entre los diversos capítulos y sus respectivas secciones. La memoria se distribuye en 9 capítulos, siendo el presente el primero de

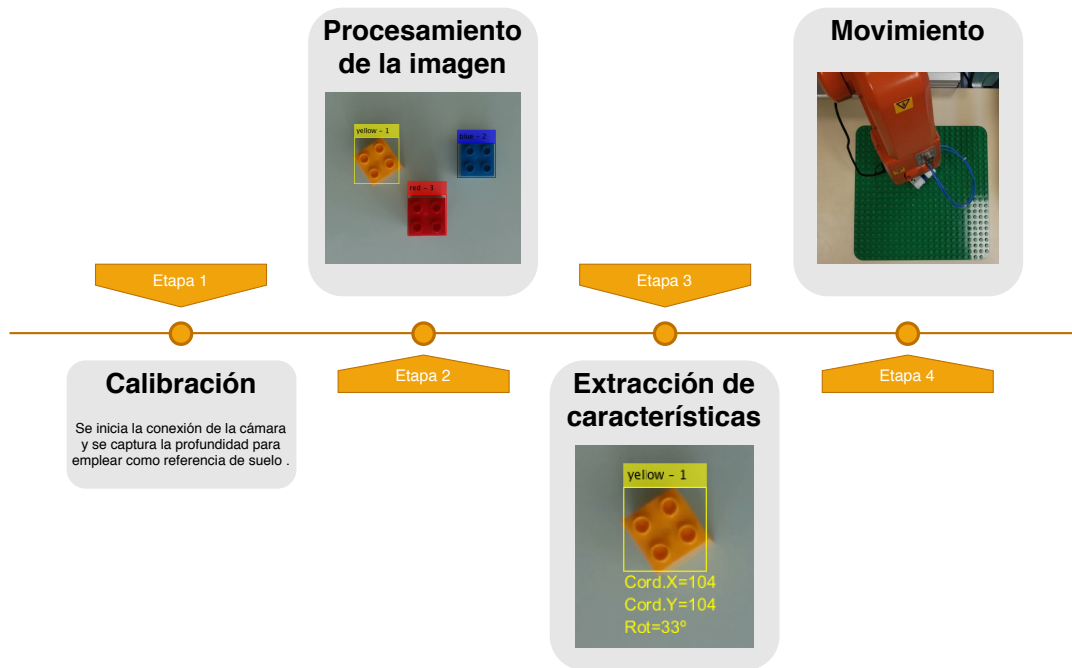


Figura 1.2. Diagrama de las etapas del sistema

ellos, que introduce el problema del proyecto y la motivación del mismo. El siguiente capítulo contiene la revisión de la literatura, en este capítulo se contextualiza el proyecto y se desarrollan todos los temas tratados en este proyecto.

Una vez introducido y contextualiza el proyecto se comienza a desarrollar las herramientas y métodos necesarios. Esto comienza por el perfeccionamiento del trabajo desarrollado por Ana Berjón Valles y Lucia Díaz García. Esto se lleva a cabo en el tercer capítulo con la segmentación con mascarar de color. El cuarto capítulo introduce las redes neuronales y sus capacidades. En este capítulo también se desarrollaran dos clasificadores para su posterior reaprovechamiento en el capítulo 5. En el quinto capítulo se desarrollan diversos métodos para la segmentación con redes neuronales y se evalúan dichos métodos. El sexto capítulo trata sobre la extracción del resto de atributos de las piezas de LEGO, esto incluye la obtención de la altura así como la estimación de la orientación. Una vez desarrollados todos los métodos para la extracción de atributos y la detección, se compararan y contrastaran resultados en el séptimo capítulo.

Por último, en los dos últimos capítulos se desarrollará una conclusión final y completa del proyecto así como futuras vías de desarrollo y de mejora de este proyecto. Esto se llevará a cabo en los capítulos octavo y noveno respectivamente.

2

Estado del arte

En este capítulo se desarrollará la situación sobre la que se desarrolla el proyecto. Se presentará la situación actual y se explicaran algunas de las herramientas y algoritmos usados en el proyecto.

Como ya se ha indicado en Sección 1.4, este proyecto está claramente definido por dos elementos que deben de coaccionar entre sí para poder cumplir el objetivo de identificar y recolocar piezas de LEGO. 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.

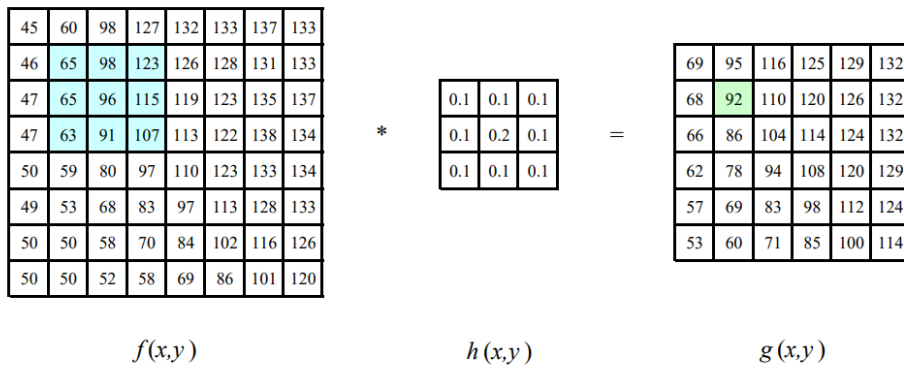


Figura 2.1. Ejemplo de un proceso de convolución aplicado a una matriz (Fuente: Computer Vision: Algorithms and Applications, Richard Szeliski figure 3.10 [Sze11])

Un claro ejemplo de este tipo de filtros es el suavizado. Permite suavizar la imagen y hacerla más homogénea al combinar cada píxel 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 [Sze11].

- **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 en lugar de tres. Sin embargo, en este proyecto es importante identificar el color de las piezas, por ello primero se aplicará una mascarará de color para detectar dichas piezas y después se transformará a escala de grises. 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].

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 píxeles [Ber].

En la práctica, estos cambios no son tan bruscos, sino que se producen gradualmente a lo largo de varios píxeles. 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 píxeles 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.

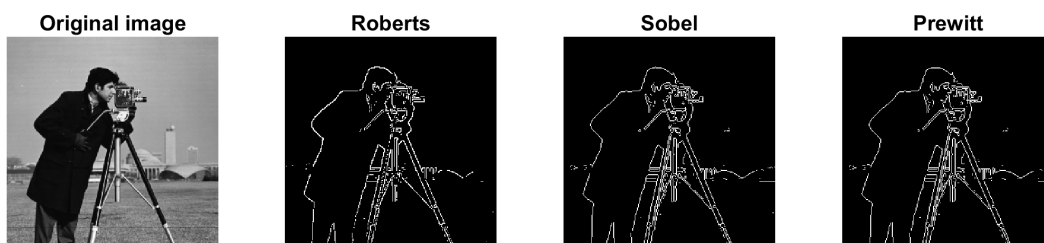


Figura 2.2. Comparativa de algoritmos de detección de borde (Fuente: Ejemplo de MATLAB con imágenes de stock)

- **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 [Ber].

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 (LDP) 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, presente un gran inconveniente que impide que sea usada tan extensamente. Requiere de una alta carga computacional ya que debe de 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 o mejorarlo 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.

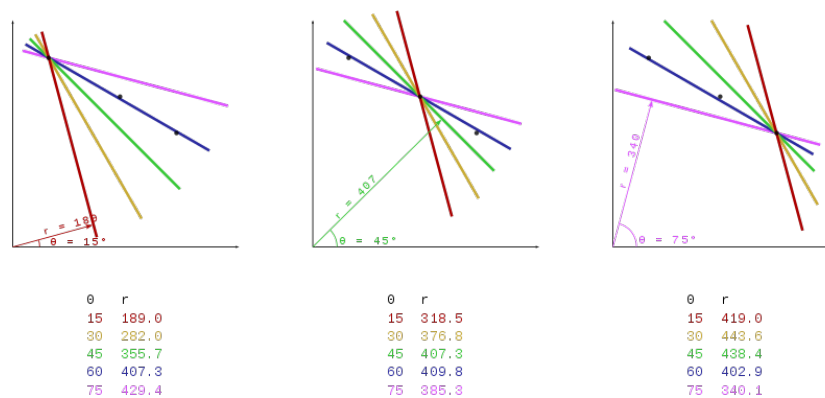


Figura 2.4. Representación de la transformada de Hough (Fuente: https://en.wikipedia.org/wiki/Hough_transform)

- **RANSAC:** También conocido como RANdom SAMple Consensus, 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. RANSAC

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 R-RANSAC (Recursive RANSAC).

2.1.2. Redes Neuronales Convolucionales

La idea del aprendizaje automatizado/aprendizaje profundo/inteligencia artificial no es novedosa, ya en 1955 IBM formo 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 que 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.

Que 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 Convolucional impulsado por Yann LeCun. Yann desarrollo una de las primeras redes capaces del auto aprendizaje diseñada para reconocer la escritura a mano [Lec+89]. LeCun destaco por el uso de la convolución y fue uno de los primeros en obtener unos resultados positivos con este método.

Pero no fue hasta 2012 cuando 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 procesamiento 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.

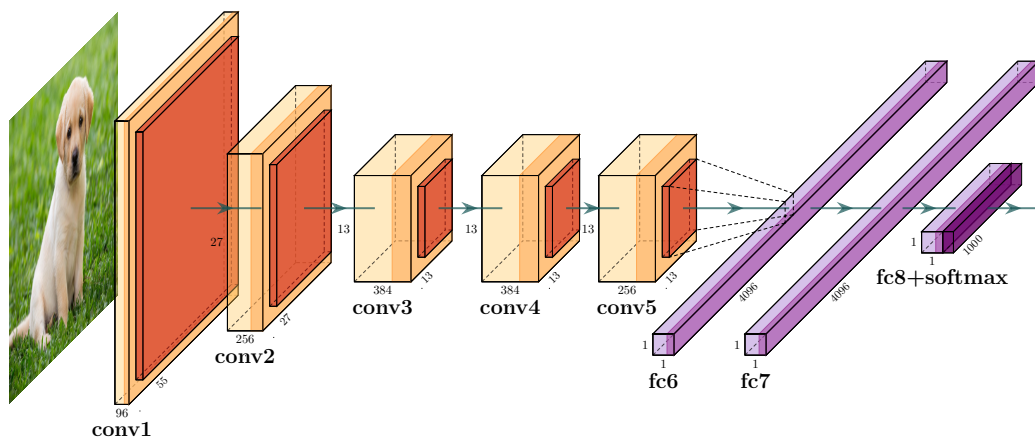


Figura 2.5. Representación de la estructura de AlexNet [KSH17]

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 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:** Son conocidas como R-CNN y se basan en el principio de "propuesta de regiones". Anteriormente, se usaba un sistema conocido 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 muy costoso a nivel computacional y lento, R-CNN propone

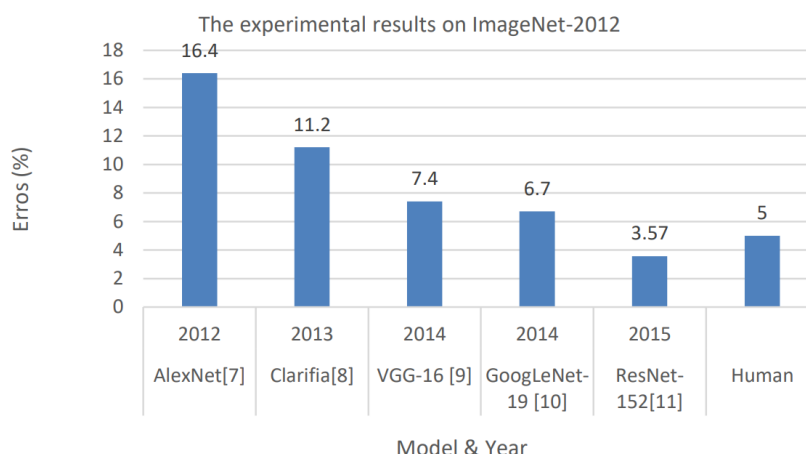


Figura 2.6. Errores cometidos en ImageNet en los últimos años (Fuente: The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches [Alo+18])

usar un sistema que proponga un máximo de 2000 regiones a estudiar. De esta forma se reduce bastante la carga, pero manteniendo un buen nivel de confianza y precisión.

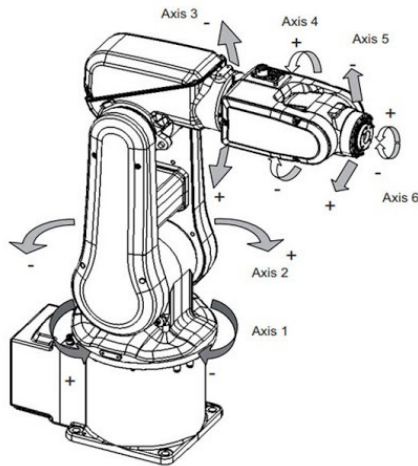
- **Faster Region-proposal Convolutional Neuronal Network:** Son conocidas como Faster R-CNN y al igual que R-CNN se basan en el principio de "propuesta de regiones" pero difiere de este en el método para proponer dichas regiones de interés. En este método, la propia red se encarga de analizar la escena y decidir que regiones debe de considerar de interés. Esto resulta en una red bastante más rápida, aunque puede ser menos precisa que R-CNN.
- **You Only Look Once:** Este método es conocido como YOLO y 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 es más imprecisa al localizar los objetos.

2.2. Robótica Industrial

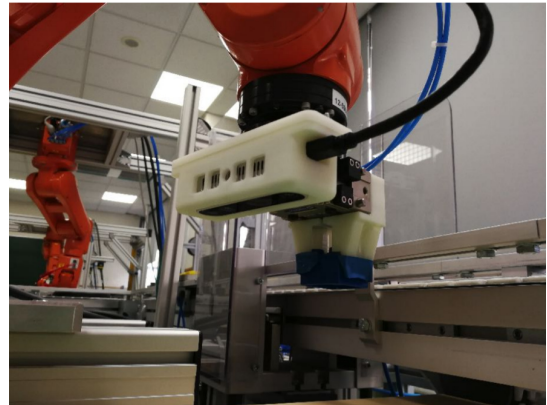
El termino robot se remonta hasta 1921 donde por primera vez se introdujo en la obra de teatro R.U.R (Rossum's Universal Robots) obra del novelista y autor checo Karel Čapek. La palabra deriva de robota que en checo significa fuerza del trabajo o servidumbre [Oll05]. En la actualidad es un término aceptado 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:



(a) Esquema del IRB120



(b) Cabezal del IRB120

Figura 2.7. Brazo robótico IRB120

- 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 de 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].

Como ya se ha explicado previamente, el objetivo de este proyecto es el desarrollo de un sistema capaz de localizar pieza LEGO y reubicarlas. 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, un cámara y una unidad de procesamiento para analizar las imágenes y 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].

Para la elaboración de este proyecto se va a usar el brazo robótico ABB IRB120 [ABBa], al cual se le ha modificado el cabezal de forma que pueda alojar una cámara Intel RealSense D435

[Intc] junto con un sistema de pinza para la recogida de piezas, mostrados en la Figura 2.7. Este mecanismo fue elaborado previamente por Ana Berjón y se explica de forma más detallada su funcionamiento en su TFG [Ber].

3

Segmentación con mascarar de color

El presente capítulo describe de forma detallada el proceso de análisis de imágenes mediante segmentación por color. El proceso para la detección de piezas se divide en tres fases: segmentación por color, mejora de la imagen y extracción de atributos.

El primer paso para poder recolocar piezas es localizarlas. Con el objetivo de localizarlas, se han desarrollado múltiples herramientas capaces de hacerlo, pero cada una presenta grandes ventajas y desventajas. Pero para poder entender la evolución y el desarrollo de estas es necesario entender el contexto de este proyecto. Se parte del trabajo previo llevado a cabo por Ana Berjón [Ber], y el objetivo de este proyecto era el perfeccionamiento de este sistema. Tanto el perfeccionamiento del procesado de la imagen como de la interacción con el robot. Desgraciadamente, debido a una pandemia ha sido imposible poder trabajar con el brazo robótico y por ello este proyecto se ha centrado solo en el procesado de la imagen.

Teniendo todo lo anterior en cuenta, se puede entender la evolución del proyecto:

- Perfeccionamiento del sistema basado en filtros de color, detección de borde y de formas.
- Desarrollo de clasificadores
- Desarrollo de detectores basados en los clasificadores y R-CNN.
- Desarrollo de detectores basados en los clasificadores y Faster R-CNN.
- Desarrollo de detectores basados en los clasificadores y YOLO.

Se van a desarrollar de forma individual cada uno de estos métodos explicando su estructura, funcionamiento y capacidades y por último, se hará una comparativa de todos los métodos desarrollados. Se va a comenzar desarrollando el perfeccionamiento en filtros de color, detección de borde y de forma.

Como su nombre indica, es un proceso que depende fuertemente del filtrado de color. Esto se debe a qué para localizar las piezas, obtener sus atributos y su orientación es recomendable

trabajar en escala de grises e imágenes binarias. Pero para esta aplicación también es necesario saber el color de la pieza. Por ello primero se debe de filtrar por color para poder separar las piezas. Esta etapa es crucial y por ello también es el motivo de fallo del sistema. Como se verá mas adelante, cambios en la iluminación pueden implicar un fallo total del sistema y este deja de ser capaz de identificar piezas.

Para analizar cada imagen, se debe de repetir el mismo proceso para cada posible color de las piezas de LEGO. El proceso a seguir se puede dividir en tres etapas:

3.1. Filtrado de color

Se aplican tres filtros de color a la imagen original con el objetivo de separar las piezas y clasificar las por su color. Para trabajar con mayor facilidad y precisión, primero se transforma la imagen al formato HSV. A continuación, se crea la máscara binaria, para ello se comparan los píxeles de la imagen en formato HSV, frente a unos máximos y mínimos. Si el píxel se encuentra dentro de los valores permitidos, se representará en la imagen binaria como un uno, en caso contrario será un cero. De esta forma, se obtienen las máscaras mostradas en la columna del medio de la Figura 3.2. Y multiplicando la imagen original por la imagen binaria, se obtienen las imágenes de la última columna.

Desgraciadamente, cambios en la iluminación pueden modificar severamente los resultados de los filtros de color. Este proceso se puede ver a continuación en la Figura 3.1. Para evitar que esto suceda, es recomendable calibrar de forma regular los filtros de color y asegurarse de que las condiciones lumínicas permanezcan lo más constantes posibles.

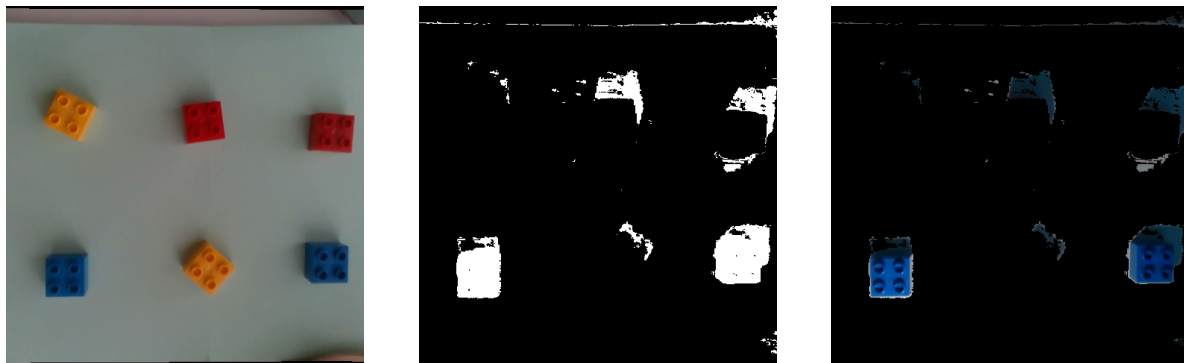


Figura 3.1. Error por mala calibración del filtrado por color

Para el desarrollo de los filtros por color, se ha usado una herramienta de MATLAB llamada `colorThresholder`. Con la ayuda de esta herramienta se han creado tres funciones. Una función para cada color la cual devuelve la imagen binaria y la combinación de la imagen original y la imagen binaria.

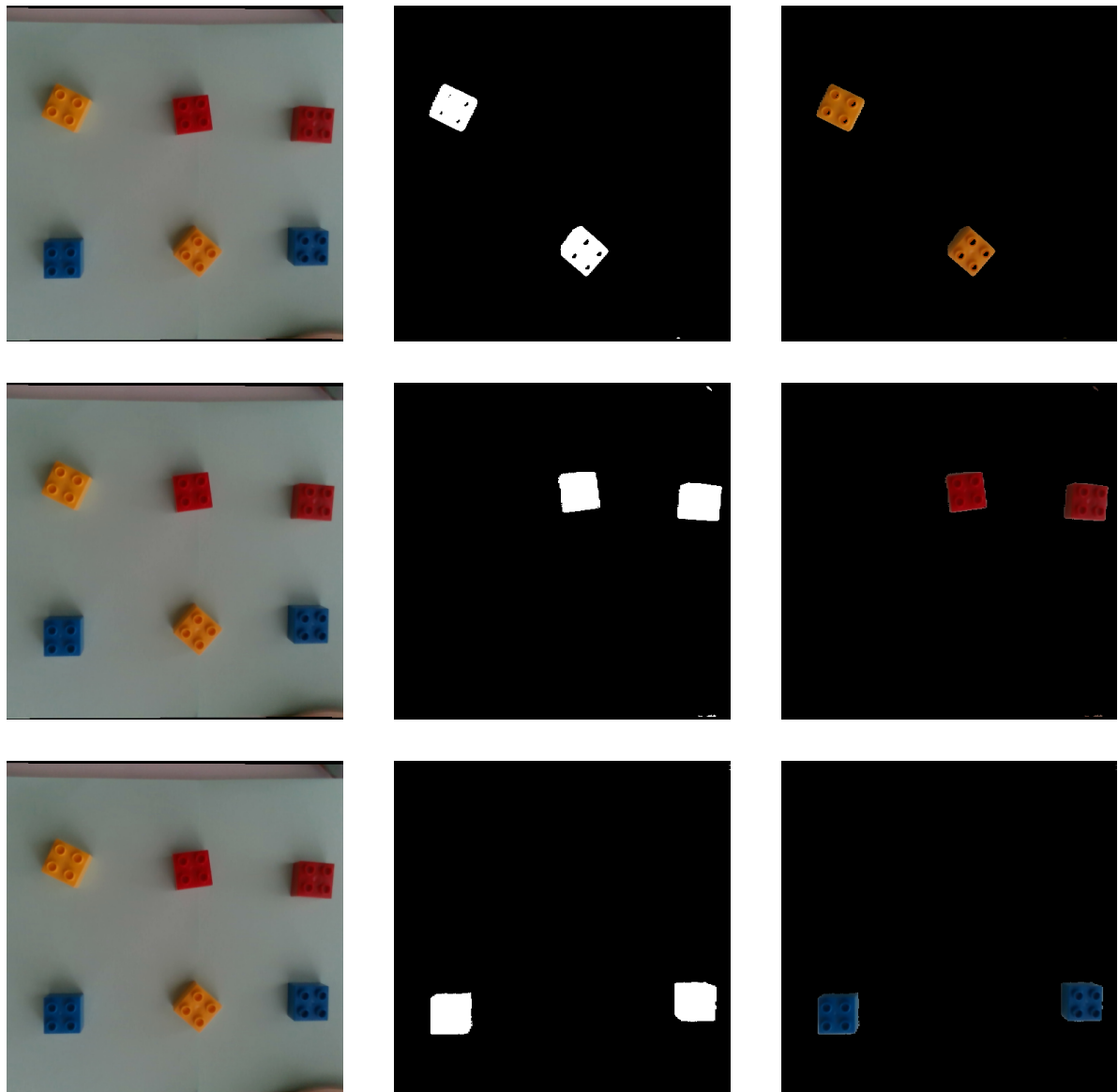


Figura 3.2. Filtrado por color

3.2. Identificación y mejora de la imagen

Una vez aplicado el filtro de color se realiza un estudio exhaustivo de las piezas. Este comienza por la separación de las piezas para cada color. Para ello, primero se hace una limpieza de la imagen eliminando los pequeños objetos de la imagen binaria (ruido). A continuación, se emplea la función de MATLAB `regionprops` que devuelve las propiedades de los objetos encontrados y el número de objetos encontrados. De esta forma, ya sabemos el número de piezas de cada color y de forma aproximada su posición y área.

Desgraciadamente la información extraída con `regionprops` no es suficientemente precisa. Sobre todo, cuando se tienen varias piezas apiladas ya que con los filtros también se puede haber detectado las piezas inferiores aumentando así el tamaño de la pieza. Por ello es necesario un estudio más exhaustivo analizando cada pieza de forma independiente. Para facilitar el cálculo y la mejor detección de la pieza, se pasan todas las piezas a escala de grises [AMS18]. A continuación, se muestran el proceso de separación de piezas y transformación a escala de grises.

El proceso a seguir para mejorar la precisión de la detección de la pieza consiste en eliminar el borde de la pieza y los elementos pequeños. De esta forma se intenta solo detectar la cara superior del LEGO. Una vez eliminado el borde se vuelve a emplear `regionprops` para obtener todas las propiedades, pero con una mayor precisión.

Para poder detectar correctamente el borde y eliminarlo de la imagen primero es necesario ajustar y filtrar la imagen. El proceso total de mejora de la imagen y extracción de características consta de cinco pasos:

1. Se realiza un ajuste de de la intensidad de la imagen en escala de grises. De esta forma se resaltan más los cambios de color/iluminación.
2. Se desea resaltar aún más los detalles de la imagen. Esto se va a llevar a cabo con la ayuda de un filtro de difuminado Gaussiano. Primero se difumina la imagen para ensalzar los rasgos generales y después se le restan estos rasgos a la imagen original. De esta forma se destacan los pequeños detalles.
3. Una vez mejorada la imagen, toca detectar el borde de la pieza. Para ello se va a aplicar el algoritmo de Canny.
4. Al filtrar las imágenes por color también se puede haber incluido las caras laterales de la pieza y las caras laterales de las piezas situadas debajo de la pieza a detectar. Para reducir el efecto que esto puede causar, se ensanchan los bordes de forma que incluyan pequeños detalles que puedan estropear el proceso.
5. El último paso para obtener la cara interna del LEGO consiste en invertir los colores de los bordes obtenidos. De esta forma, solo se conserva la cara interna.

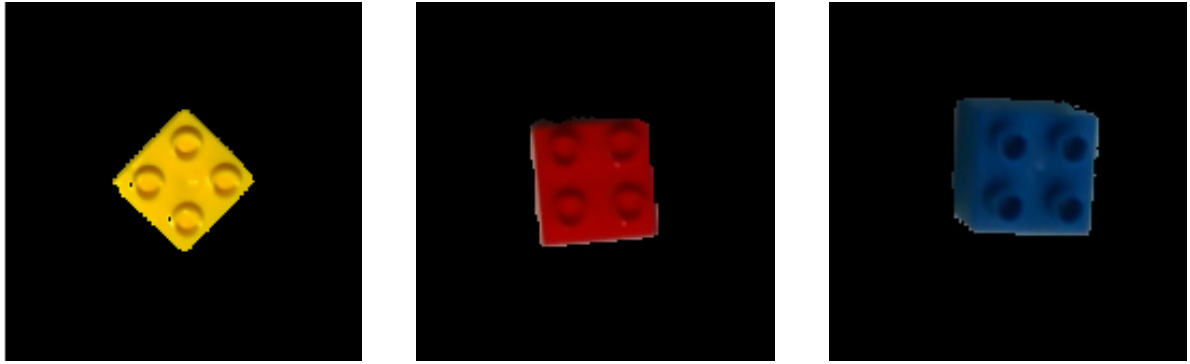


Figura 3.3. Segmentación de las piezas a analizar

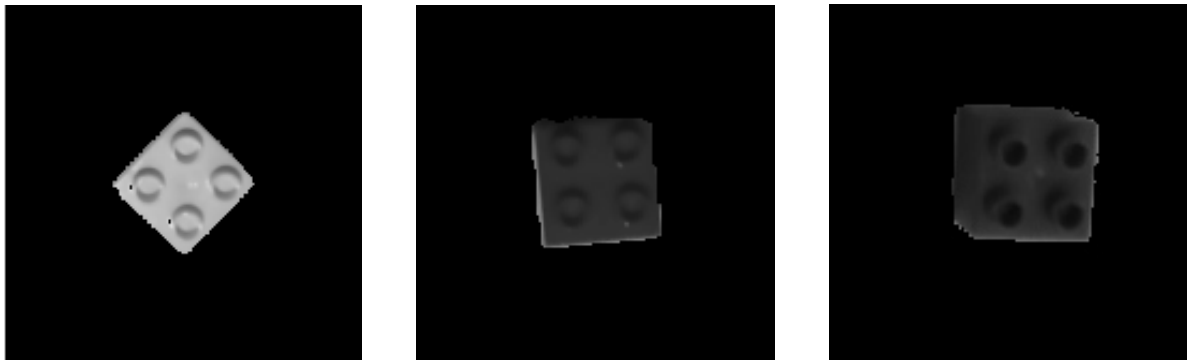


Figura 3.4. Escalado a gris de las piezas segmentadas

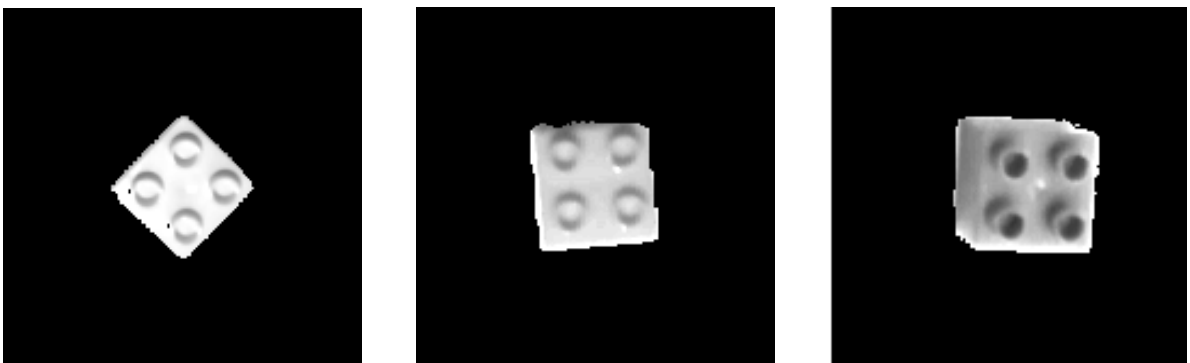


Figura 3.5. Primer paso: ajuste de intensidad

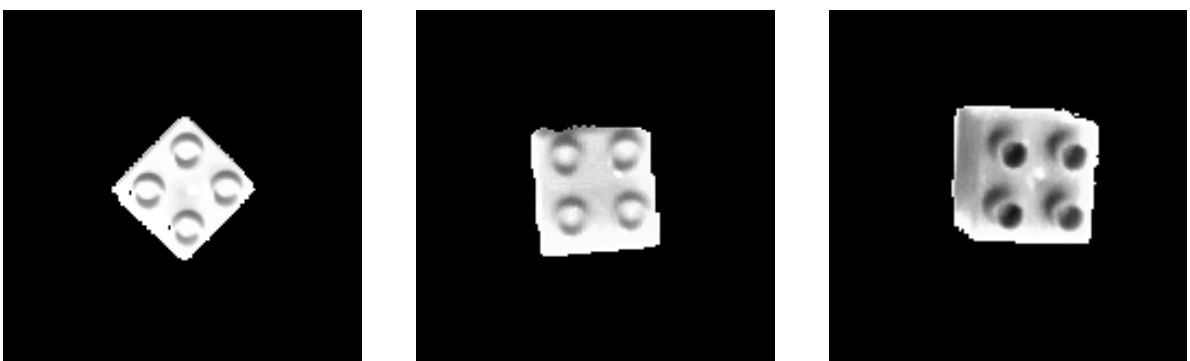


Figura 3.6. Segundo paso: Resalto de pequeños detalles

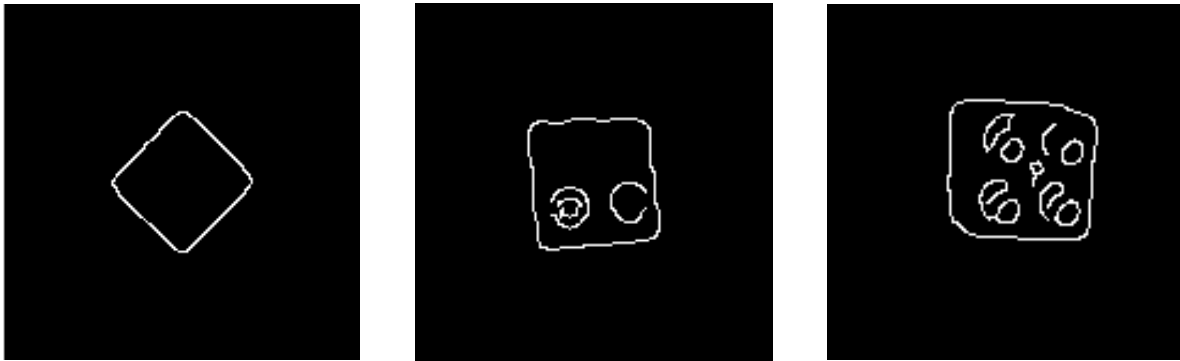


Figura 3.7. Tercer paso: aplicación del algoritmo de Canny

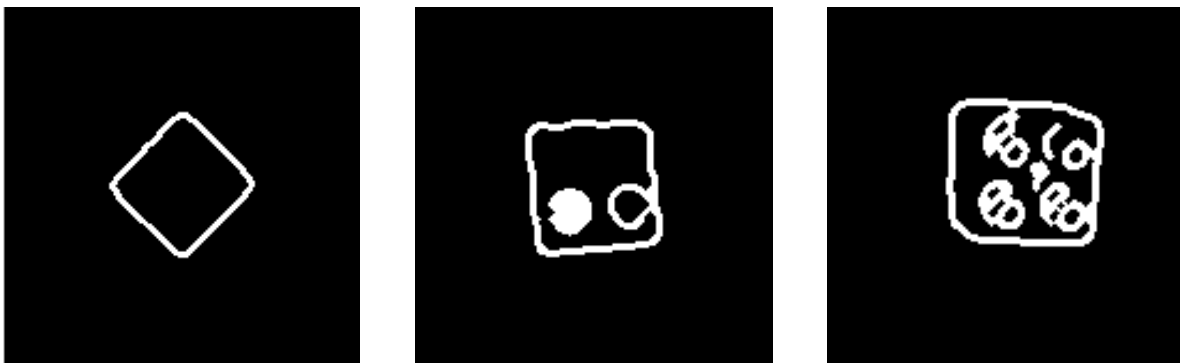


Figura 3.8. Cuarto paso: dilatación y contracción de los bordes

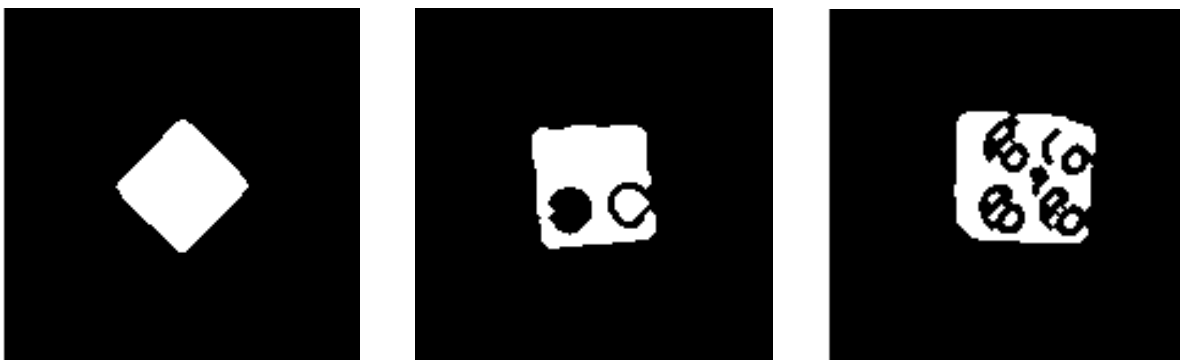


Figura 3.9. Quinto paso: Inversión de los colores de la pieza

3.3. Extracción de características

El último paso para la obtención de las características de la pieza se vuelve a emplear la función `regionprops` de MATLAB. con cada una de las piezas identificadas. Al trabajarse de forma individual con cada pieza y como estas han sido mejoradas, la precisión es bastante elevada y se puede obtener de forma bastante precisa el centroide de cada pieza.

3.4. Resultados

Para comprobar la eficacia de la segmentación por color, se ha desarrollado un *script* con la ayuda de MATLAB para determinar su capacidad de para identificar piezas y la precisión con las que la detecta. En este *script* se han analizado 74 imágenes con más de 380 piezas de LEGO. Y con el fin de obtener un análisis riguroso y que refleje la capacidad de este método ante diversas condiciones, se han seleccionado fotos con múltiples ángulos y escenas. A continuación, se muestra algunas de las imágenes usadas para la evaluación:

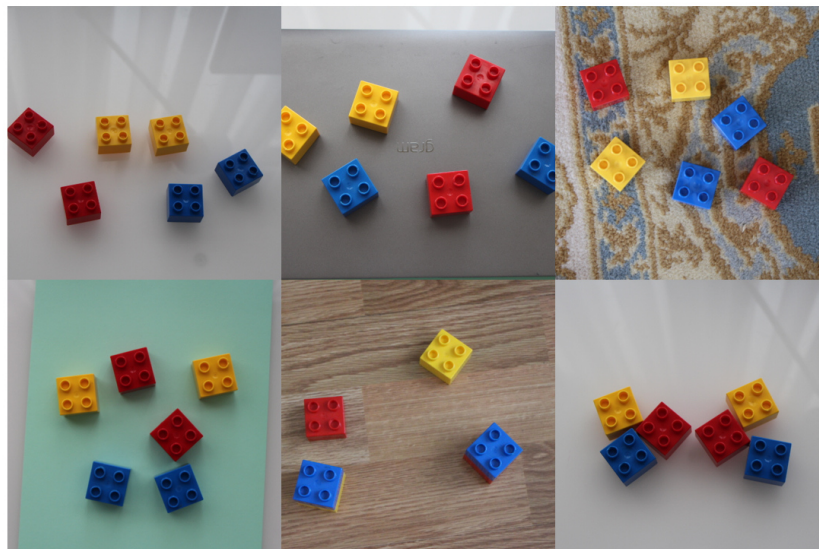


Figura 3.10. Muestra de imágenes para la evaluación del proceso de segmentación

Con la ayuda del *script* se han podido obtener cuatro parámetros que permiten evaluar los resultados. En primer lugar, se ha obtenido la tasa de fallos y el número de falsos positivos por imagen para cada una de las clases. Y a continuación se ha evaluado la precisión de las piezas encontradas y la exhaustividad, que es la relación entre verdaderos positivos y la suma de verdaderos positivos y falsos negativos.

Tras analizar los resultados se ha llegado a las siguientes conclusiones:

- **Amarillo:** Del total de piezas, estas han sido las que mejor han sido detectadas. Presentan una precisión muy elevada y una tasa de fallos muy pequeña. Las formas de las gráficas se deben a que este método apenas ha tenido falsos positivos y cuando estos han ocurrido estaban muy controlados. Es por ello que la gráfica es tan brusca y con tan pocos puntos.
- **Rojo:** La tasa de fallos es bastante superior respecto al amarillo y la precisión también baja bastante. Esto de nuevo se debe al filtro de color. En los resultados se han dado numerosos falsos positivos y en numerosas ocasiones ha habido piezas rojas que no se

han detectado. Esto explica los valores de exhaustividad obtenidos y el porqué de que aunque la exhaustividad no sea muy elevado, la precisión es muy baja.

- Azul: El azul sufre de los mismos problemas que el rojo. Hay demasiados falsos positivos y múltiples situaciones en las que a pesar de haber falsos positivos, no se han detectado las piezas azules.

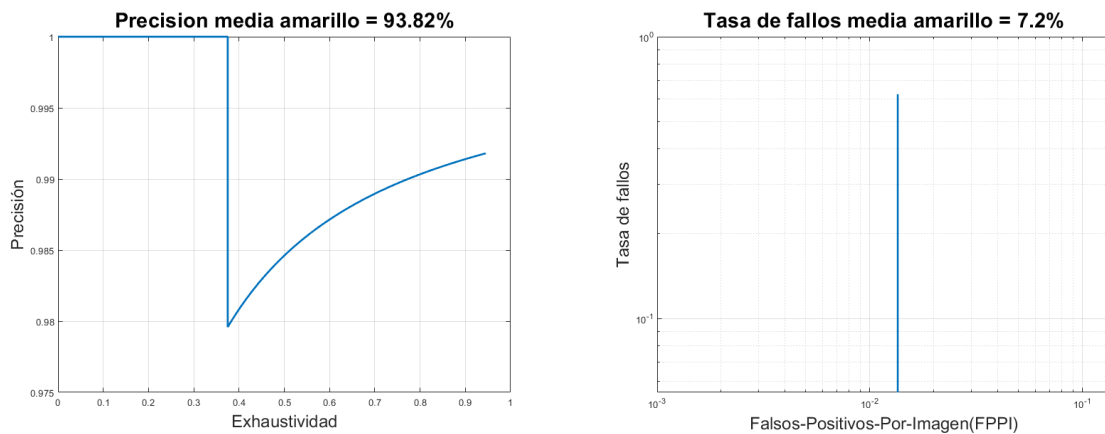


Figura 3.11. Estudio de la segmentación por color al detectar piezas amarillas

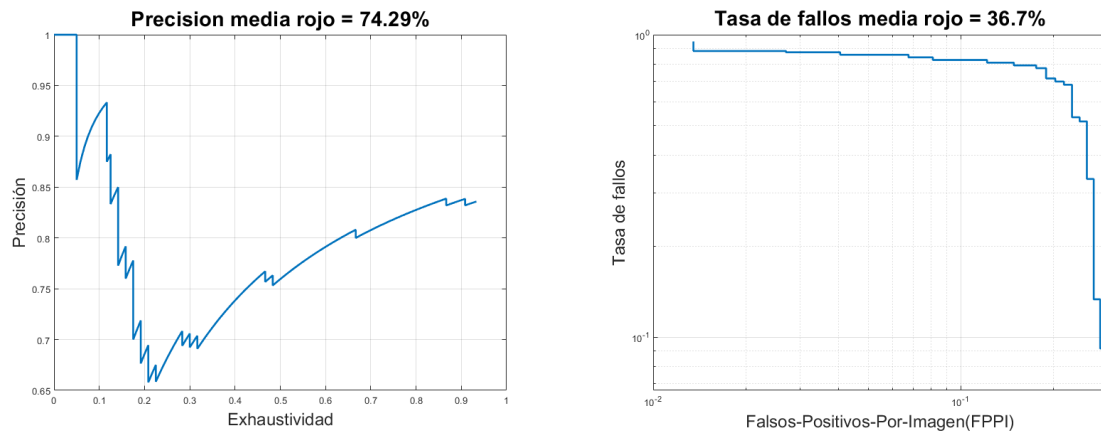


Figura 3.12. Estudio de la segmentación por color al detectar piezas rojas

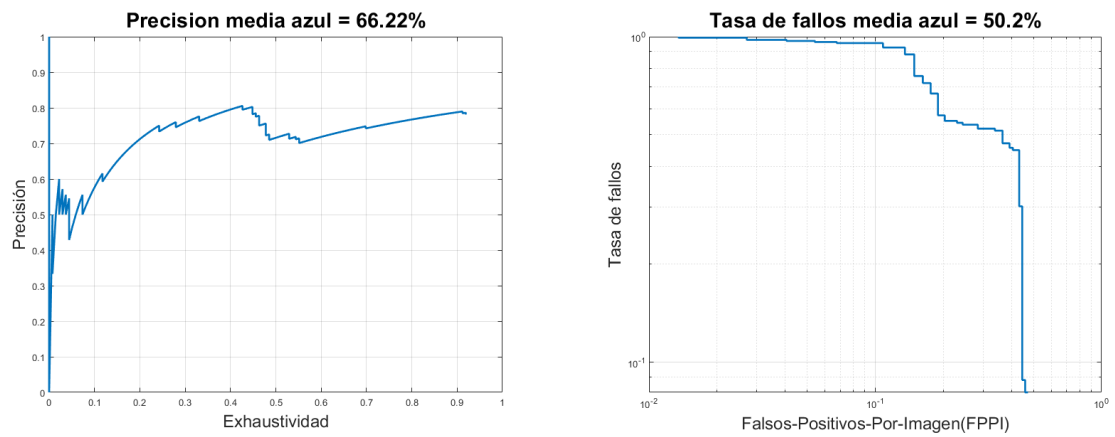


Figura 3.13. Estudio de la segmentación por color al detectar piezas azules

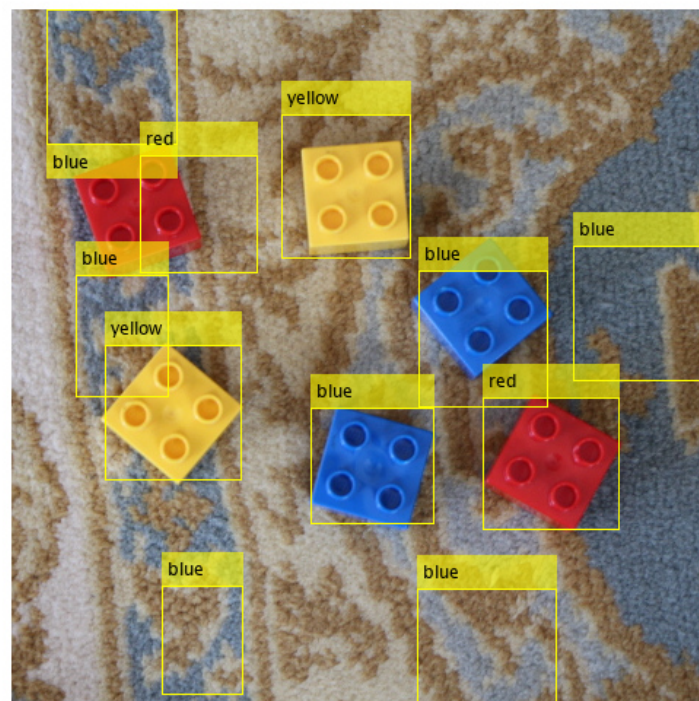


Figura 3.14. Error en la evaluación de segmentación por color

4

Clasificación con redes neuronales

El presente capítulo describe de forma detallada el proceso de análisis de imágenes mediante clasificadores basados en redes neuronales. Se explica de forma detallada el proceso de creación de los clasificadores y se muestra sus capacidades.

Los clasificadores son redes neuronales diseñadas con el objetivo de identificar. Estas son capaces de indicar si un objeto está en una imagen o no. En este proyecto identificar los objetos no es suficiente ya que también hace falta localizarlos. A pesar de ello, primero se van a entrenar dos clasificadores con el objetivo de que sirvan como base para el desarrollo de los detectores de objetos basados en técnicas one-shot (Capítulo 5).

Por falta tanto de datos como de cálculo computacional, se va a partir de dos clasificadores ya entrenados y se van a reentrenar. Este proceso es conocido como aprendizaje por transferencia y es ampliamente usado al trabajar con redes neuronales. De esta forma podemos partir de dos clasificadores con estructuras ampliamente corroboradas a la vez que se consigue reducir la carga computacional y con ello el tiempo de entrenamiento. Además, estos clasificadores ya han sido entrenados para la extracción de características lo cual ayudará al sistema a pesar del poco tiempo de entrenamiento. Para llevar a cabo el reentrenamiento primero se han descongelado las capas de convolución para permitir así el aprendizaje. Los dos clasificadores elegidos para ser reentrenados son AlexNet y VGG-16. Los motivos de su elección son: Se caracterizan por redes simples y directas lo cual facilita su comprensión y el trabajo con ellas. Reflejan la evolución de las redes neuronales durante los últimos años. Los dos nuevos clasificadores han sido nombrados como LEGONet y LEGO16 respectivamente. De ahora en adelante se referirá a ellos de esta forma.

4.1. Condiciones de entrenamiento

Para poder reentrenar dos clasificadores basados en redes neuronales es necesario disponer de una buena base de datos de la que puedan aprender. Para ello se ha empleado el conjunto

de imágenes clasificadas por Intel [Inta], el conjunto de imágenes de LEGOS clasificadas por Joost Hazelzet [Haz] y el conjunto de imágenes de LEGOS clasificadas por Francisco García [Gar]. Con la ayuda de estas bases de datos se dispone de un total de 7 clases sobre las que entrenar y un total de 16.398 imágenes de alta resolución. En la Tabla 4.1 se puede ver un resumen de las imágenes usadas. Y en la Figura 4.1 se pueden ver algunos ejemplos de las imágenes empleadas para el entrenamiento. Se ha decidido emplear la base de datos de Intel ya que disponía de imágenes de alta resolución y al contar con varias estructuras, edificios y calles se dispone de numerosas imágenes con ángulos rectos y secciones. Característica en común con las piezas de LEGO.

Total de imágenes	
LEGOS	2464
Edificios	2191
Bosques	2171
Glaciares	2404
Montañas	2512
Océanos	2274
Calles	2382

Tabla 4.1. Base de datos para el entrenamiento de clasificadores

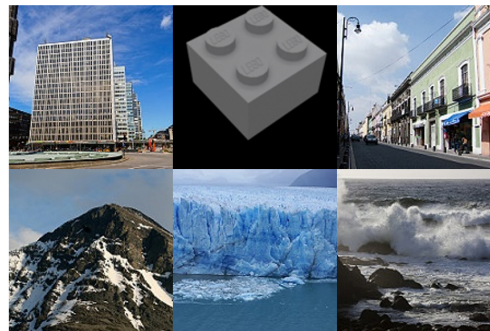


Figura 4.1. Muestra de imágenes para el entrenamiento de clasificadores

Todo el entrenamiento se ha llevado a cabo en un ordenador personal equipado con un i7 4790K, 16GB de memoria RAM y una tarjeta gráfica Nvidia Geforce GTX970 con 4GB de VRAM. Teniendo en cuenta las limitaciones por *hardware* y tiempo, se han realizado múltiples entrenamientos con diferentes opciones de entrenamiento para obtener los mejores resultados con cada red.

4.2. LEGONet

Originalmente AlexNet fue diseñado para trabajar con la base de datos de ImageNet, que cuenta con más de 14 millones de fotos y 1000 clases diferentes. Para este proyecto se ha decidido reentrenar esta red neuronal con imágenes de LEGOS para mejorar su reconocimiento de estos y así usarla posteriormente como base para los detectores de objetos. Por ello, con la ayuda de MATLAB y del conjunto de imágenes clasificadas por Intel [Inta], el conjunto de imágenes de LEGOS clasificadas por Joost Hazelzet [Haz] y el conjunto de imágenes de LEGOS clasificadas por Francisco García [Gar] se ha reentrenado AlexNet para clasificar un total de 7 clases diferentes.

4.2.1. Estructura

En el 2012, AlexNet se caracterizó por ser una red neuronal muy grande y pesada para la época. Está formada por 60 millones de parámetros y un total de 650.000 neuronas. Al contrastarlo con sistemas actuales puede no parecer tan grande, pero dada la época y las limitaciones por hardware para entrenarla, fue un gran salto. La estructura de la red se divide en 5 capas convoluciones y dos capas completamente conectadas. Las capas convolucionales se unen entre si con capas *Relu*, *Batch normalization* y *Max Pooling*. Las capas de activación *Relu* se usan para evitar la presencia de valores nulos y negativos que puedan entorpecer el aprendizaje, su función es $ReLU = \max(0, x)$ (ver Figura 4.2) [Aga18]. Existen múltiples funciones de activación además de *Relu*, pero esta es altamente usada, ya que apenas supone carga computacional y permite que en grandes redes, los errores se puedan propagar y las capas más profundas puedan aprender. *Batch normalization* tal y como su nombre indica, regulan y normalizan el aprendizaje de forma que evitan un aprendizaje repentino y brusco y reducen la posibilidad de un sobreaprendizaje. Además, permiten que las capas aprendan de forma independiente del resto de la red agilizando así el entrenamiento. Y por último, las capas *Max Pooling* se emplean para reducir las dimensiones de las capas intermedias y contener así a las red neuronal.

Las capas completamente conectadas están también conectadas entre sí con capas *Relu* y *dropout*. Las capas *dropout* sirven para evitar el sobre aprendizaje, es decir, evitar que la red se aprenda los casos en lugar de las características. Para ello lo que se hace es que todas las neuronas tienen una probabilidad previamente establecida de ser desconectadas. En AlexNet esta probabilidad es del 50 %. De esta forma se controla el sobre aprendizaje.

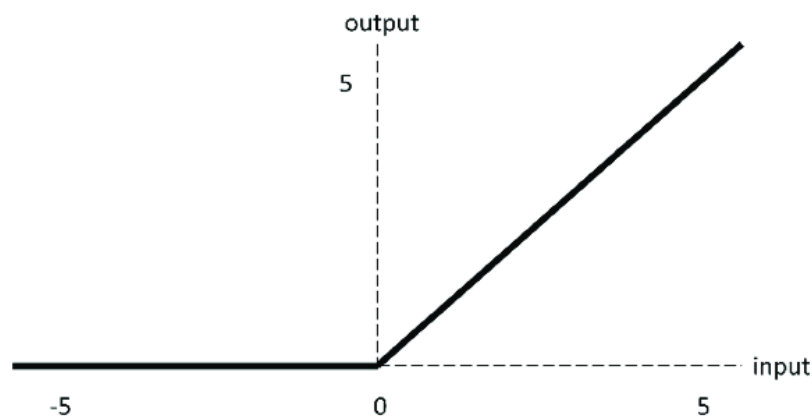


Figura 4.2. Funcionamiento de *Relu*(Rectified Linear Unit)

Para reentrenar AlexNet con las nuevas clases es necesario modificar lo ya que este ha sido diseñado para reconocer 1000 clases en lugar de 7. EL primer paso consiste en el descongelamiento de las capas de convolución para que así la red pueda aprender. A continuación, es necesario eliminar las tres últimas capas que se encargan de la clasificación y sustituirlas por capas similares, pero de una correcta dimensión para detectar las siete clases. Los cambios a aplicar son:

- Capa 23: Completamente conectada (1x1x1000) → (1x1x7)
- Capa 24: *Softmax* (1x1x1000) → (1x1x7)
- Capa 23: Salida clasificador

A continuación se muestra la estructura final en la Figura 4.3 y la activación de la primera capa convolucional en la Figura 4.4.

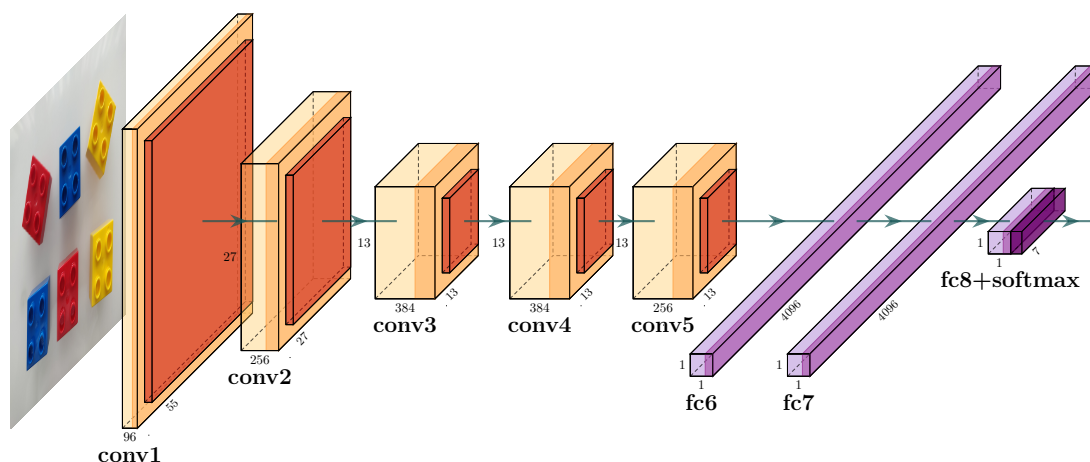


Figura 4.3. Estructura de LEGONet

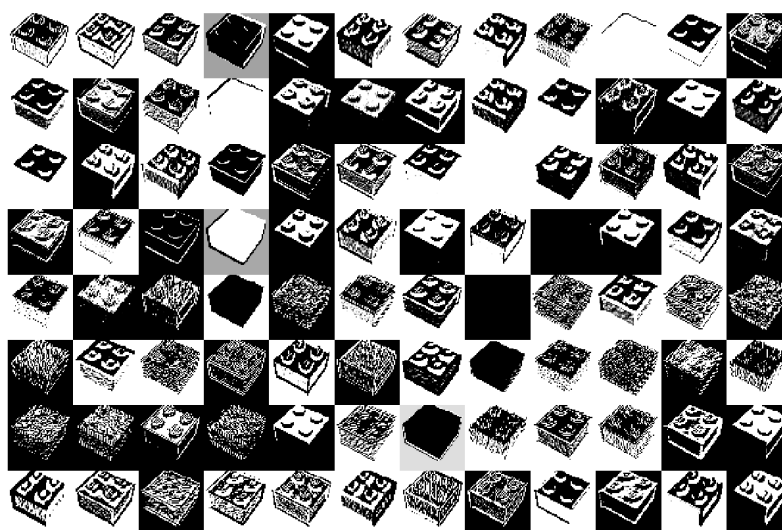


Figura 4.4. Activación primera convolución de LEGONet

4.2.2. Entrenamiento

Con la ayuda de MATLAB y del conjunto de imágenes elaborado en Sección 4.1 se ha reentrenado a AlexNet para identificar 7 clases. Tras numerosas pruebas se han seleccionado las opciones de entrenamiento mostradas en la Tabla 4.2.

4.2.3. Resultados

Con la ayuda de MATLAB y de las bases de datos previamente comentadas, se va a realizar una evaluación del clasificador. En total se han empleado más de 3000 imágenes y a continuación se muestran los resultados obtenidos en forma de matriz de confusión (ver Figura 4.5).

Analizando estos resultados se puede obtener las tasas de acierto y tasas de fallo y si se realiza un estudio más intensivo analizando los verdaderos positivos y los falsos positivos

Opciones de entrenamieto	
Solver	Stochastic Gradient Descent with Momentum (SGDM)
Momentum	0.9
Initial Learn Rate	1.00E-04
Learn Rate Schedule	piecewise
Learn Rate Drop Factor	0.1
Learn Rate Drop Period	5
L2Regularization	0.004
Max Epochs	30
Mini Batch Size	128
Shuffle data	every epoch

Tabla 4.2. Opciones de entrenamiento de LEGONet

(ver Figura 4.6). Este tipo de representación se conoce como curva *ROC* (*Receiver Operating Characteristic* o Característica Operativa del Receptor) y es ampliamente usada al evaluar clasificadores ya que permite mostrar de forma gráfica el comportamiento de la red. Otro dato importante de esta curva es el área que encierra, este dato es conocido como *AUC* (*Area Under the Curve*) y es muy representativo de las capacidades de la red para identificar. Este dato se puede ver en la Tabla 4.3.

Las tasas de acierto y las ACUs obtenidas es bastante elevada si se tiene en cuenta el poco tiempo de entrenamiento y las circunstancias dadas. Si nos fijamos solo en los LEGOS observamos que los resultados son incluso mejores ya que no ha habido ninguna pieza mal identificada y solo se han identificado mal como LEGOs tres imágenes. Es decir, una tasa de acierto al identificar LEGOS del 100 % y una tasa de falsos positivos para LEGO del 0.7 %.

<i>AUC</i>	
LEGOS	1.00
Edificios	0.9902
Bosques	0.9993
Glaciares	0.9816
Montañas	0.9856
Océanos	0.9962
Calles	0.9935

Tabla 4.3. Evaluación de LEGONet: áreas encerradas debajo de la curva *ROC* para cada clase

Matriz de confusión

Predicción	LEGO	433 14.3%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	1 0.0%	0 0.0%	99.8% 0.2%
	buildings	0 0.0%	400 13.2%	0 0.0%	2 0.1%	0 0.0%	1 0.0%	19 0.6%	94.8% 5.2%
	forest	0 0.0%	1 0.0%	431 14.2%	1 0.0%	0 0.0%	1 0.0%	1 0.0%	99.1% 0.9%
	glacier	0 0.0%	0 0.0%	1 0.0%	378 12.5%	34 1.1%	4 0.1%	0 0.0%	90.6% 9.4%
	mountain	0 0.0%	0 0.0%	1 0.0%	41 1.4%	392 12.9%	2 0.1%	0 0.0%	89.9% 10.1%
	sea	0 0.0%	2 0.1%	0 0.0%	9 0.3%	6 0.2%	424 14.0%	2 0.1%	95.7% 4.3%
	street	0 0.0%	30 1.0%	0 0.0%	2 0.1%	1 0.0%	0 0.0%	411 13.6%	92.6% 7.4%
		100% 0.0%	92.4% 7.6%	99.5% 0.5%	87.3% 12.7%	90.5% 9.5%	97.9% 2.1%	94.9% 5.1%	94.7% 5.3%
		LEGO	buildings	forest	glacier	mountain	sea	street	
		Imagen							

Figura 4.5. Evaluación de LEGONet: matriz de confusión

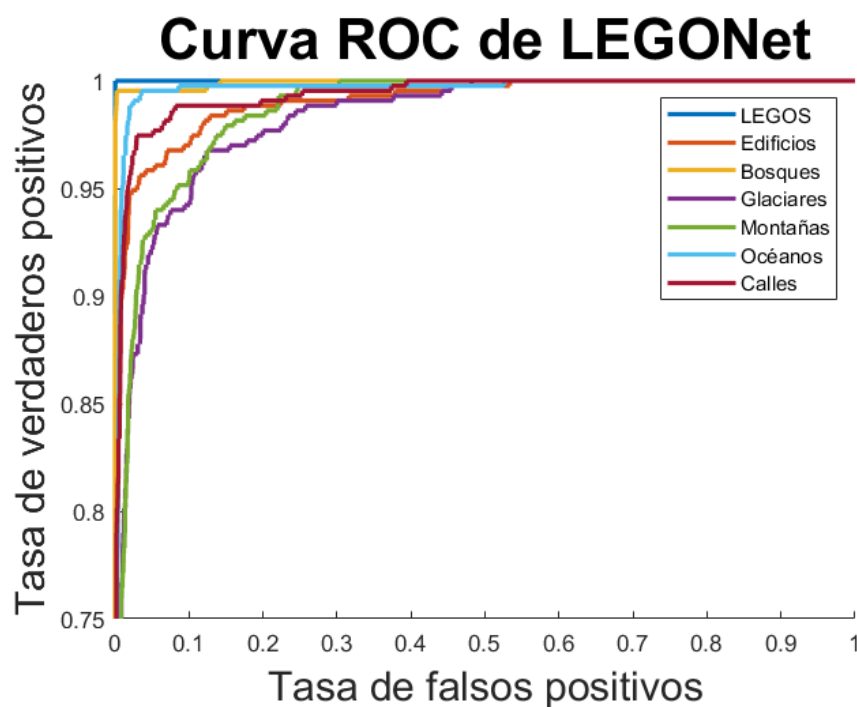


Figura 4.6. Evaluación de LEGONet: curvas ROC

4.3. LEGO16

En el 2014 VGG-19 y VGG-16 ganaron el primer y segundo puesto de ImageNet y en el 2015 fueron publicadas por Karen Simonyan y Andrew Zisserman en su artículo "*Very Deep Convolutional Networks For Large-Scale Image Recognition*" [SZ14]. Tal y como su nombre indica, estas redes se caracterizan por tener 19 y 16 capas con pesos respectivamente. En este proyecto dadas las limitaciones de hardware y la simplicidad de los objetos a detectar, se ha decantado por el uso de VGG-16 frente a VGG-19.

Para poder emplear VGG-16 ha sido necesario modificarla reemplazando las últimas capas y reentrenarla con varias bases de datos. Esta nueva red se ha nombrado LEGO16.

4.3.1. Estructura

Como su nombre indica, esta red qse caracteriza por tener 16 capas con pesos, un total de 138 millones de parámetros y 13 millones de neuronas. Estas se reparten en 13 capas convolucionales y 3 capas completamente conectadas. Las capas convolucionales siguen una arquitectura definida donde detrás de cada convolución hay una capa *Relu* y a su vez las convoluciones se agrupan en dos grupos de dos y tres grupos de tres. Al final de cada grupo de convoluciones hay una capa *Max Pooling* para reducir la dimensión. Destaca la inexistencia de las capas *Batch normalization* presentes en AlexNet.

Las capas completamente conectadas presentan una arquitectura idéntica a AlexNet por lo que cada capa va acompañada por un capa *Relu* y una capa *dropout*.

Para reentrenar VGG-16 con las nuevas clases es necesario modificar lo ya que este ha sido diseñado para reconocer 1000 clases en lugar de 7. Es necesario eliminar las tres últimas capas que se encargan de la clasificación y sustituirlas por capas similares, pero de una correcta dimensión para detectar las siete clases. Los cambios a aplicar son:

- Capa 23: Completamente conectada (1x1x1000) $\rightarrow$ (1x1x7)
- Capa 24: *Softmax* (1x1x1000) $\rightarrow$ (1x1x7)
- Capa 23: Salida clasificador

A continuación se muestra la estructura final en la Figura 4.7 y la activación de la primera capa convolucional en la Figura 4.8.

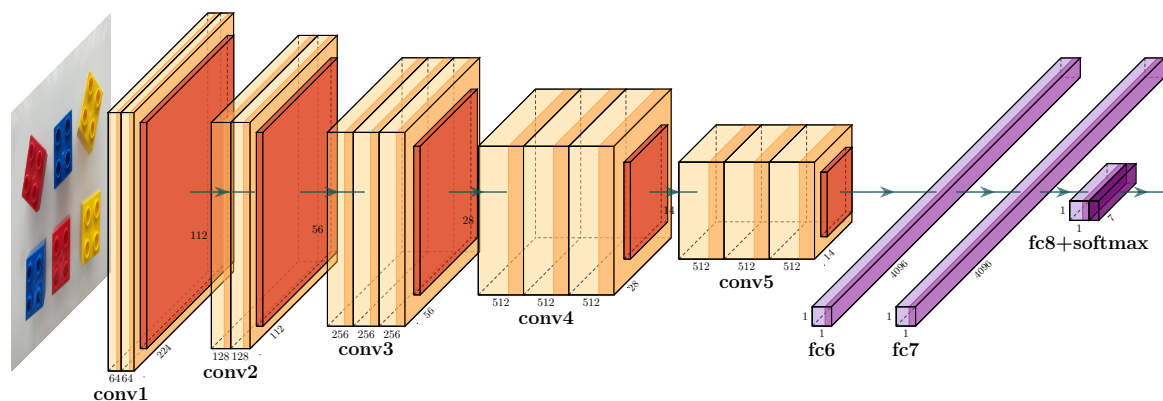


Figura 4.7. Estructura de LEGO16

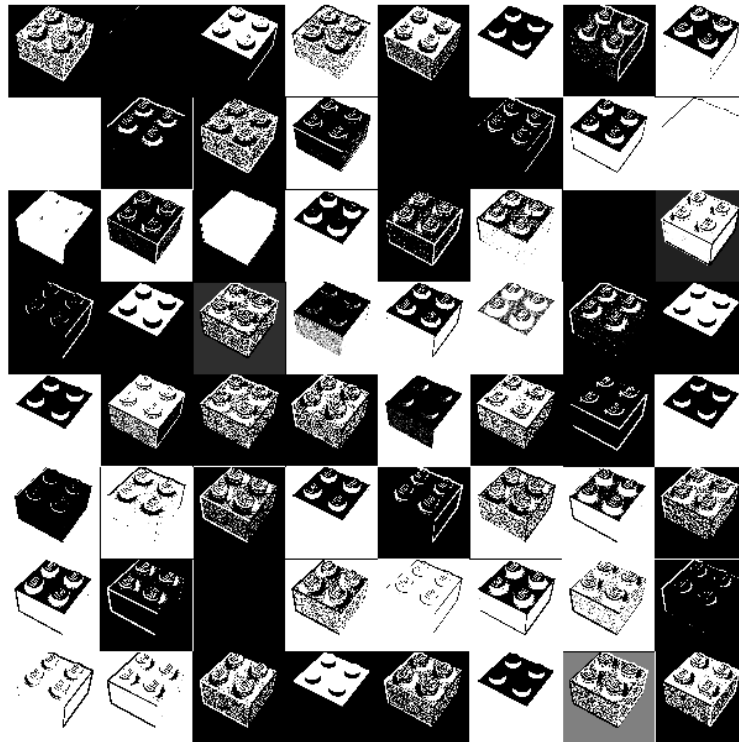


Figura 4.8. Activación de la primera convolución de LEGO16

4.3.2. Entrenamiento

Con la ayuda de MATLAB y del conjunto de imágenes elaborado en Sección 4.1 se ha reentrenado a VGG-16 para identificar 7 clases. Tras numerosas pruebas se han seleccionado las opciones de entrenamiento mostradas en la Tabla 4.4.

Opciones de entrenamineto	
Solver	Stochastic Gradient Descent with Momentum (SGDM)
Momentum	0.9
Initial Learn Rate	1.00E-04
Learn Rate Schedule	piecewise
Learn Rate Drop Factor	0.1
Learn Rate Drop Period	2
L2Regularization	0.004
Max Epochs	10
Mini Batch Size	16
Shuffle data	every epoch

Tabla 4.4. Opciones de entrenamiento de LEGO16

4.3.3. Resultados

Con la ayuda de MATLAB y de las bases de datos previamente comentadas, se va a realizar una evaluación del clasificador. En total se han empleado más de 3000 imágenes y a continuación se muestran los resultados obtenidos. Analizando estos resultados se puede obtener las tasas de aciertos y tasas de fallos (ver Figura 4.9).

Se puede realizar un estudio más intensivo analizando los verdaderos positivos y los falsos positivos (ver Figura 4.10). Este tipo de representación se conoce como curva *ROC* (curva de característica operativa del recepto) y es ampliamente usada al evaluar clasificadores ya que permite mostrar de forma gráfica el comportamiento de la red. Otro dato importante de esta curva es el área que encierra, este dato es conocido como *AUC* (Area Under the Curve). Este dato se puede ver en la Tabla 4.5.

Las tasas de acierto y las *AUC* obtenidas es bastante elevada si se tiene en cuenta el poco tiempo de entrenamiento. Si nos fijamos solo en los LEGOS observamos que los resultados son incluso mejores ya que solo ha habido un falso positivos y ninguna pieza mal identificada. Es decir, una tasa de falsos positivos es del 0.2 % y una tasa de acierto al identificar LEGOS del 100 %.

En general, este clasificador ha dado mejores resultados frente a LEGONet reduciendo los errores cometidos entre clases. Aunque, en la clase LEGO ambos han obtenido una tasa de acierto del 100 % pero LEGO16 ha conseguido reducir el número de falsos positivos. Probablemente se puedan obtener mejores resultados si se consigue entrenar durante más tiempo a la red, con más datos y con mayor VRAM. De esta forma se pueden evitar problemas por regularización aumentado el tamaño del *batch*.

<i>AUC</i>	
LEGOS	1.00
Edificios	0.9927
Bosques	1.00
Glaciares	0.9910
Montañas	0.9928
Océanos	0.9992
Calles	0.9946

Tabla 4.5. Evaluación de LEGO16: áreas encerradas debajo de la curva *ROC* para cada clase

Matriz de confusión

Predicción	LEGO	433 14.3%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	1 0.0%	0 0.0%	99.8% 0.2%
	buildings	0 0.0%	400 13.2%	0 0.0%	2 0.1%	0 0.0%	1 0.0%	19 0.6%	94.8% 5.2%
	forest	0 0.0%	1 0.0%	431 14.2%	1 0.0%	0 0.0%	1 0.0%	1 0.0%	99.1% 0.9%
	glacier	0 0.0%	0 0.0%	1 0.0%	378 12.5%	34 1.1%	4 0.1%	0 0.0%	90.6% 9.4%
	mountain	0 0.0%	0 0.0%	1 0.0%	41 1.4%	392 12.9%	2 0.1%	0 0.0%	89.9% 10.1%
	sea	0 0.0%	2 0.1%	0 0.0%	9 0.3%	6 0.2%	424 14.0%	2 0.1%	95.7% 4.3%
	street	0 0.0%	30 1.0%	0 0.0%	2 0.1%	1 0.0%	0 0.0%	411 13.6%	92.6% 7.4%
		100% 0.0%	92.4% 7.6%	99.5% 0.5%	87.3% 12.7%	90.5% 9.5%	97.9% 2.1%	94.9% 5.1%	94.7% 5.3%
		Imagen							
		LEGO	buildings	forest	glacier	mountain	sea	street	

Figura 4.9. Evaluación de LEGO16: matriz de confusión

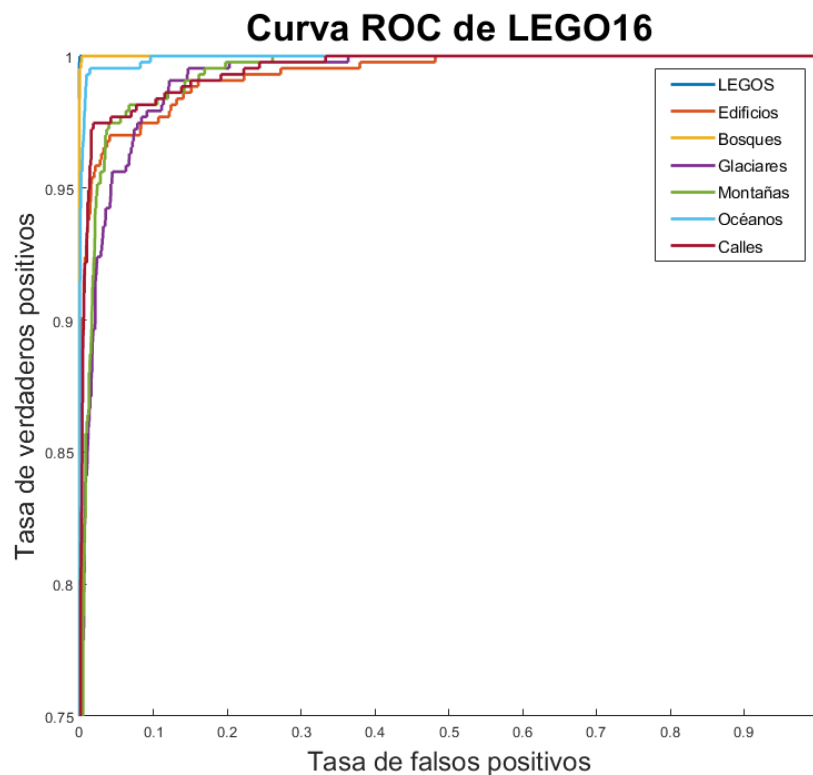


Figura 4.10. Evaluación de LEGO16: curvas ROC

5

Segmentación con redes neuronales

En este capítulo se van a desarrollar tres técnicas diferentes para la detección de piezas LEGO. Las técnicas desarrollados son R-CNN, Faster R-CNN y YOLO. A su vez, para cada una de estas se desarrollaran dos redes basadas en LEGONet y LEGO16.

El objetivo final de este proyecto es la implantación de un sistema de detección de piezas de LEGO de forma que un brazo robótico pueda identificar y recoger dichas piezas. En el Capítulo 3 ya se ha desarrollado un método capaz de realizar dicha tarea pero debido a la tecnología empleada, el sistema presenta demasiados falsos positivos y solo es capaz de trabajar bajo condiciones ideales. Por ello se ha decidido emplear las últimas tecnologías para la detección de objetos en imágenes. Y en la actualidad los detectores de objetos más modernos se basan en redes neuronales convolucionales. Para su desarrollo es común y recomendado partir de un clasificador ya entrenado y reusarlo con el propósito de convertirlo en un detector de objetos.

En el Capítulo 4 se ha desarrollado dos clasificadores diferentes con el fin de ser usados para el desarrollo de múltiples detectores de imágenes. Se van a desarrollar un total de seis detectores de objetos empleando diferentes tecnologías y se compararan entre sí. Para facilitar el desarrollo del proyecto, las redes neuronales se han desarrollado de forma escalonada en función de su nivel de complejidad. Empezando por la red más simple y terminando por la red más compleja. Por nivel de complejidad nos referimos a la complejidad de las tecnologías empleadas y a su dificultad de comprensión y no al número de capas que constituyen la red neuronal. Pero antes de desarrollar estas redes, es necesario contextualizar el proyecto. A continuación, se va a mostrar de que datos se parte y como se van a entrenar dichas redes neuronales.

5.1. Preparativos para el entrenamiento y evaluación

Para poder entrenar detectores de objetos basado en clasificadores es necesario disponer de una buena base de datos de la que puedan aprender y ser evaluados. En esta sección se van a desarrollar ambas bases de datos.

5.1.1. Entrenamiento

Para el desarrollo de este proyecto se va a emplear aprendizaje supervisado lo cual implica que toda la información debe haber sido preparada, etiquetada y analizada para que la red pueda aprender de esta. En el caso de los detectores de objetos esta información consta de dos objetos. En primer lugar, la imagen de la que se desea aprender y en segundo lugar, las posiciones de los objetos que debe de detectar y el tipo de objeto que es. Esta posición se indica en forma de *bounding box*, es decir, se dan la posición y dimensión de un rectángulo que contiene dicho objeto. Los *bounding boxes* se dan en píxeles con forma de un vector de dimensión cuatro. Se da la posición en ejes x e y de la esquina superior izquierda y el alto y ancho del rectángulo.

Con la ayuda de MATLAB e *imageLabeler* se ha preparado un conjunto de imágenes clasificadas y etiquetadas para llevar acabo el entrenamiento de las redes neuronales. En total se han preparado y etiquetado un total de 346 imágenes con un total de 981 piezas de LEGO. A continuación, se muestra en la Figura 5.1 algunas de las imágenes empleadas para el entrenamiento.

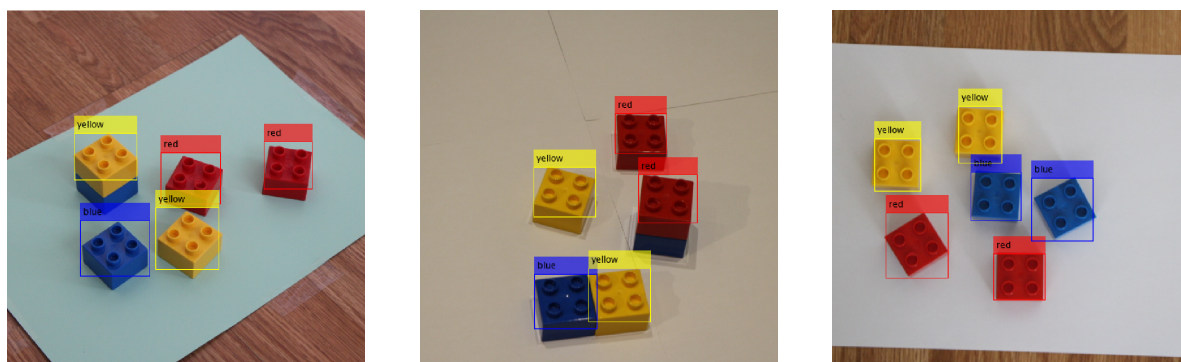


Figura 5.1. Imágenes empleadas para el entrenamiento de segmentación con redes neuronales

Para el entrenamiento de redes neuronales es recomendable tener una buena y amplia base de datos sobre la que la red pueda aprender. Por ello, para aumentar aún más el número de imágenes disponibles, se ha acudido a diversas herramientas.

- Transformación del color: se ha generado simultáneamente variaciones del contraste, matiz y brillo de forma aleatoria pero dentro de unos límites. Al aplicar estas modificaciones a las imágenes originales se consigue variar notablemente la imagen respecto a la original.
- Espejo: se han reflejado las imágenes de tres formas posibles. Se han reflejado en el eje x, el eje y y respecto a ambos ejes a la vez.
- Recortes: se han realizado de forma aleatorio, pero dentro de unos límites, recortes de las imágenes originales para crear nuevas.

- Deformaciones: se han generado pequeñas deformaciones de forma de las imágenes originales para crear nuevas. Las deformaciones se encuentran dentro de unos límites.

Aplicando estas técnicas, se ha conseguido aumentar notablemente en el número de imágenes pasando de 346 a 2.076 y de 981 piezas de LEGO a 5.886. Es decir, se ha conseguido incrementar seis veces el número de imágenes y de piezas.

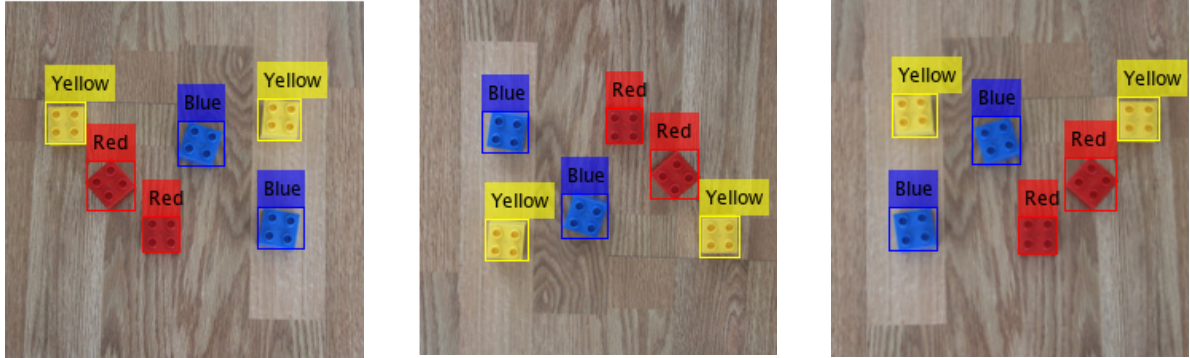


Figura 5.2. Aumento del número de imágenes para el entrenamiento

Total de piezas	
Rojo	2106
Azul	1818
Amarillo	1962

Tabla 5.1. Base de datos para el entrenamiento de detectores de objetos

Todo el entrenamiento se ha llevado a cabo en un ordenador personal equipado con un i7 4790K, 16GB de memoria RAM y una tarjeta gráfica Nvidia Geforce GTX970 con 4GB de VRAM. Teniendo en cuenta las limitaciones por *hardware* y tiempo, se han realizado múltiples entrenamientos con diferentes opciones de entrenamiento para obtener los mejores resultados con cada red.

5.1.2. Evaluación

Para comprobar la eficacia de las redes creadas, se ha desarrollado un *script* con la ayuda de MATLAB para determinar su capacidad de para identificar piezas y la precisión con las que la detecta. En este *script* se han analizado 74 imágenes con más de 380 piezas de LEGO. Y con el fin de obtener un análisis riguroso y que refleje la capacidad de este método ante diversas condiciones, se han seleccionado fotos con múltiples ángulos y escenas. A continuación, se muestra algunas de las imágenes usadas para la evaluación:

Con la ayuda del *script* se han podido obtener cuatro parámetros que permiten evaluar los resultados. En primer lugar, se ha obtenido la tasa de fallos y el número de falsos positivos por imagen para cada una de las clases. Y a continuación se ha evaluado la precisión de las piezas encontradas y la exhaustividad, que es la relación entre verdaderos positivos y la suma de verdaderos positivos y falsos negativos.

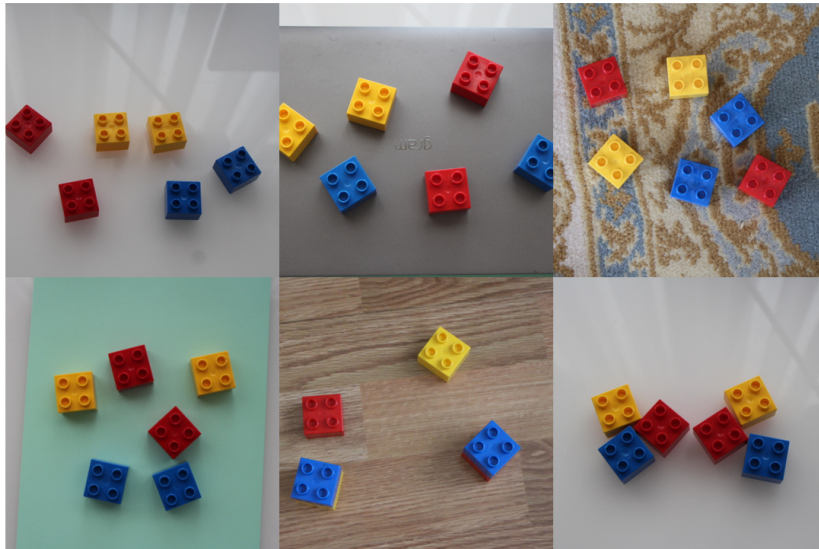


Figura 5.3. Muestra de imágenes para la evaluación de redes neuronales

5.2. R-CNN

Tal y como se ha explicado en la Sección 2.1.2, R-CNN (*Region-proposal Convolutional Neuronal Network*) es un sistema basado en un clasificador y el sistema de propuesta de regiones. La idea surgió porque previamente a este sistema, la solución para poder detectar objetos consistía en correr un clasificador por toda la imagen variando el tamaño del clasificador. Este método se conoce como ventana flotante y suponía una gran carga computacional ya que se tenía que correr el clasificador numerosas veces para analizar una sola imagen. Pero surgió el método R-CNN [Gir+13], este consiste en la propuesta de 2000 zonas con posibilidades de contener un objeto. De esta forma se limitaba el número de regiones mejorando así el rendimiento.

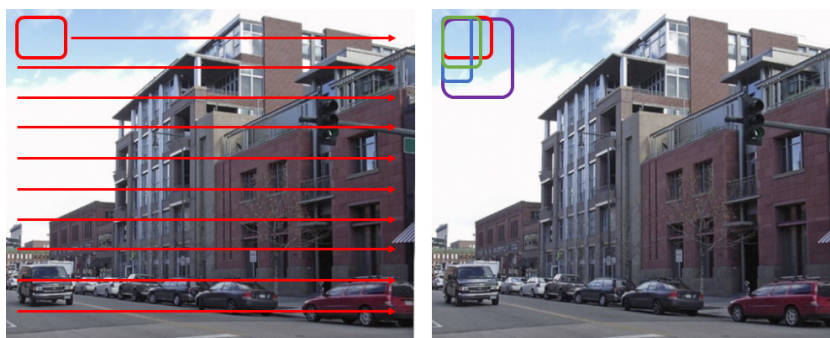


Figura 5.4. Demostración del concepto de ventana flotante (Fuente: [Tsa])

El proceso de propuesta de regiones se basa en diferentes características de la imagen. Se intenta buscar similitudes de color, texturas, tamaños.... Una vez analizada toda la imagen y obtenidas todas las posibles regiones, estas se analizan y se intenta unir las regiones pequeñas y próximas de forma que se termine con aproximadamente 2000 regiones a analizar.

Con el objetivo de poder comparar diversas redes neuronales y su evolución en el paso del tiempo, se han desarrollado dos redes tipo R-CNN basadas en dos clasificadores diferentes. Se

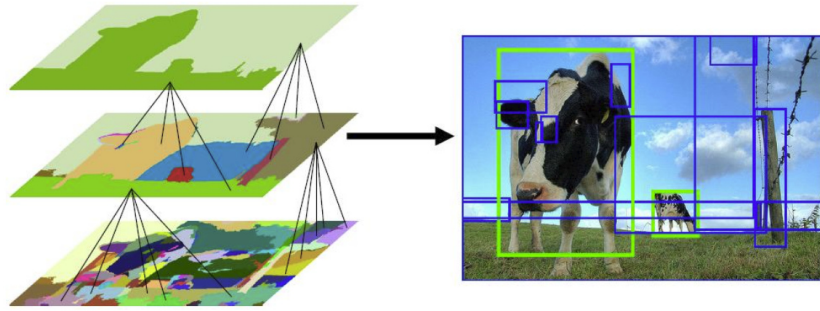


Figura 5.5. Estimación de las regiones de interés con R-CNN (Fuente: [Tsa])

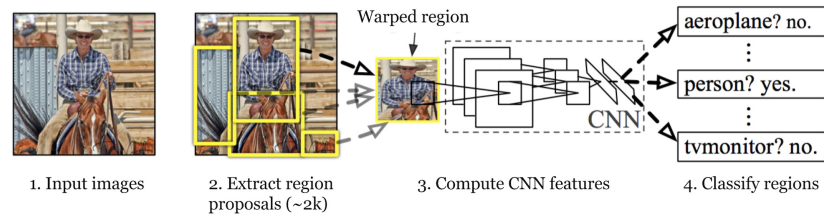


Figura 5.6. Representación del funcionamiento de R-CNN (Fuente: [Gir+13])

han usado los clasificadores creados en el Capítulo 4 que a su vez están basados en AlexNet y VGG-16.

5.2.1. R-CNN basado en LEGONet

Partiendo del clasificador LEGONet y con la ayuda de MATLAB y una base de datos de imágenes de LEGOS, se ha creado y entrenado un detector de objetos tipo R-CNN.

Estructura

LEGONet esta formada por un total de 25 capas, 60 millones de parámetros y 650.000 neuronas. De las 25 capas, 5 son capas convoluciones, 2 son capas completamente conectadas y entre medias se encuentran capas del tipo *ReLU*, *Batch normalization*, *Max pooling* y *Dropouts*. Para poder ser reutilizado y empleado como detector de objetos, es necesario modificar la estructura y el funcionamiento de la red neuronal.

Como se puede ver en la Figura 5.6, R-CNN es un proceso definido por dos etapas, la primera etapa es la propuesta de regiones y la segunda es el análisis de dichas regiones con el clasificador. Esto implica que a LEGONet se le va a tener que incluir el proceso de propuesta de regiones y de nuevo, se van a tener que modificar las tres últimas capas para reentrenar al clasificador para distinguir las piezas de LEGO.

- Capa 23: Completamente conectada $(1 \times 1 \times 7) \rightarrow (1 \times 1 \times 4)$
- Capa 24: *Softmax* $(1 \times 1 \times 1000) \rightarrow (1 \times 1 \times 4)$
- Capa 23: Salida del clasificador

La dimensión de estas nuevas capas es cuatro ya que R-CNN necesita también incluir la clase fondo para el caso en que no detecte nada en alguna de las regiones propuestas. Con estos cambios, la nueva estructura se puede ver en la Figura 5.7

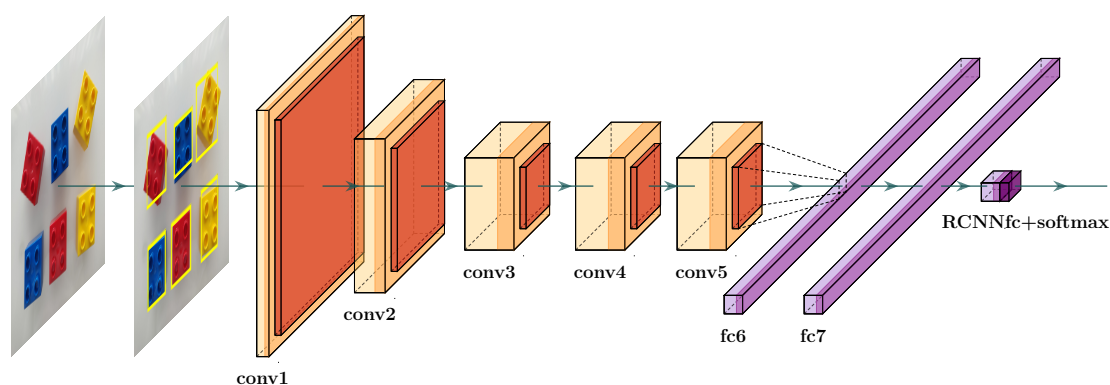


Figura 5.7. Estructura de R-CNN basado en LEGONet

Entrenamiento

Para el entrenamiento de esta red se ha usado la base de datos diseñada en la Sección 5.1. Esta base de datos consta de un total de 2.076 imágenes con un total de 5.886 piezas de LEGO. Se han realizado numerosas pruebas de entrenamiento y se ha descubierto que las mejores opciones de entrenamiento son las mostradas en la Tabla 5.2.

Opciones de entrenamineto	
Solver	Stochastic Gradient Descent with Momentum (SGDM)
Momentum	0.9
Initial Learn Rate	1.00E-04
Learn Rate Schedule	piecewise
Learn Rate Drop Factor	0.5
Learn Rate Drop Period	35
L2Regularization	0.004
Max Epochs	400
Mini Batch Size	200
Shuffle data	every epoch

Tabla 5.2. Opciones de entrenamiento de R-CNN basado en LEGONet

Resultados

Con la ayuda del *script* definido en la Sección 5.1.2, se han podido obtener cuatro parámetros que permiten evaluar la capacidad de esta red: tasa de fallos, falsos positivos por imagen (FPPI), precisión de los *bounding boxes* y la exhaustividad. Al correr un detector de objetos existen múltiples parámetros a definir por el usuario que marcan el comportamiento de la red. Se ha evaluado la red con todas las posibles combinaciones de parámetros lógicas y se ha determinado el mejor rango de funcionamiento de la red. Estos parámetros son el nivel de superposición de las *bounding boxes* y el límite de incertidumbre aceptable para determinar que lo detectado es un objeto. Al analizar todos los casos lógicos se han obtenido los resultados mostrados en la Figura 5.8. En el eje x se puede ver los casos que se han analizado. El umbral de confianza se ha variado desde 0.5 hasta 1 en intervalos de 0.05 y para cada caso se ha evaluado modificando el nivel de superposición entre *bounding boxes*. El nivel de superposición se ha variado entre 0 y 0.5 con intervalos de 0.05.

Analizando estos resultados, se ha determinado que un buen punto de compromiso entre precisión, tasa de fallos y falsos positivos se obtiene con un límite de incertidumbre de 0.95 y un

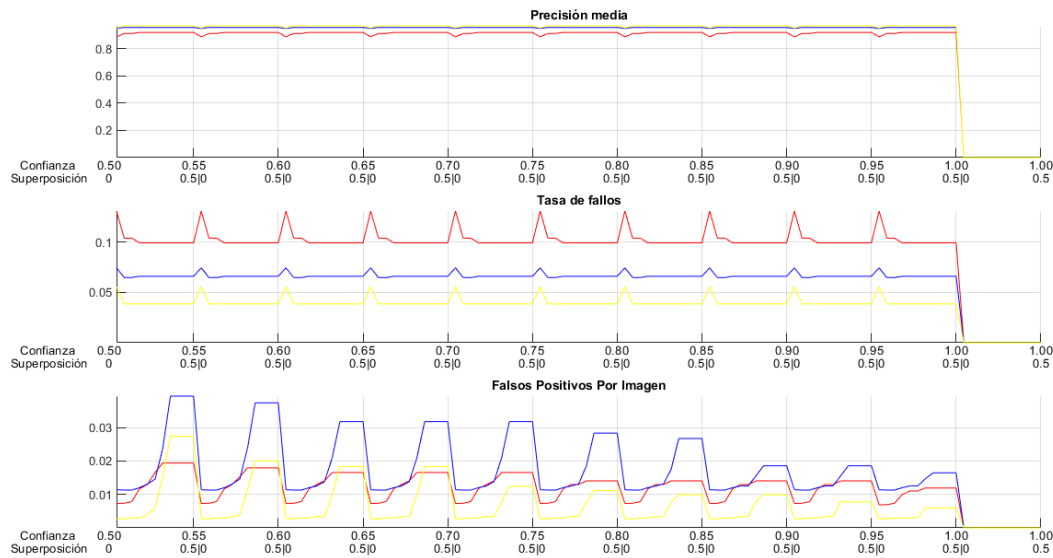


Figura 5.8. Evaluación de R-CNN basado en LEGONet en función de diversos parámetros

factor de superposición de 0.25. En los resultados mostrados a continuación se ha trabajado en este punto de operación. Tras analizar observamos que todas las gráficas siguen las distribuciones esperadas, al aumentar los falsos positivos hay más posibilidades de detectar la pieza y por ello aumenta la precisión. Al reducir los falsos positivos pasa el efecto contrario. Al reducir los falsos positivos hay mayor probabilidad de no detectar la pieza y al aumentarlos, pasa el caso contrario. Analizando cada caso por separado, se ha llegado a las siguientes conclusiones:

- **Amarillo:** El error cometido al detectar las piezas tanto en precisión como en fallos y falsos positivos es bastante pequeño. Y si se observan las gráficas se puede deducir que no han sido numerosos los casos en los que no se han detectado correctamente las piezas y por ello tienen una forma tan rectilínea.
- **Rojo:** La tasa de fallos es bastante superior respecto al amarillo y la precisión también baja. El número de falsos positivos por imagen también crece bastante. Analizando las imágenes de forma individual se ha observado que esto se debe a que en varias situaciones el sistema detectaba varias veces la misma pieza roja.
- **Azul:** El azul es una situación intermedia entre el rojo y el amarillo. Tiene una precisión bastante elevada y una tasa de fallos baja, pero presenta una mayor número de falsos positivos que el amarillo.

Al comparar estos resultados con los previamente obtenidos en segmentación con máscaras de color (Capítulo 3) observamos la verdadera capacidad de las redes neuronales y por qué representan el futuro. R-CNN destaca por ser mucho más robusto y constante y con un menor número de falsos positivos. Se puede observar una mejor comparación entre todos los métodos en el Capítulo 7.

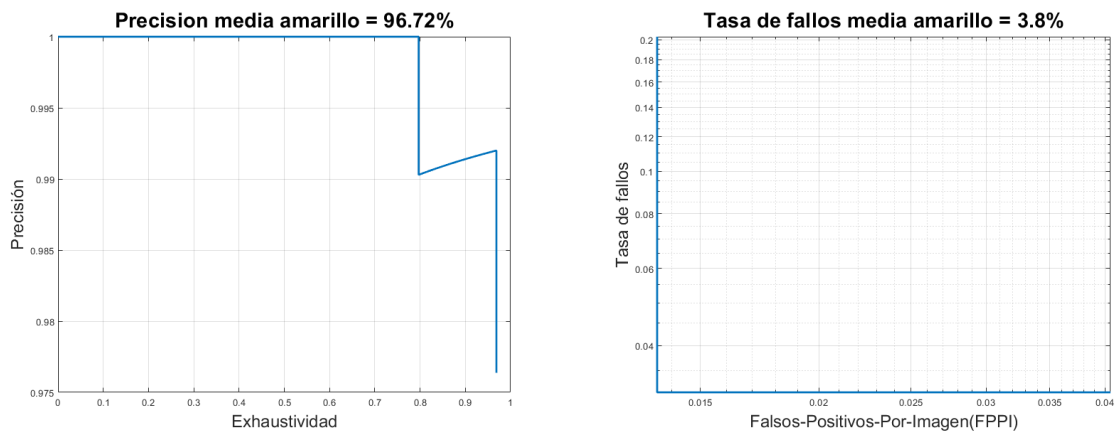


Figura 5.9. Estudio de la segmentación por R-CNN al detectar piezas amarillas

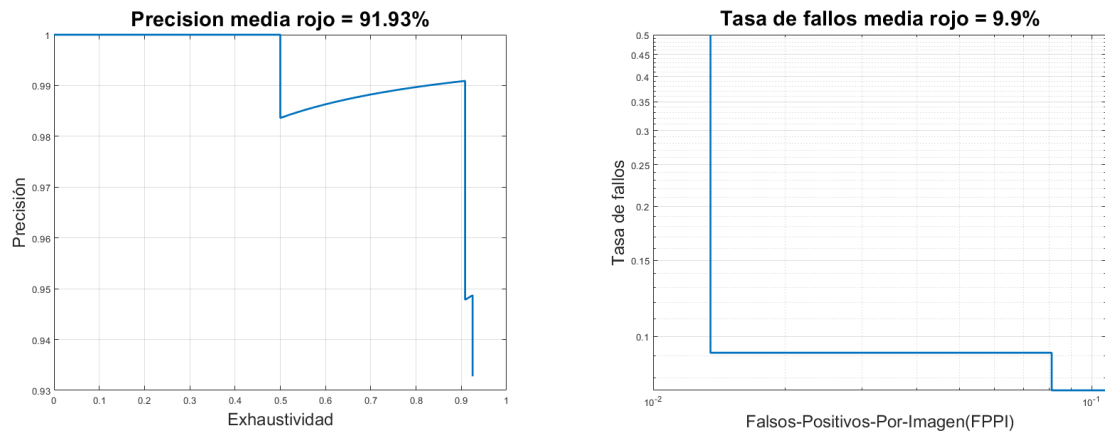


Figura 5.10. Estudio de la segmentación por R-CNN al detectar piezas rojas

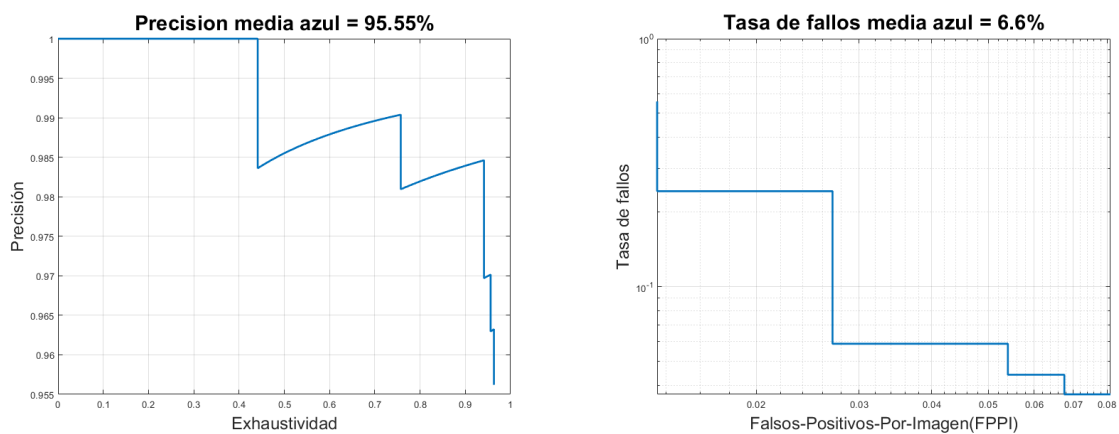


Figura 5.11. Estudio de la segmentación por R-CNN al detectar piezas azules

5.2.2. R-CNN basado en LEGO16

Partiendo del clasificador LEGO16 y con la ayuda de MATLAB y la base de datos de imágenes de LEGOS de la Sección 5.1.1, se ha creado y entrenado un detector de objetos tipo R-CNN.

Estructura

LEGO16 está formada por un total de 16 capas con pesos y un total de 41 capas, 138 millones de parámetros y 13 millones de neuronas. De las 16 capas, 13 son capas convoluciones, 3 son capas completamente conectadas y entre medias se encuentran capas del tipo *ReLU*, *Max pooling* y *Dropouts*. Para poder ser reutilizado y empleado como detector de objetos, es necesario modificar la estructura y el funcionamiento de la red neuronal.

Como se puede ver en la Figura 5.6, R-CNN es un proceso definido por dos etapas, la primera etapa es la propuesta de regiones y la segunda es el análisis de dichas regiones con el clasificador. Esto implica que a LEGONet se le va a tener que incluir el proceso de propuesta de regiones y de nuevo, se van a tener que modificar las tres últimas capas para reentrenar al clasificador para distinguir las piezas de LEGO.

- Capa 39: Completamente conectada $(1 \times 1 \times 7) \rightarrow (1 \times 1 \times 4)$
- Capa 40: *Softmax* $(1 \times 1 \times 1000) \rightarrow (1 \times 1 \times 4)$
- Capa 41: Salida del clasificador

La dimensión de estas nuevas capas es cuatro ya que R-CNN necesita también incluir la clase fondo para el caso en que no detecte nada en alguna de las regiones propuestas. Con estos cambios, la nueva estructura se puede ver en la Figura 5.12

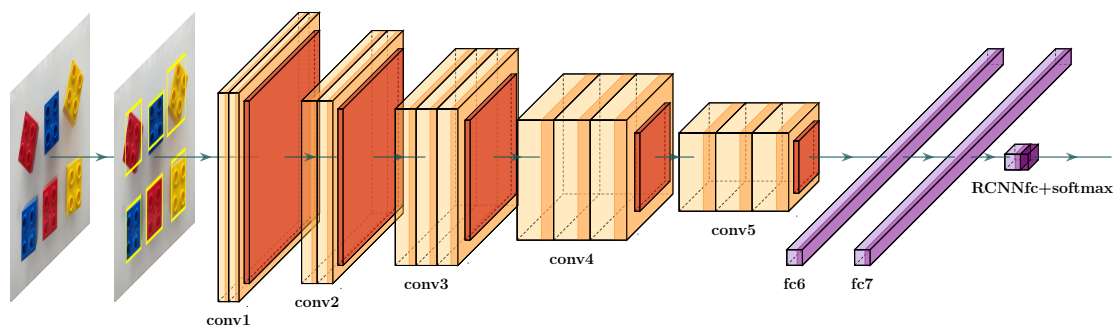


Figura 5.12. Estructura de R-CNN basado en LEGO16

Entrenamiento y resultados

En este proyecto se ha desarrollado la estructura y se han hecho todos los preparativos para el entrenamiento de esta red neuronal, desgraciadamente, debido a limitaciones por *hardware* no se ha podido entrenar y por lo tanto no hay resultados que mostrar. Por suerte, estas limitaciones solo han sucedido en el desarrollo de R-CNN basado en LEGO16 y esto es debido a falta de memoria VRAM.

5.3. Faster R-CNN

Como ya se ha visto en la sección anterior, en 2013 R-CNN debutó y triunfó frente a la competencia gracias a que suponía una mejora de rendimiento frente a otros métodos como

ventana flotante ya que en lugar de analizar todas las secciones posibles solo se tenían que analizar aproximadamente 2000. Aunque este representaba un avance y una mejora, seguía siendo un proceso lento y lejos de poder ser empleado para el tratamiento de imágenes en tiempo real. El cuello de botella de este método era el cálculo de las regiones de interés y el proceso de tener que analizar 2000 regiones de forma independiente. Este proceso era lento y frenaba notablemente el potencial de R-CNN. Por ello un par de años después, el mismo autor de R-CNN, desarrollo una variante de R-CNN que bautizó como Fast R-CNN [Gir15]. Este nuevo sistema se caracteriza porque cada imagen solo tiene que ser pasada por la red neuronal una vez. La red neuronal crea un mapa convolucional con las propiedades de la imagen y a continuación analiza y clasifica cada una de las regiones propuestas.

Con este nuevo método se consiguió reducir significativamente el tiempo de entrenamiento y el tiempo de ejecución para analizar una imagen. Desgraciadamente el sistema sigue presentando un cuello de botella, la obtención de regiones de interés. Es por ello, por lo que un par de meses después un equipo de investigadores entre los que se incluye Ross Girshick, el autor y creador de R-CNN y Fast R-CNN, desarrollaron un nuevo sistema capaz de obtener las regiones de interés con la ayuda de redes convolucionales. Este método es conocido como Faster R-CNN [Ren+15] y en la actualidad sigue siendo uno de los métodos más empleados, sobre todo cuando lo que se desea es precisión al detectar los objetos.

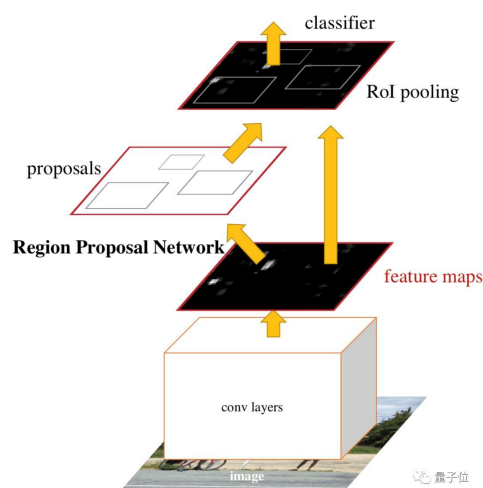


Figura 5.13. Esquema del funcionamiento de Faster R-CNN (Fuente: <https://medium.com/egen/region-proposal-network-rpn-backbone-of-faster-r-cnn-4a744a38d7f9>)

Siguiendo con el objetivo de poder comparar redes neuronales y su evolución con el paso de los años, en esta sección se van a desarrollar dos redes tipo Faster R-CNN basadas en LEGONet y LEGO16.

5.3.1. Faster R-CNN basado en LEGONet

LEGONet cuenta con un total de 25 capas, 5 son capas de convolución y 2 son completamente conectadas. En total cuenta con 60 millones de parámetros y 650.000 neuronas. Frente a los estándares actuales, es una red simple y fácil de entrenar pero antes es necesario modificar la para transformarla en una red tipo Faster R-CNN. En comparación con R-CNN, la transformación a Faster R-CNN es más laboriosa y requiere de más pasos.

Estructura

El primer paso, es idéntico a R-CNN. Como se desea reentrenar LEGONet para detectar nuevos objetos, es necesario eliminar las tres últimas capas y sustituirlas por similares, pero con la dimensión correcta, que es el número total de clases que se desean detectar más un clase para el fondo.

- Capa 23: Completamente conectada $(1 \times 1 \times 7) \rightarrow (1 \times 1 \times 4)$
- Capa 24: *Softmax* $(1 \times 1 \times 1000) \rightarrow (1 \times 1 \times 4)$
- Capa 23: Salida del clasificador

Una vez se ha terminado de modificar las capas centradas en clasificación, es el momento de añadir a la red neuronal el sistema encargado de determinar las posibles regiones de interés. Tal y como se ha explicado en la introducción a Faster R-CNN, dentro de la red neuronal, hay una sección dedicada al cálculo de posibles regiones de interés y ello a través de capas de convolución. El primer paso para la creación de esta sección es la determinación de la capa para la extracción de características. En este caso, como se trabaja con una variante de AlexNet, la capa escogida es Relu5. Se trata de una capa del tipo *Max pooling* que será sustituida por una del tipo *ROI Max pooling*. Las capas *Max pooling* se encargan de reducir las dimensiones de las capas internas de una red para evitar que estas crezcan demasiado, en comparación, *ROI Max pooling* cumple una función similar pero con la diferencia de ser capaz de ajustarse para trabajar con dimensiones no uniformes.

Tras estos cambios, la red ya esta preparada para introducir dentro de esta misma a una red encargada de proponer regiones de interés (Region Proposal Network - RPN). Esta nueva red RPN se caracteriza porque internamente esta constituida por capas de convolución y una capa de clasificación. Este clasificador determina la probabilidad de que las regiones de interés tengan un objeto. Toda esta información extraída se manda a la capa de *ROI Max pooling* para así poder analizar la imagen por completo y determinar la presencia de objetos en la imagen. Para que esto funcione correctamente y se pueda entrenar a la red, es necesario incluir dos capas más al final de la red para ayudar a determinar las regiones de interés. Estas últimas capas son una capa completamente conectada y una capa *Box regression*. Tras aplicar todos los cambios mencionados, la estructura final de la red se puede observar en la Figura 5.15.

Entrenamiento

Para el entrenamiento de esta red se ha usado la base de datos diseñada en la Sección 5.1. Esta base de datos consta de un total de 2.076 imágenes con un total de 5.886 piezas de LEGO. Se han realizado numerosas pruebas de entrenamiento y tras numerosas pruebas se ha descubierto que la red presentaba problemas para aprender debido a la red RPN, esta tardaba en aprender y por ello evitaba que el clasificador también aprendiese. Por ello, el aprendizaje se ha realizado en cuatro etapas muy parecidas pero con el objetivo de ir enseñando poco a poco

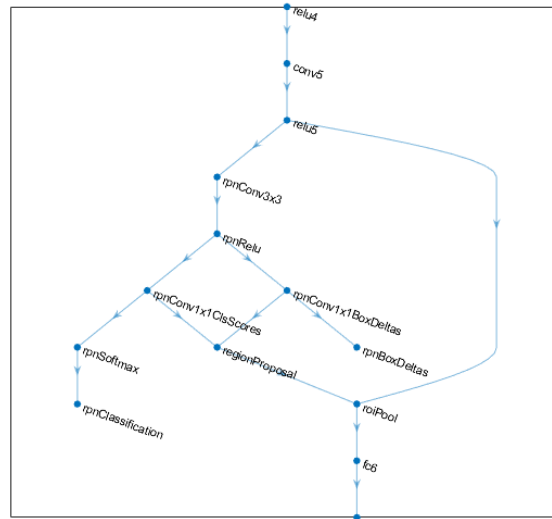


Figura 5.14. Esquema de la red RPN empleada en Faster R-CNN basado en LEGONet

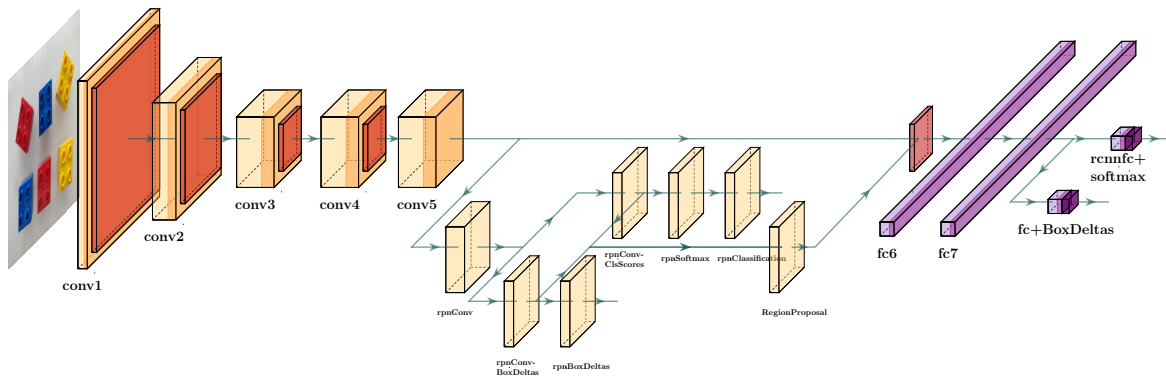


Figura 5.15. Estructura de Faster R-CNN basado en LEGONet

al clasificador y a la red RPN. En el caso de Faster R-CNN el sistema depende de la red RPN para poder entrenar y esta a su vez depende del clasificador ya que extrae información con sus capas de convolución. Es por ello por lo que el proceso de aprendizaje es del tipo iterativo. La red se ha entrenado cuatro veces de forma que en cada etapa la red RPN pueda ser reentrenada con los nuevos pesos de las capas de convolución. Tras varias pruebas, se ha determinado que las mejores opciones para el entrenamiento son las mostradas en la Tabla 5.3.

Resultados

Con la ayuda del *script* definido en la Sección 5.1.2, se han podido obtener cuatro parámetros que permiten evaluar la capacidad de esta red: tasa de fallos, falsos positivos por imagen (FPPI), precisión de los *bounding boxes* y la exhaustividad. Al correr un detector de objetos existen múltiples parámetros a definir por el usuario que marcan el comportamiento de la red. Se ha evaluado la red con todas las posibles combinaciones de parámetros lógicos y se ha determinado el mejor rango de funcionamiento de la red. Estos parámetros son el nivel de superposición de las *bounding boxes* y el límite de incertidumbre aceptable para determinar que lo detectado es un objeto. Al analizar todos los casos lógicos se han obtenido los resultados mostrados en la Figura 5.16. En el eje x se puede ver los casos que se han analizado. El umbral de confianza se

Opciones de entrenamieto	
Solver	Stochastic Gradient Descent with Momentum (SGDM)
Momentum	0.9
Initial Learn Rate	1.00E-04
Learn Rate Schedule	piecewise
Learn Rate Drop Factor	0.1
Learn Rate Drop Period	70
L2Regularization	0.004
Max Epochs	100-100-100-100
Mini Batch Size	200
Shuffle data	every epoch

Tabla 5.3. Opciones de entrenamiento de Faster R-CNN basado en LEGONet

ha variado desde 0.5 hasta 1 en intervalos de 0.05 y para cada caso se ha evaluado modificando el nivel de superposición entre *bounding boxes*. El nivel de superposición se ha variado entre 0 y 0.5 con intervalos de 0.05.

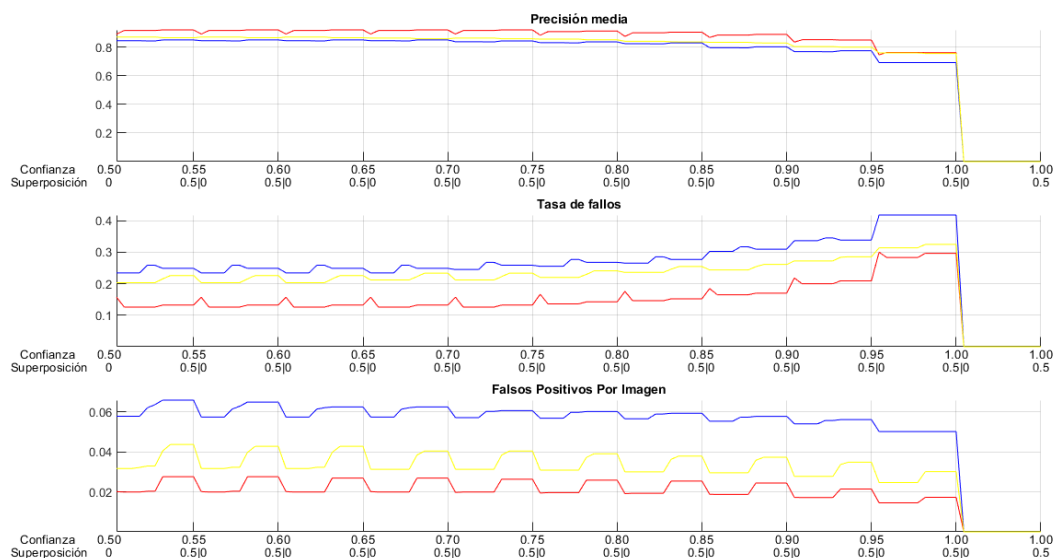


Figura 5.16. Evaluación de Faster R-CNN basado en LEGONet en función de diversos parámetros

Analizando estos resultados, se ha determinado que un buen punto de compromiso entre precisión, tasa de fallos y falsos positivos se obtiene con un límite de incertidumbre de 0.65 y un factor de superposición de 0.05. En los resultados mostrados a continuación se ha trabajado en este punto de operación. Tras analizar observamos que todas las gráficas presentan peores resultados frente a R-CNN. Esto es algo de esperar ya que Faster R-CNN reduce precisión a cambio de rapidez. Aun así, los resultados son positivos y demuestra que la red ha aprendido.

- **Amarillo:** El error cometido al detectar las piezas tanto en precisión como en fallos y falsos positivos es demasiado elevado como para poder ser implantado. Además se observa que la exhaustividad nunca llega a uno lo cual es debido a la presencia de falsos positivos. También se observa como la precisión es demasiado baja como para ser implantado.

- Rojo: El rojo presenta una situación similar al amarillo, aunque con un menor número de falsos positivos. Presenta una tasa de fallos y una precisión un poco baja como para poder ser implantada.
- Azul: El azul se asemeja bastante al amarillo en la distribución de sus gráficas, aunque en general ha obtenido mejores resultados.

Analizando los resultados en conjunto se puede ver que la red ha tenido problemas para distinguir las piezas. Aunque se esperaba unos resultados inferiores a la red R-CNN basado en LEGONet estos son más bajos a lo esperado. Sería recomendable dedicar más tiempo al entrenamiento de esta red y con mayor potencia para así poder comprobar la verdadera capacidad de esta red. Se puede observar una mejor comparación entre todos los métodos en el Capítulo 7.

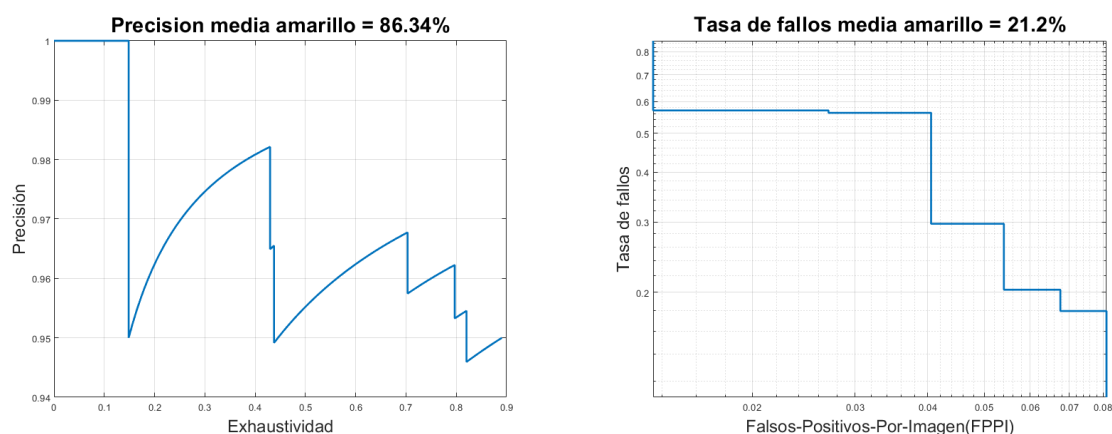


Figura 5.17. Estudio de la segmentación por Faster R-CNN basado en LEGONet al detectar piezas amarillas

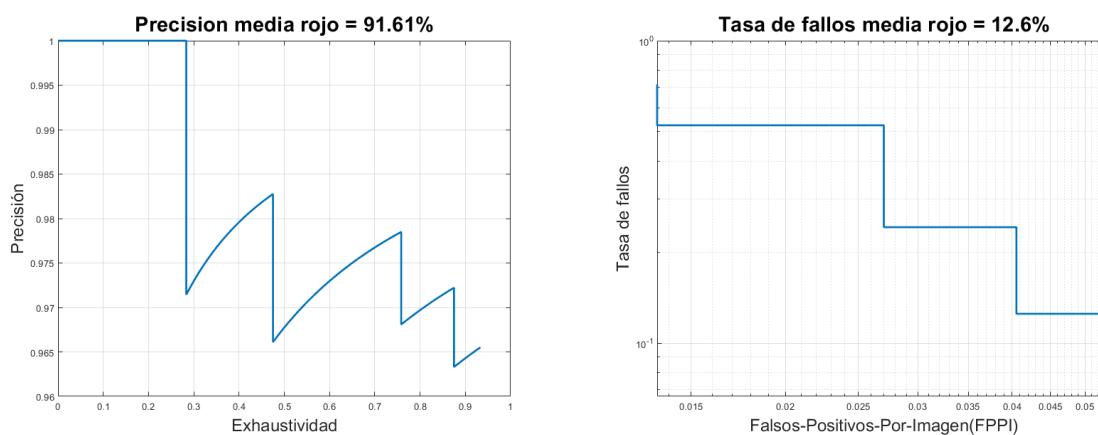


Figura 5.18. Estudio de la segmentación por Faster R-CNN basado en LEGONet al detectar piezas rojas

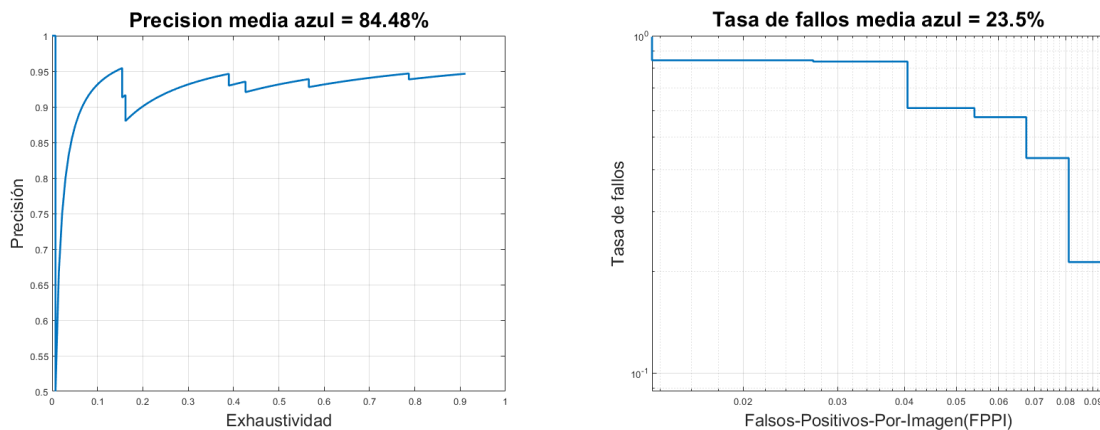


Figura 5.19. Estudio de la segmentación por Faster R-CNN basado en LEGONet al detectar piezas azules

5.3.2. Faster R-CNN basado en LEGO16

LEGO16 cuenta con un total de 41 capas, 13 son capas de convolución y 3 son completamente conectadas. En total cuenta con 138 millones de parámetros y 13 millones de neuronas. Frente a LEGONet, se trata de una red grande y pesada que requiere de más tiempo y potencia para ser entrenada, pero antes es necesario modificar la para transformarla en una red tipo Faster R-CNN. En comparación con R-CNN, la transformación a Faster R-CNN es más laboriosa y requiere de más pasos.

Estructura

El proceso para la obtención de una red tipo Faster R-CNN basado en LEGO16 es muy similar al proceso basado en LEGONet. En ambos casos se comienza por la variación del clasificador para la detección de las nuevas clases y a continuación se añade la red RPN para la detección de regiones de interés. La principal diferencia reside en como se implanta la red RPN dentro del clasificador, como ya se ha visto en la Sección 5.3.1, en LEGONet se introdujo en la capa ReLu5. En el caso de LEGO16, como se desea que las capas de convolución extraigan toda la información posible, la red RPN se implanta en la capa ReLu5_3 tras las últimas capas de convolución. Tras aplicar todos los cambios se obtiene la estructura final de la red, esta se muestra en la Figura 5.21.

Entrenamiento

Para el entrenamiento de esta red se ha usado la base de datos diseñada en la Sección 5.1. Esta base de datos consta de un total de 2.076 imágenes con un total de 5.886 piezas de LEGO. Se han realizado numerosas pruebas de entrenamiento y tras numerosas pruebas se ha descubierto que la red presentaba problemas para aprender debido a la red RPN, esta tardaba en aprender y por ello evitaba que el clasificador también aprendiese. Por ello, el aprendizaje se ha realizado en cuatro etapas muy parecidas pero con el objetivo de ir enseñando poco a poco al clasificador y a la red RPN. En el caso de Faster R-CNN el sistema depende de la red RPN para poder entrenar y esta a su vez depende del clasificador ya que extrae información con sus capas de convolución. Es por ello por lo que el proceso de aprendizaje es del tipo iterativo. La

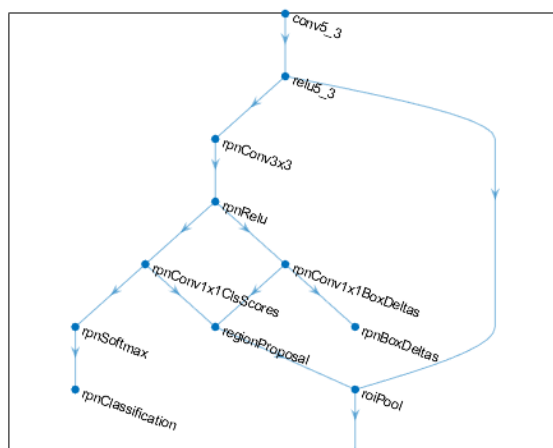


Figura 5.20. Esquema de la red RPN empleada en Faster R-CNN basado en LEGO16

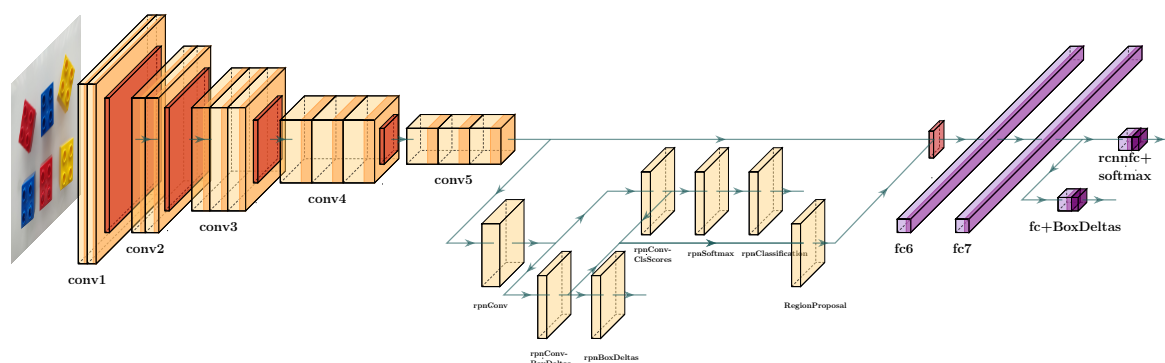


Figura 5.21. Estructura de Faster R-CNN basado en LEGO16

red se ha entrenado cuatro veces de forma que en cada etapa la red RPN pueda ser reentrenada con los nuevos pesos de las capas de convolución. Tras varias pruebas, se ha determinado que las mejores opciones para el entrenamiento son las mostradas en la Tabla 5.4.

Opciones de entrenamineto	
Optimizador	Stochastic Gradient Descent with Momentum (SGDM)
Momentum	0.9
Initial Learn Rate	1.00E-03
Learn Rate Schedule	piecewise
Learn Rate Drop Factor	0.1
Learn Rate Drop Period	70
L2Regularization	0.004
Max Epochs	30-30-30-30
Mini Batch Size	3
Shuffle data	every epoch

Tabla 5.4. Opciones de entrenamiento de Faster R-CNN basado en LEGO16

Resultados

Con la ayuda del *script* definido en la Sección 5.1.2, se han podido obtener cuatro parámetros que permiten evaluar la capacidad de esta red: tasa de fallos, falsos positivos por imagen (FPPI),

precisión de los *bounding boxes* y la exhaustividad. Al correr un detector de objetos existen múltiples parámetros a definir por el usuario que marcan el comportamiento de la red. Se ha evaluado la red con todas las posibles combinaciones de parámetros lógicos y se ha determinado el mejor rango de funcionamiento de la red. Estos parámetros son el nivel de superposición de las *bounding boxes* y el límite de incertidumbre aceptable para determinar que lo detectado es un objeto. Al analizar todos los casos lógicos se han obtenido los resultados mostrados en la Figura 5.22. En el eje x se puede ver los casos que se han analizado. El umbral de confianza se ha variado desde 0.5 hasta 1 en intervalos de 0.05 y para cada caso se ha evaluado modificando el nivel de superposición entre *bounding boxes*. El nivel de superposición se ha variado entre 0 y 0.5 con intervalos de 0.05.

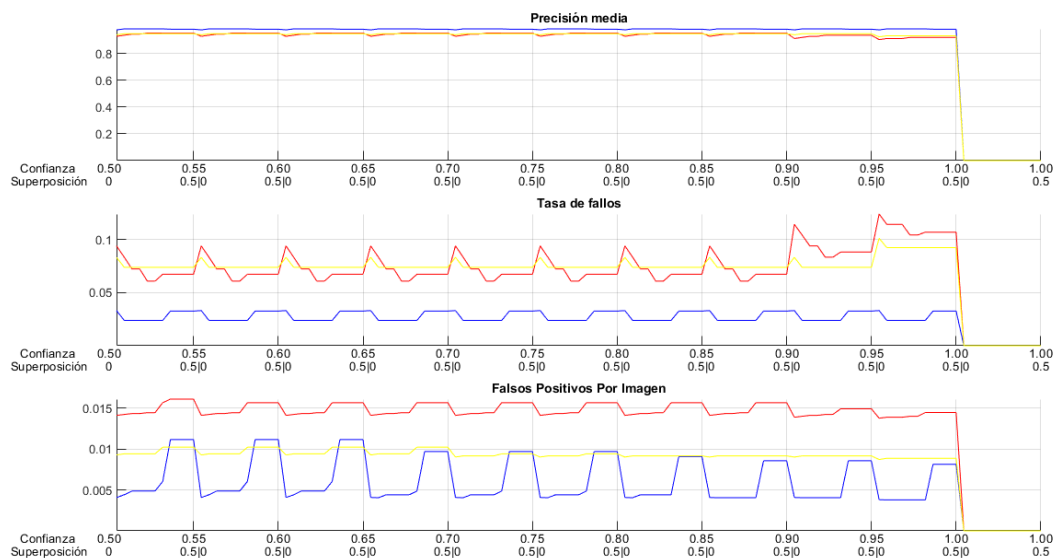


Figura 5.22. Evaluación de Faster R-CNN basado en LEGO16 en función de diversos parámetros

Analizando estos resultados, se ha determinado que un buen punto de compromiso entre precisión, tasa de fallos y falsos positivos se obtiene con un límite de incertidumbre de 0.85 y un factor de superposición de 0.20. En los resultados mostrados a continuación se ha trabajado en este punto de operación. Tras analizar observamos que todas las gráficas presentan mejores resultados frente a Faster R-CNN basado en LEGONet. Esto es algo de esperar ya que LEGO16 es una red mayor, más compleja y con más capacidad para extraer características.

- **Amarillo:** La tasa de fallos es demasiado elevado en comparación al resto de resultados y similar al obtenido con la segmentación por color en el Capítulo 3. Por el contrario, los falsos positivos por imagen son bastante reducidos en general y la precisión es bastante elevada.
- **Rojo:** El rojo se asemeja bastante al amarillo tanto en precisión como tasa de fallos y falsos positivos.
- **Azul:** El azul presenta los mejores resultados de los tres. La precisión es muy elevada y la tasa de fallos son muy bajos, aunque los falsos positivos son superiores al rojo y amarillo.

Analizando los resultados en conjunto se puede ver que la red no ha tenido problemas para distinguir las piezas. Se esperaba unos resultados superiores a los de Faster R-CNN basado en LEGONet y ha cumplido con las expectativas aunque sería recomendable dedicar más tiempo

al entrenamiento de esta red y con mayor potencia para así poder comprobar la verdadera capacidad de esta red. Se puede observar una mejor comparación entre todos los métodos en el Capítulo 7.

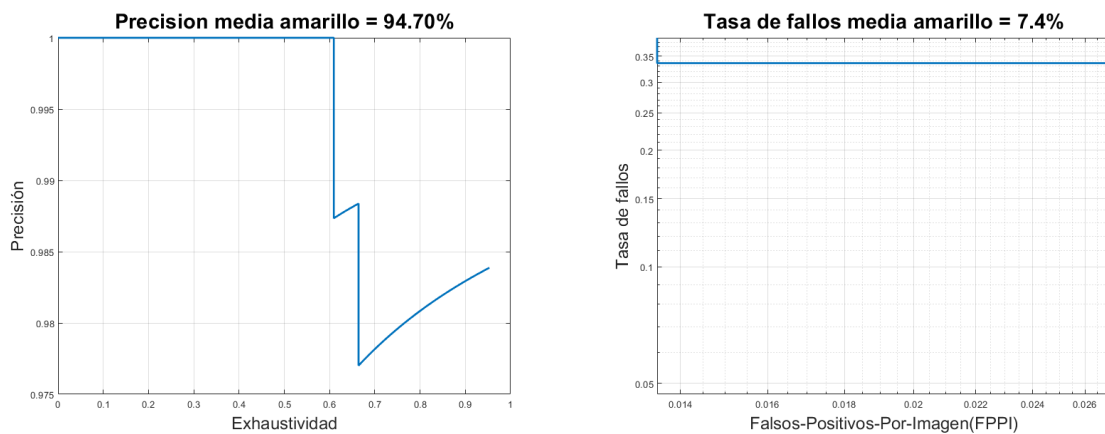


Figura 5.23. Estudio de la segmentación por Faster R-CNN basado en LEGO16 al detectar piezas amarillas

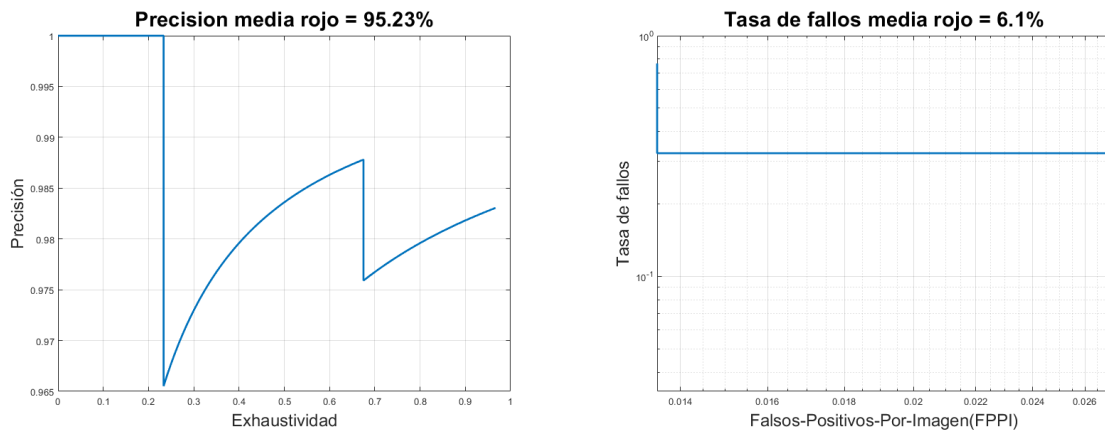


Figura 5.24. Estudio de la segmentación por Faster R-CNN basado en LEGO16 al detectar piezas rojas

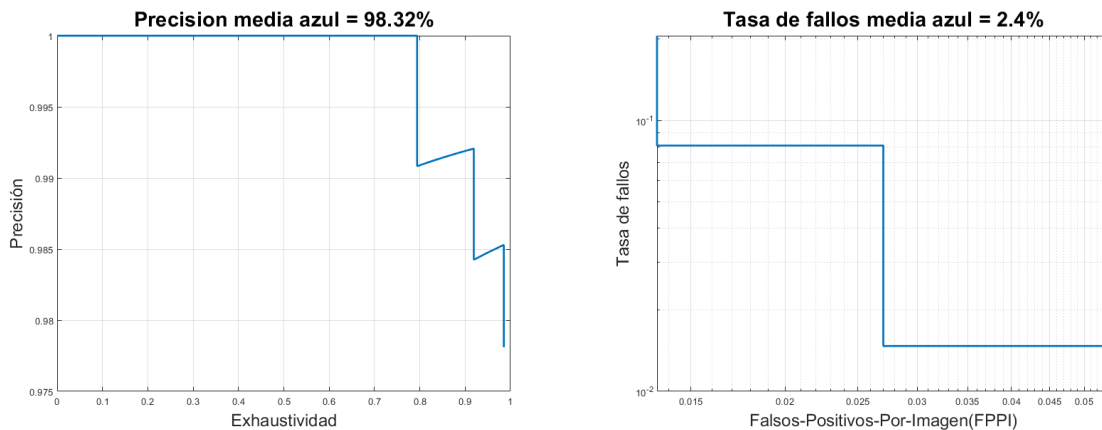


Figura 5.25. Estudio de la segmentación por Faster R-CNN basado en LEGO16 al detectar piezas azules

5.4. 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 ve una vez. Esto puede parecer similar a Faster R-CNN pero difiere bastante. En Faster R-CNN ya la imagen es primero pasada por las capas de convolución para a continuación determinar regiones de interés con la ayuda de una red tipo RPN y a continuación se analiza cada una de las posibles regiones de interés. Esto implica qué 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.

A diferencia de las redes ya vistas en este proyecto, YOLO no fue diseñada con la idea de reaprovechar 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 7x7 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. Este método presenta numerosas ventajas frente a R-CNN y Faster R-CNN, en primer lugar, destaca por su rapidez. La red diseñada por los desarrolladores de YOLO cuenta con 24 capas de convolución y es capaz de correr a 45 imágenes por segundo. Esta red también se caracteriza por tener una estructura más propia de las Fully Convolutional Neural 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 la 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 5.26

Siguiendo con el objetivo de poder comparar redes neuronales y su evolución con el paso de los años, en esta sección se van a desarrollar dos redes tipo YOLO basadas en LEGONet y LEGO16. Debido al gran interés y las capacidades de YOLO, este ha sido desarrollado bastante en los últimos dos años. En la actualidad se han publicado cinco versiones de YOLO que mejoran su rendimiento y le dotan de más capacidades. Desgraciadamente, debido a la rapidez con la que se ha actualizado y desarrollado YOLO, en el momento de desarrollo de este proyecto MATLAB solo soporta hasta la segunda versión de YOLO. Es por ello que todas las redes del tipo YOLO se han basado en YOLO v2.

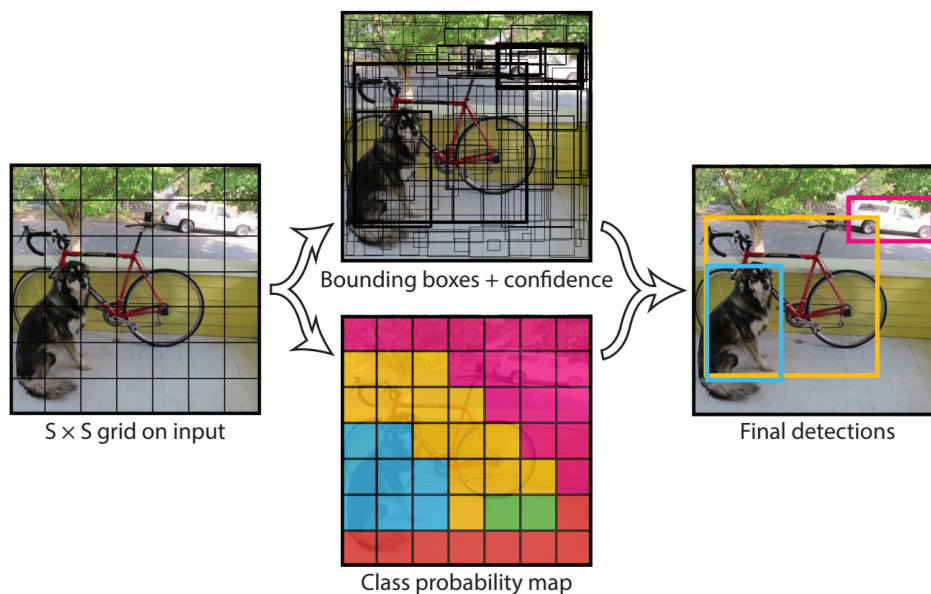


Figura 5.26. Esquema del funcionamiento de YOLO (Fuente: publicación original de YOLO [Red+15])

5.4.1. YOLO basado en LEGONet

LEGONet cuenta con un total de 25 capas, 5 son capas de convolución y 2 son completamente conectadas. En total cuenta con 60 millones de parámetros y 650.000 neuronas. Frente a los estándares actuales, es una red simple y fácil de entrenar, pero antes es necesario modificar la para transformarla en una red tipo YOLO.

Estructura

YOLO se caracteriza por trabajar de forma diferente al resto de detectores de objetos ya que no se basa en un clasificador. Es por ello que aunque se vaya a partir de un clasificador, es necesario eliminar las últimas capas encargadas de la clasificación. El motivo por el que se parte de un clasificador es porque las primeras capas de convolución ya han sido entrenadas y se puede reaprovechar ese entrenamiento.

Con la ayuda de MATLAB, se han eliminado todas las capas de LEGONet pasado la capa Relu5 y se han sustituido por el sistema de YOLO. Esto implica la inclusión de dos capas de convolución acompañadas por capas *Batch* y *ReLu*. Y por último, se incluye una capa de transformación y la capa de salida. Estas dos últimas capas se encargan de coger las activaciones de las capas de convolución y redefinir los *bounding boxes* para que se asemejen a los datos con los que han sido entrenados. Para que la red puede redefinir los *bounding boxes*, necesita de una referencia. Al crear la red es necesario estipular el número de *bounding boxes* y las dimensiones de los mismos que la red se puede encontrar y que empleará como referencia. En nuestro caso y con un par de pruebas, se ha encontrado que los mejores resultados se obtienen al definir un total de 5 posibles *bounding boxes*.

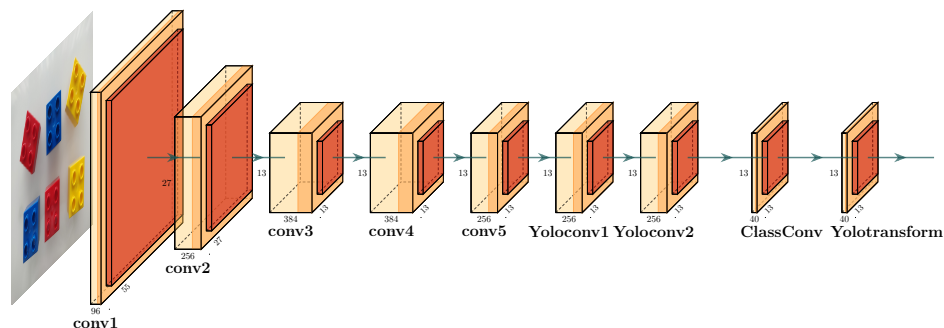


Figura 5.27. Estructura de YOLO basado en LEGONet

Entrenamiento

Para el entrenamiento de esta red se ha usado la base de datos diseñada en la Sección 5.1. Esta base de datos consta de un total de 2.076 imágenes con un total de 5.886 piezas de LEGO. Se han realizado numerosas pruebas de entrenamiento y se ha descubierto que las mejores opciones de entrenamiento son las mostradas en la Tabla 5.5.

Resultados

Con la ayuda del *script* definido en la Sección 5.1.2, se han podido obtener cuatro parámetros que permiten evaluar la capacidad de esta red: tasa de fallos, falsos positivos por imagen (FPPI), precisión de los *bounding boxes* y la exhaustividad. Al correr un detector de objetos existen múltiples parámetros a definir por el usuario que marcan el comportamiento de la red. Se ha

Opciones de entrenamineto	
Optimizador	Stochastic Gradient Descent with Momentum (SGDM)
Momentum	0.9
Initial Learn Rate	1.00E-03
Learn Rate Schedule	piecewise
Learn Rate Drop Factor	0.5
Learn Rate Drop Period	50
L2Regularization	0.004
Max Epochs	400
Mini Batch Size	128
Shuffle data	every epoch

Tabla 5.5. Opciones de entrenamiento de YOLO basado en LEGONet

evaluado la red con todas las posibles combinaciones de parámetros lógicas y se ha determinado el mejor rango de funcionamiento de la red. Estos parámetros son el nivel de superposición de las *bounding boxes* y el límite de incertidumbre aceptable para determinar que lo detectado es un objeto. Al analizar todos los casos lógicos se han obtenido los resultados mostrados en la Figura 5.28. En el eje x se puede ver los casos que se han analizado. El umbral de confianza se ha variado desde 0.5 hasta 1 en intervalos de 0.05 y para cada caso se ha evaluado modificando el nivel de superposición entre *bounding boxes*. El nivel de superposición se ha variado entre 0 y 0.5 con intervalos de 0.05.

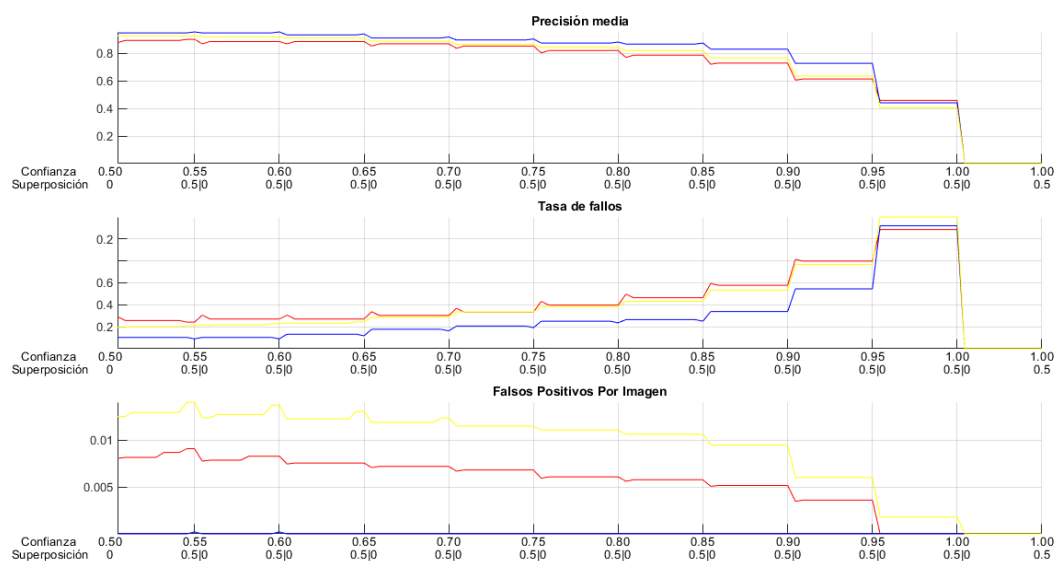


Figura 5.28. Evaluación de YOLO basado en LEGONet en función de diversos parámetros

Analizando estos resultados, se ha determinado que un buen punto de compromiso entre precisión, tasa de fallos y falsos positivos se obtiene con un límite de incertidumbre de 0.55 y un factor de superposición de 0.15. En los resultados mostrados a continuación se ha trabajado en este punto de operación. Tras analizar observamos que todas las gráficas presentan mejores resultados frente a Faster R-CNN basado en LEGONet y que todas las gráficas presenta distribuciones usuales. Esto es más notable en el caso de las piezas amarillas y rojas.

- **Amarillo:** El error cometido al detectar las piezas tanto en precisión como en fallos y falsos positivos es moderadamente elevado. Analizando las gráficas vemos que el número

de falsos positivos no es elevado y sin embargo la tasa de fallos es elevada. Además, se observa en las gráficas que los cambios son abruptos y escasos, esto se debe a que el sistema apenas ha tenido falsos positivos lo cual también implica que la exhaustividad apenas varía.

- Rojo: El rojo presenta una situación similar al amarillo. No presenta una tasa de falsos positivos elevada pero la tasa de fallos es más elevada. Como consecuencia, la precisión es más baja.
- Azul: El azul presenta los mejores resultados de los tres, pero en las gráficas no se refleja esta situación. Esto se debe a que al analizar las imágenes no ha habido ningún falso positivo y por ello la precisión no varía con la exhaustividad y tampoco la tasa de fallos con los falsos positivos.

Analizando los resultados en conjunto se puede ver que la red ha tenido problemas para distinguir las piezas amarillas y rojas ya que en determinadas situaciones estas no han sido detectadas debido a cambios de iluminación. Aun así supone un gran avance frente a Faster R-CNN basado en LEGONet. Se puede observar una mejor comparación entre todos los métodos en el Capítulo 7.

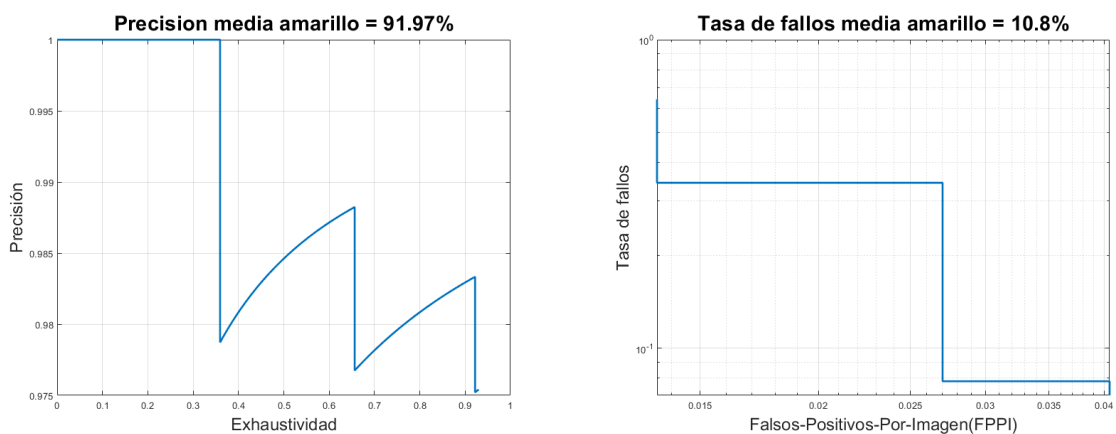


Figura 5.29. Estudio de la segmentación por YOLO basado en LEGONet al detectar piezas amarillas

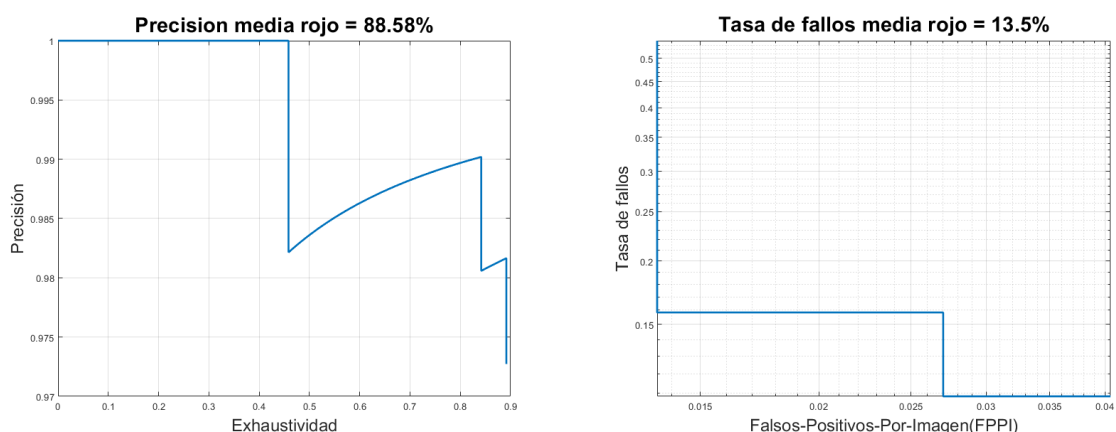


Figura 5.30. Estudio de la segmentación por YOLO basado en LEGONet al detectar piezas rojas

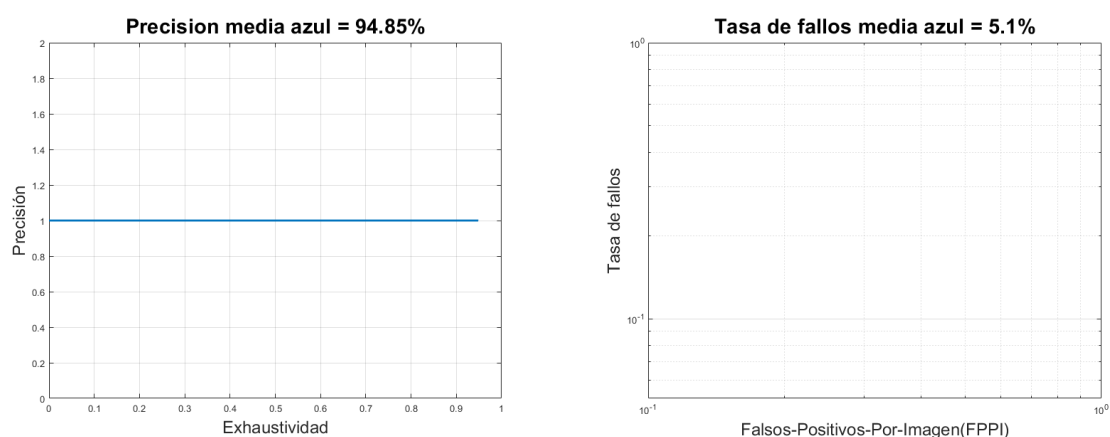


Figura 5.31. Estudio de la segmentación por YOLO basado en LEGONet al detectar piezas azules

5.4.2. YOLO basado en LEGO16

LEGO16 cuenta con un total de 41 capas, 13 son capas de convolución y 3 son completamente conectadas. En total cuenta con 138 millones de parámetros y 13 millones de neuronas. Frente a LEGONet, se trata de una red grande y pesada que requiere de más tiempo y potencia para ser entrenada, por suerte YOLO es bastante rápida para entrenar y esto no supone un gran problema. Pero antes es necesario modificar la para transformarla en una red tipo YOLO.

Estructura

Como ya se ha visto en la anterior sección, YOLO se caracteriza por no funcionar como un clasificador. Esto implica que es necesario cambiar el final del clasificador añadiendo más capas de convolución y una capa de transformación y otra de salida. Estas capas se van a añadir a

partir de la capa ReLu5_3 que es la capa *ReLU* de la última capa de convolución. Se puede observar la estructura final empleada para el entrenamiento en la Figura 5.32.

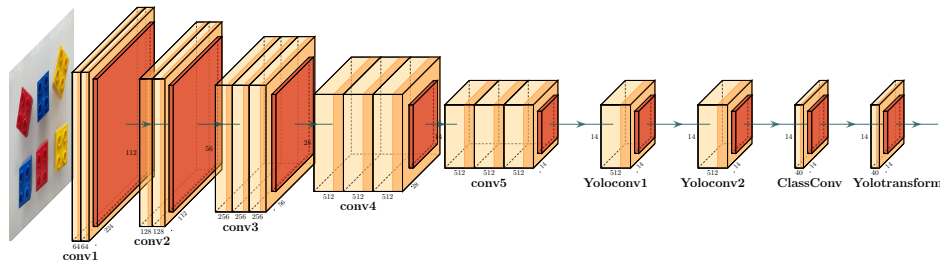


Figura 5.32. Estructura de YOLO basado en LEGO16

Entrenamiento

Para el entrenamiento de esta red se ha usado la base de datos diseñada en la Sección 5.1. Esta base de datos consta de un total de 2.076 imágenes con un total de 5.886 piezas de LEGO. Se han realizado numerosas pruebas de entrenamiento y se ha descubierto que las mejores opciones de entrenamiento son las mostradas en la Tabla 5.6.

Opciones de entrenamineto	
Optimizador	Stochastic Gradient Descent with Momentum (SGDM)
Momentum	0.9
Initial Learn Rate	1.00E-03
Learn Rate Schedule	piecewise
Learn Rate Drop Factor	0.1
Learn Rate Drop Period	50
L2Regularization	0.004
Max Epochs	100
Mini Batch Size	25
Shuffle data	every epoch

Tabla 5.6. Opciones de entrenamiento de YOLO basado en LEGO16

Resultados

Con la ayuda del *script* definido en la Sección 5.1.2, se han podido obtener cuatro parámetros que permiten evaluar la capacidad de esta red: tasa de fallos, falsos positivos por imagen (FPPI), precisión de los *bounding boxes* y la exhaustividad. Al correr un detector de objetos existen múltiples parámetros a definir por el usuario que marcan el comportamiento de la red. Se ha evaluado la red con todas las posibles combinaciones de parámetros lógicas y se ha determinado el mejor rango de funcionamiento de la red. Estos parámetros son el nivel de superposición de las *bounding boxes* y el límite de incertidumbre aceptable para determinar que lo detectado es un objeto. Al analizar todos los casos lógicos se han obtenido los resultados mostrados en la Figura 5.33. En el eje x se puede ver los casos que se han analizado. El umbral de confianza se ha variado desde 0.5 hasta 1 en intervalos de 0.05 y para cada caso se ha evaluado modificando el nivel de superposición entre *bounding boxes*. El nivel de superposición se ha variado entre 0 y 0.5 con intervalos de 0.05.

Analizando estos resultados, se ha determinado que un buen punto de compromiso entre precisión, tasa de fallos y falsos positivos se obtiene con un límite de incertidumbre de 0.75 y

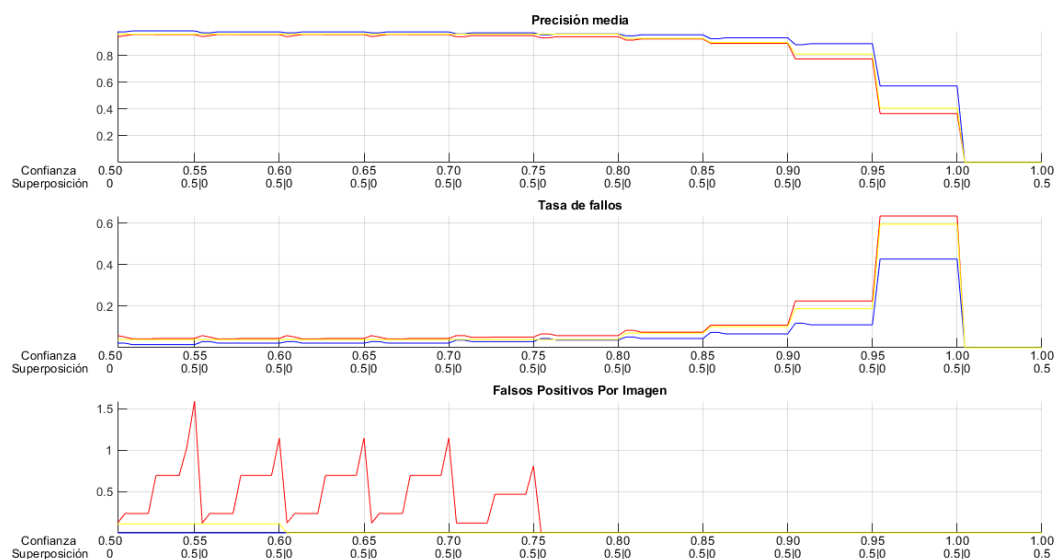


Figura 5.33. Evaluación de YOLO basado en LEGO16 en función de diversos parámetros

un factor de superposición de 0.2. Para los resultados mostrados a continuación se ha trabajado en este punto de operación.

Analizando estos resultados, se ha determinado que un buen punto de compromiso entre precisión, tasa de fallos y falsos positivos se obtiene con un límite de incertidumbre de 0.60 y un factor de superposición de 0.2. En los resultados mostrados a continuación se ha trabajado en este punto de operación. Tras analizar observamos que todas las gráficas presentan mejores resultados frente a YOLO basado en LEGONet. A pesar de que los resultados son muy positivos, las gráficas no lo reflejan correctamente debido a la falta de puntos para mostrar. Los falsos positivos han sido muy escasos y por ello hay tan pocos puntos y las gráficas son tan abruptas.

- **Amarillo:** El error cometido al detectar las piezas tanto en precisión como en fallos y falsos positivos es bastante bajo. Analizando las gráficas vemos que la tasa de fallos no es elevada y que no se ha dado ningún caso con falsos positivos. La precisión es muy elevada y aunque parece que se mantiene constante en 1, esto no es así ya que ha habido casos en los que no se han detectado las piezas rojas. Por ello la precisión media es inferior al 100 %.
- **Rojo:** El rojo presenta una situación similar al amarillo. Presenta una tasa de fallos baja y una precisión elevada y a diferencia del amarillo, sí que presenta falsos positivos, aunque muy bajo.
- **Azul:** El azul presenta los mejores resultados de los tres, pero en las gráficas no se refleja esta situación. Esto se debe a que al analizar las imágenes no ha habido ningún falso positivo y por ello la precisión no varía con la exhaustividad y tampoco la tasa de fallos.

Analizando los resultados en conjunto se puede ver la superioridad de VGG-16 frente a AlexNet y la capacidad de YOLO. Se puede observar una mejor comparación entre todos los métodos en el Capítulo 7.

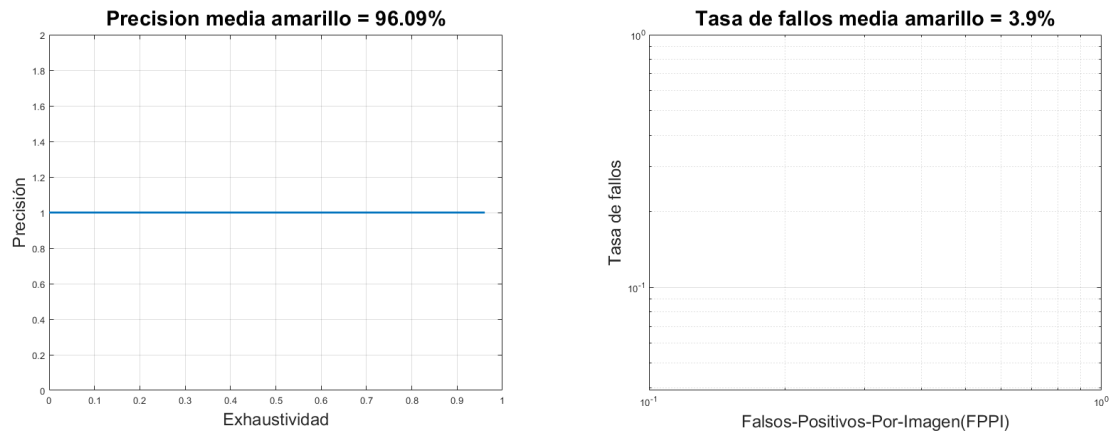


Figura 5.34. Estudio de la segmentación por YOLO basado en LEGO16 al detectar piezas amarillas

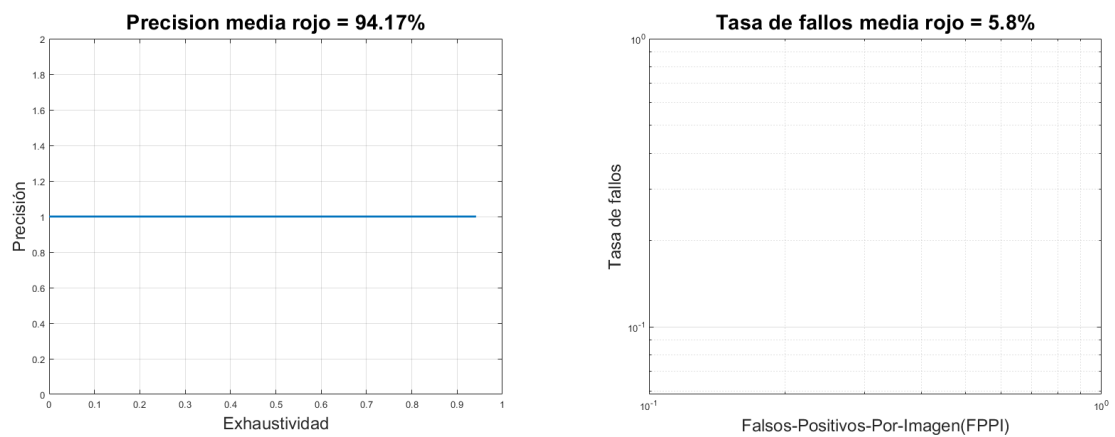


Figura 5.35. Estudio de la segmentación por YOLO basado en LEGO16 al detectar piezas rojas

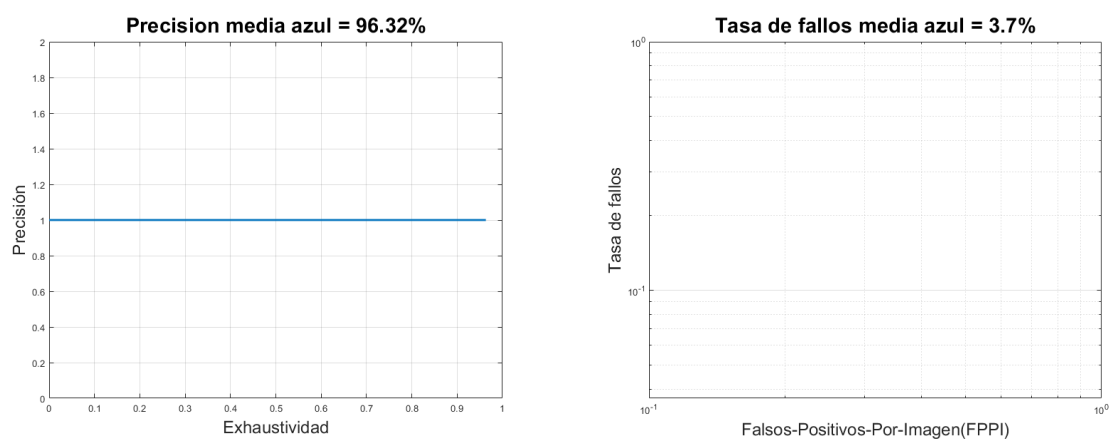


Figura 5.36. Estudio de la segmentación por YOLO basado en LEGO16 al detectar piezas azules

6

Extracción de atributos

En este capítulo se van a desarrollar las herramientas y tecnologías necesarias para la extracción de la altura y posición de cada pieza. De esta forma se dispondrá de toda la información necesaria para que el brazo robótico pueda interactuar con cada pieza.

El objetivo final de este proyecto es la implantación del sistema de visión artificial en un brazo robótico para que este pueda interactuar con las piezas detectadas. Pero para ello es necesario primero obtener las dimensiones de la pieza, la altura de la pieza, así como la orientación de la misma. En capítulos anteriores se ha visto como obtener la posición de la pieza y sus dimensiones con la ayuda de detectores de objetos. En este capítulo se van a desarrollar las herramientas y tecnologías necesarias para la extracción de la altura y posición de cada pieza. De esta forma se dispondrá de toda la información necesaria para que el brazo robótico pueda interactuar con cada pieza.

6.1. Análisis de profundidad

Como ya se ha mencionado anteriormente, para este proyecto se ha empleado la cámara Intel Realsense D435 [Intc]. Esta cámara se caracteriza por contar con múltiples sensores y para este proyecto se va a hacer uso de los sensores RGB y de profundidad. El análisis de la imagen de color ya se ha llevado a cabo en los Capítulo 3 y Capítulo 5 y en esta sección se va a desarrollar el proceso de análisis de la imagen de profundidad.

El primer paso es la obtención de la imagen de profundidad, este proceso se explica de forma detallada en el Apéndice A, la imagen devuelta por la cámara tiene una dimensión de 480x480x1 píxeles del tipo entero sin signo de 16 bits. Esto se debe a que el sensor es capaz de medir largas distancias de hasta 10 metros y por ello requiere de 16 bits. Esto para nosotros supone un problema a la hora de representar las imágenes ya que nuestros objetos suelen estar próximos a la cámara y debido a la escala estos son apenas reconocibles al mostrar las imágenes. Es por ello por lo que tras obtener la imagen de profundidad esta se reescala para poder ser mostrada. En el proceso de reescalado se pierde la capacidad de poder detectar objetos de forma precisa a más de un metro, pero por suerte esta situación nunca se da en este proyecto.

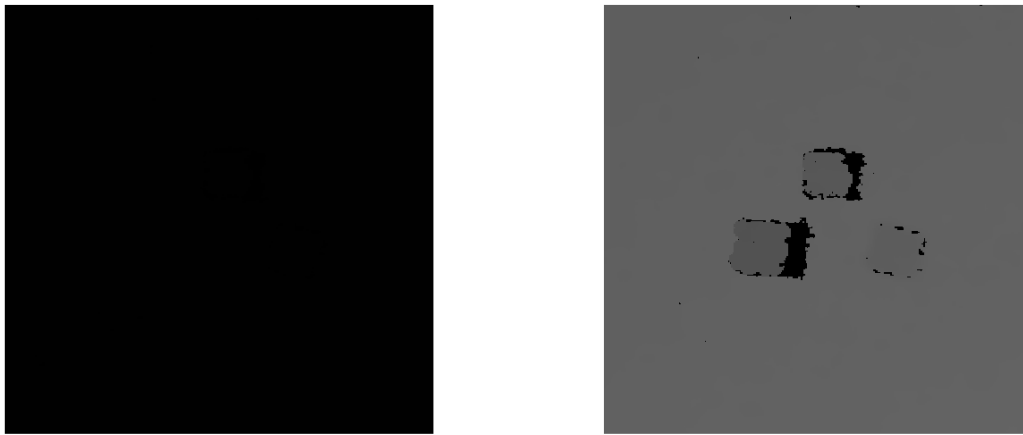


Figura 6.1. Demostración del escalado de imágenes de profundidad

El análisis de profundidad comienza por la calibración de la cámara, para poder trabajar desde diferentes puntos de operación y evitar variaciones en la profundidad por deterioro del *hardware*, es recomendable calibrar la cámara cada vez que se va a usar. El proceso para la calibración es simple y rápida, el objetivo es obtener un mapa de la profundidad del suelo para más adelante poder ser contrastada esta altura con la altura de cada pieza. Para obtener el mapa de profundidad, se divide la imagen de profundidad en 12x12 secciones y se calcula la mediana de profundidad de cada sección. Se ha decidido dividir la zona de trabajo en 12x12 secciones para así reducir la carga del programa y no tener que almacenar en memoria una imagen extra de profundidad. Además, de esta forma se reduce el efecto causado por los reflejos ya que estos se muestran como zonas de profundidad infinita. Se ha decidido emplear la mediana frente a la media debido a su robustez ante valores atípicos. Esto se puede apreciar alrededor de las piezas de LEGO, debido al material y a la orientación de la cara, la cámara no es capaz de detectar correctamente la profundidad y da valores excesivamente grandes o pequeños.

Una vez calibrada la cámara, ya se puede empezar a determinar la altura de cada pieza, para ello es necesario haber hallado previamente la posición de todas las piezas. El proceso de obtención de altura es individual e independiente del resto de piezas. El primer paso es obtener la altura media de la pieza recortada y a continuación se contrasta frente al suelo. Teniendo en cuenta el escalado y la altura media de una pieza de LEGO se puede obtener el número de piezas apiladas. A continuación, se muestra un ejemplo en el que se puede observar todo el proceso en la Figura 6.3.

Gracias a la implementación del análisis por secciones, al trabajo con la mediana y al alineamiento de la imagen de profundidad se ha conseguido mitigar notablemente los errores producidos por los reflejos. Esto se debe a que los reflejos distorsionan la imagen de profundidad y hacen que no se pueda obtener la profundidad de las regiones de la imagen afectadas por estos. Al trabajar con secciones la posibilidad de que toda una sección se vea cubierta por un reflejo es baja y por lo tanto es poco probable que no se pueda realizar un análisis de profundidad.

$$altura = (suelo - pieza) * desescalado / 20 = (390 - 330) / 20 = 3 \quad (6.1)$$

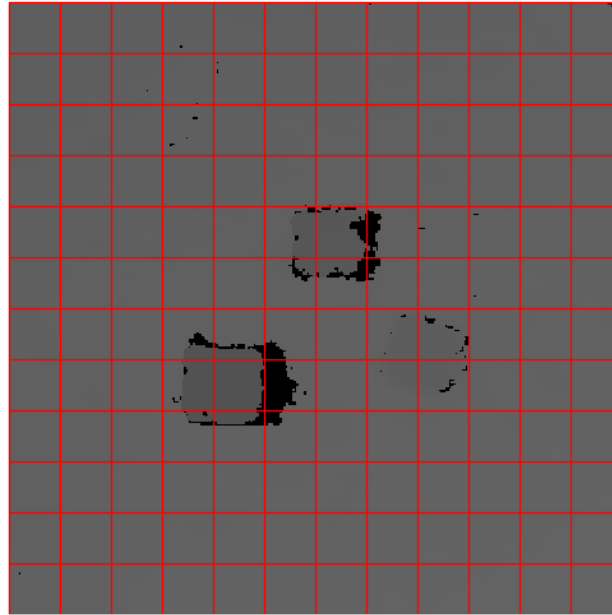


Figura 6.2. Mallado de imágenes de profundidad para determinación de altura

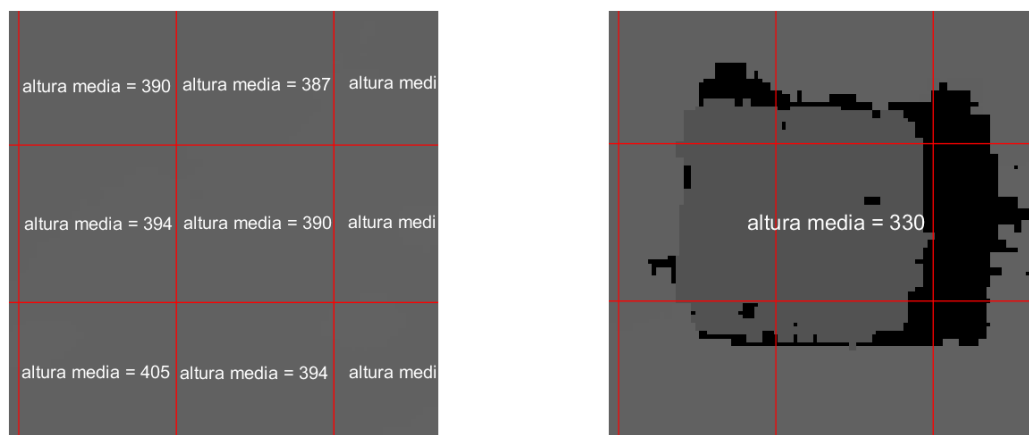


Figura 6.3. Demostración del proceso de cálculo de profundidad tras aplicar el desescalado

6.2. Análisis de la orientación

Para poder recoger las piezas con el brazo robótico es necesario saber la orientación de cada pieza. El brazo ha sido programado para orientar su cabezal y pinza de forma que siempre intente coger la pieza de LEGO con un buen agarre pinzando caras planas. El ángulo también nos permite saber si las piezas han sido encajadas en la base del suelo o por el contrario están sueltas. Esta información es vital ya que dependiendo del estado de la pieza, hay diversos métodos por los que el brazo intenta extraer la pieza. Por ello se han desarrollado dos métodos diferentes para la estimación del ángulo.

El primer método a tratar es el método originalmente desarrollado por Ana Berjón Valles [Ber] y se basa en la detección de borde y la transformada de Hough. Pero ha sido perfeccionado para mejorar su precisión y evitar situaciones en las que no se pueda detectar ningún borde. El segundo método consiste en el uso de las redes neuronales creadas en el Capítulo 4 pero con modificaciones para convertirlas en modelos de regresión basados en redes neuronales.

6.2.1. Detección de borde y transformada de Hough

Tal y como su nombre indica, este método consiste en un análisis empleando la transformada de Hough y el algoritmo de paso por cero para la detección de bordes. Los detectores de borde y la transformada de Hough ya han sido ya explicados en la Sección 2.1.1. A continuación, se desarrolla el proceso de análisis de una pieza:

- El primer paso para obtener la orientación de las piezas consiste en aplicar un filtro de color idéntico al visto en Sección 3.1 con el fin de obtener una imagen binaria en la que solo se muestren las piezas. Como ya se ha mencionado anteriormente, el filtrado por color presenta grandes problemas de inconsistencia ante cambios de iluminación. Una vez aplicado el filtro se pasa a recortar las piezas y a ser analizadas por separado. Este proceso se puede ver en la Figura 6.4.
- El segundo paso parte de las piezas ya identificadas y separadas. Consiste en aplicar un algoritmo de detección de borde para remarcar solo los bordes exteriores de las piezas. El algoritmo empleado se conoce como paso por cero y se caracteriza porque tal y como su nombre indica, busca los bordes en la imagen que pasen por cero. Es decir, el algoritmo solo resalta las zonas en las que el valor del laplaciano pase por cero ya que son las zonas en las que se producen bordes bruscos. Este proceso se puede ver en la Figura 6.5
- El tercer paso se trata de la transformada de Hough, esta consiste en analizar cada punto de la imagen y hacer pasar por este todos los patrones posibles. Una vez analizados todos los puntos se determinan las relaciones entre estos y se puede extraer información de la forma del objeto. Este proceso se puede ver en la Figura 6.6
- El cuarto paso trata de analizar la información extraída con la transformada de Hough. En nuestro caso el patrón buscado son rectas, por ello primero se analizará todos los puntos para posteriormente determinar las rectas más importantes y de ellas se extraerá el ángulo. Se puede observar los resultados de este análisis en la Tabla 6.1

Con la ayuda de MATLAB se ha desarrollado un script con el objetivo de comprobar la eficacia de este método. Para ello se han tomado múltiples fotos de piezas de LEGO con una orientación lo más próxima posible a 0° y se han rotado de forma aleatoria hasta obtener 300 fotos. Con estas fotos se pretende reflejar lo mejor posible la realidad aunque dada las limitaciones físicas y de tiempo las fotos han tenido que ser hechas con un trípode y una cámara réflex. Esto implica que las condiciones no son idénticas a las del laboratorio. Además, la cámara tiene una resolución mucho mayor que la cámara Realsense D435 y esto se refleja en la evaluación. Como el sistema tiene más información los resultados son muy precisos. En la realidad aunque los resultados son buenos, debido a la falta de resolución estos no son tan precisos. Se puede observar en la Figura 6.7 el histograma de las imágenes empleadas para la evaluación.

- Error medio: Es el error medio cometido al calcular la orientación de todas las piezas.
 $Error\ medio = 1,18^\circ$
- Error máximo: Error máximo cometido al analizar las 300 piezas. $Error\ máximo = 26^\circ$

- Desviación típica: Representa la dispersión del conjunto de datos. $\sigma = 2,5193^\circ$
- Precisión: Se define como el porcentaje de predicciones cuyo error es inferior a un umbral definido. Se ha decidido que el umbral para determinar el nivel de precisión sea 1° . $Precisión = 82,67\%$
- Diagrama de cajas: Permite analizar gráficamente el comportamiento del modelo. Se puede observar el diagrama de cajas en la Figura 6.8 Viendo el primer y tercer cuartil podemos ver que la mayoría de las predicciones tienen un error menor a 1° . También se puede observar la numerosa presencia de valores atípicos. Estos son aquellos cuya desviación sea mayor a 1.5 veces la del recorrido intercuartílico. Se observa hasta un caso con un error de 25 grados.

Este método se caracteriza por ser rápido y bastante preciso en la mayoría de las situaciones. Desgraciadamente, no es lo suficientemente constante. Esto se debe al gran número de valores atípicos.

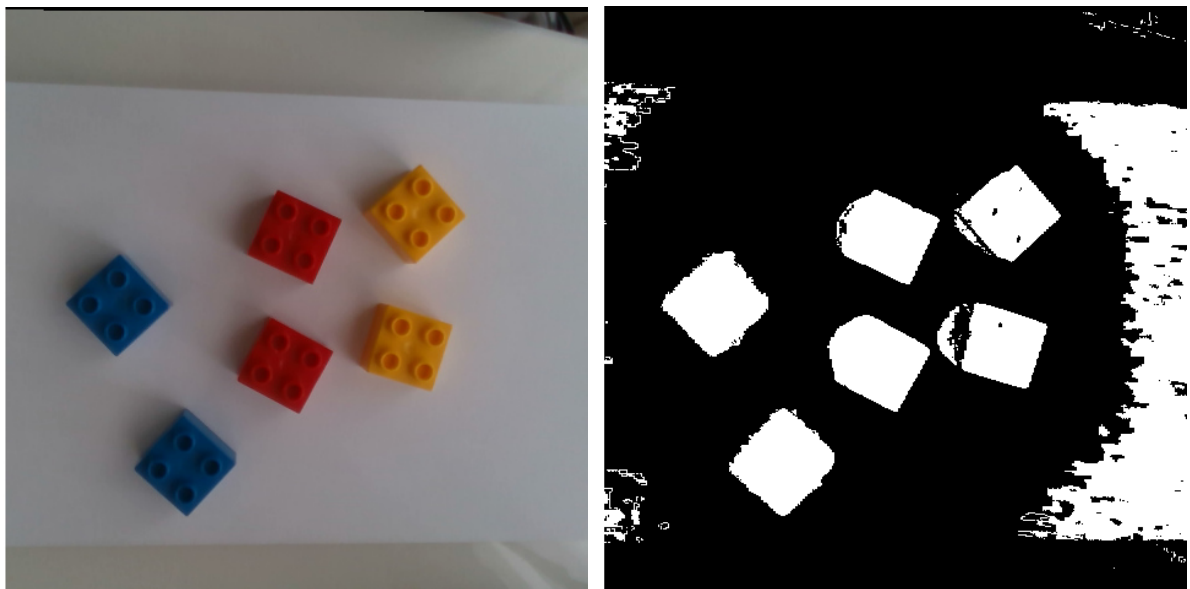


Figura 6.4. Primer paso: Filtrado por color para el análisis de la orientación

Punto 1	Punto 2	Theta	Rho
(27,75)	(59,108)	-43	-31
(49,36)	(62,24)	45	59
(69,105)	(80,95)	45	122
(86,88)	(93,82)	45	122
(101,74)	(106,69)	45	122
(31,52)	(50,35)	48	58
(54,31)	(62,24)	48	58
(41,89)	(51,100)	-46	-36

Tabla 6.1. Cuarto paso: Análisis de los resultados de la transformada de Hough y extracción de las rectas más significativas

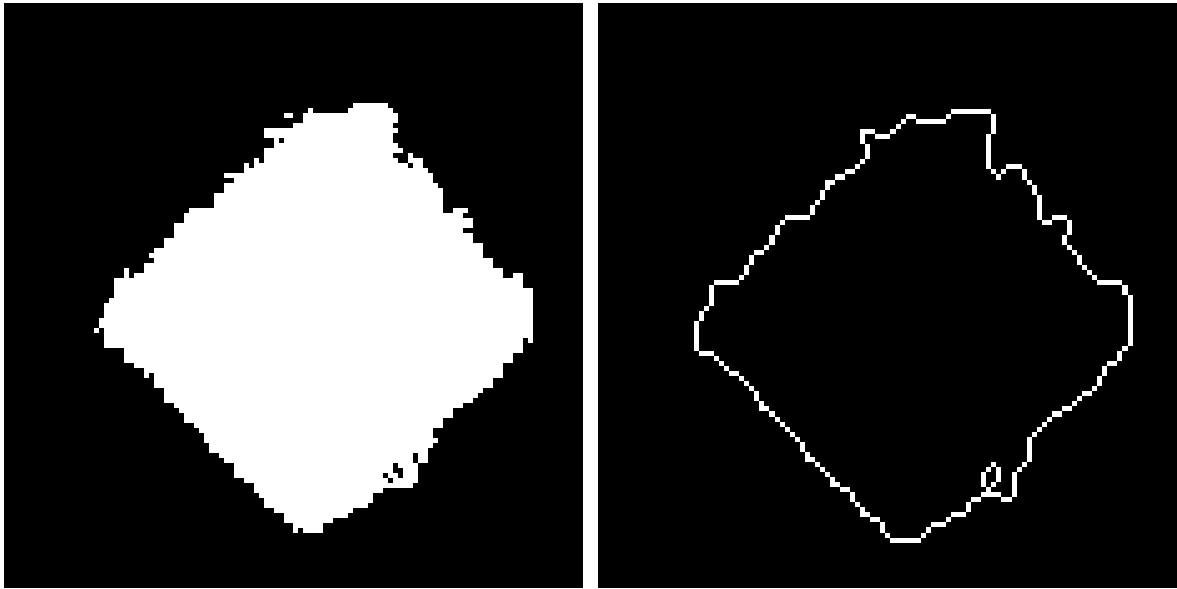


Figura 6.5. Segundo paso: Separación y obtención de bordes para el análisis de la orientación

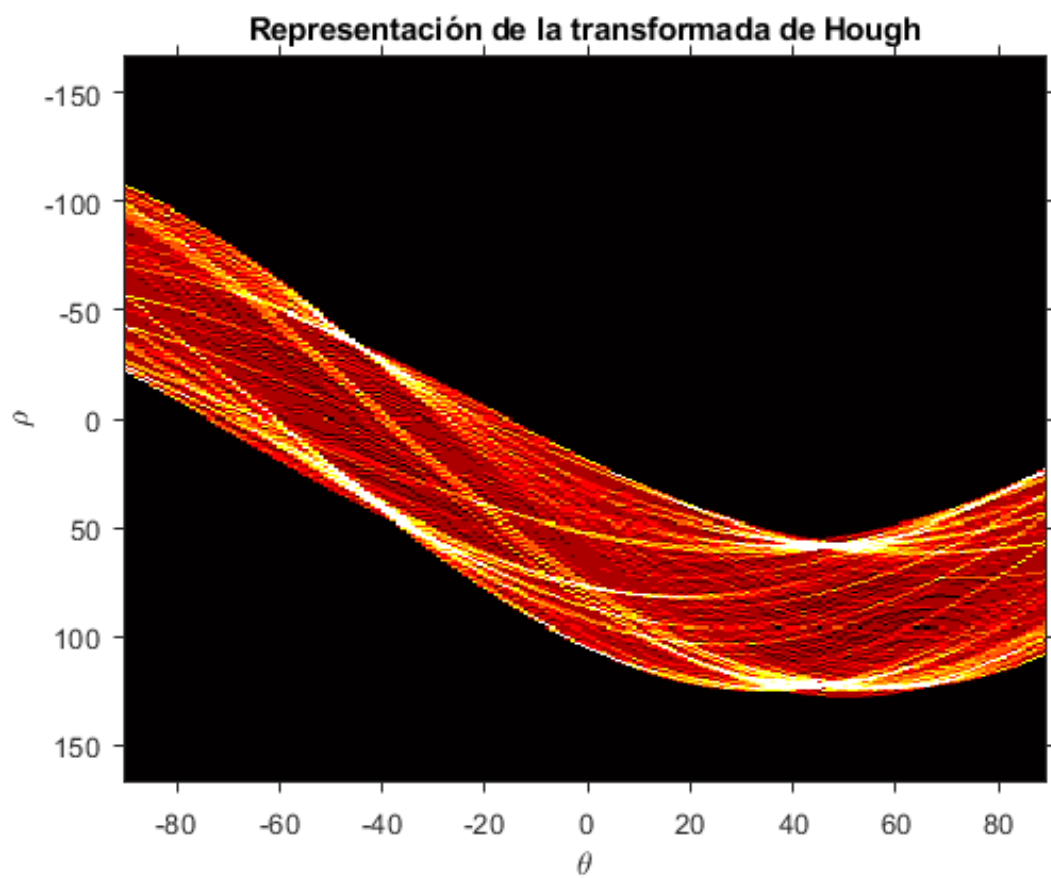


Figura 6.6. Tercer paso: Análisis de una pieza mediante la transformada de Hough

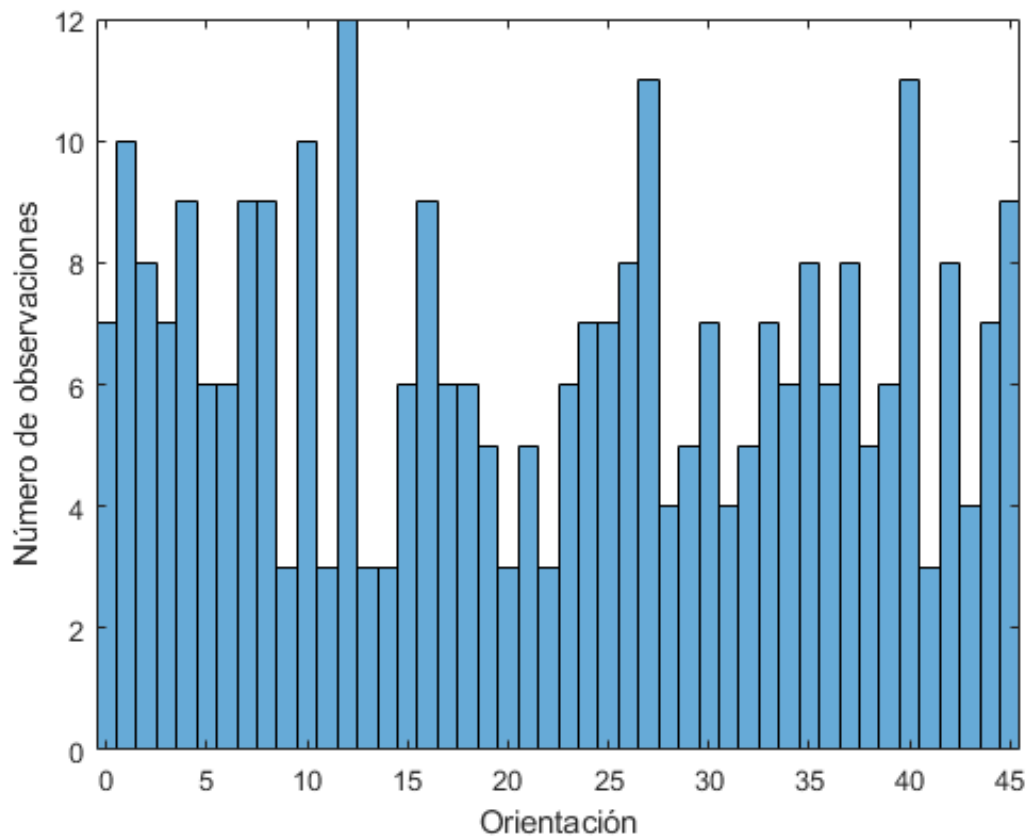


Figura 6.7. Histograma de imágenes empleadas para la evaluación del cálculo de la orientación por la transformada de Hough

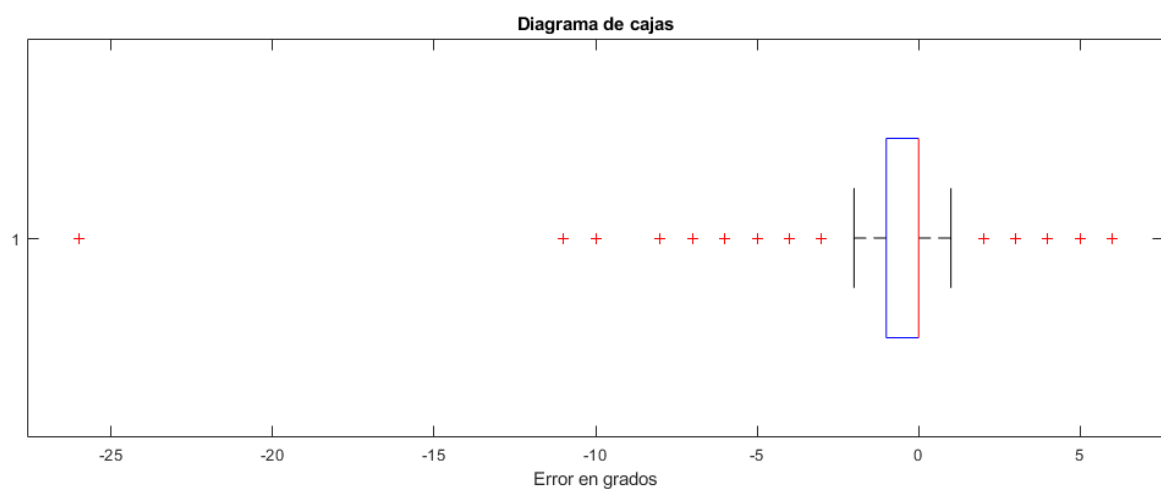


Figura 6.8. Diagrama de cajas de la evaluación del cálculo de la orientación por la transformada de Hough

6.2.2. Modelo de regresión con redes neuronales

Las redes neuronales convolucionales tienen la característica de que pueden ser usadas con múltiples fines. Ya hemos visto en los Capítulo 4 y Capítulo 5 dos ejemplos de usos para esta tecnología y a continuación, se van a modificar ambos clasificadores ya creados con el objetivo de convertirlos en un modelo de regresión capaz de estimar el ángulo de rotación.

Los modelos de regresión son uno de los primeros usos que se les dio a las redes neuronales y se caracterizan porque la salida no es una clase o la posición de un objeto, en lugar es uno o varios números que representan algún atributo de la imagen. Los modelos de regresión no son exclusivos para el análisis de imágenes, tienen multitud de usos y el uso más común es el análisis estadístico.

Preparativos para el entrenamiento y evaluación

Para el entrenamiento de la red se ha preparado una base de datos con la ayuda de MATLAB. El proceso ha sido la captura de imágenes de alta resolución con piezas de LEGO sin rotación. A continuación, las piezas de LEGO son recortadas de la imagen original y son rotadas múltiples veces. Con esta técnica, se ha conseguido transformar 225 imágenes en un total de 10350 imágenes. Para intentar mostrar todas las posiciones posibles de las piezas de LEGO. Se han tomado fotos en tres escenas diferentes y para cada escena se han tomado 25 fotos para cada pieza de LEGO de color. A continuación, se muestra un montaje con el conjunto de 25 imágenes tomadas para un color en una escena. A todas las imágenes tomadas, se les ha aplicado una rotación para todos los ángulos comprendidos entre 0° y 45° y por último se han reescalado al tamaño adecuado para la red. En el caso de LEGONet esta es 227×227 y en el caso de LEGO16 esta es 224×224 .

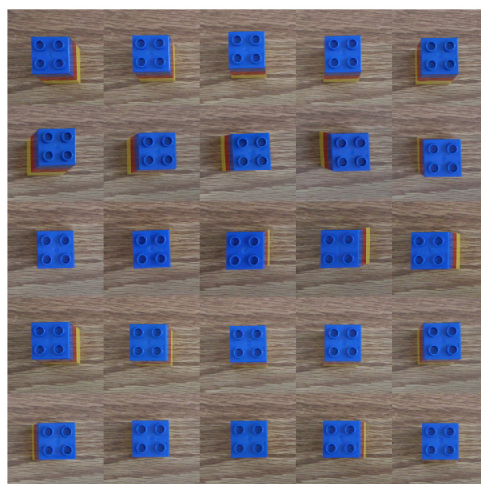


Figura 6.9. Imágenes para el entrenamiento de los modelos de regresión

Con este conjunto de imágenes se ha llevado a cabo el entrenamiento de un modelo de regresión basado en LEGONet y se ha llevado a cabo en un ordenador personal equipado con un i7 4790K, 16GB de memoria RAM y una tarjeta gráfica Nvidia Geforce GTX970 con 4GB de VRAM. Teniendo en cuenta las limitaciones por *hardware* y tiempo, se han realizado múltiples entrenamientos con diferentes opciones de entrenamiento para obtener los mejores resultados. Se ha decidido que el 20 % de las imágenes sean destinadas para validación y el restante para entrenamiento.

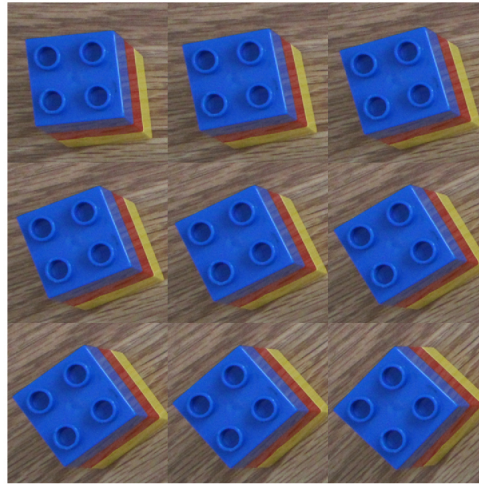


Figura 6.10. Aumento de imágenes para el entrenamiento de los modelos de regresión

Con la ayuda de MATLAB se ha desarrollado un *script* con el objetivo de comprobar la eficacia de este método. Para ello se han tomado 300 fotos de piezas de LEGO con diferentes orientaciones. Con estas fotos se pretende reflejar lo mejor posible la realidad, aunque dada las limitaciones físicas y de tiempo las fotos han tenido que ser hechas con un trípode y una cámara réflex. Esto implica que las condiciones no son idénticas a las del laboratorio. Además, la cámara tiene una resolución mucho mayor que la cámara Realsense D435 y esto se refleja en la evaluación. Como la red tiene más información los resultados son muy precisos. En la realidad, aunque los resultados son muy buenos, debido a la falta de resolución estos no son tan precisos. Se puede observar en la Figura 6.11 el histograma de las imágenes empleadas para la evaluación. En caso de que haya diferencia debido a la diferencia de resolución, esto se puede solventar tomando una foto próxima a la pieza de LEGO.

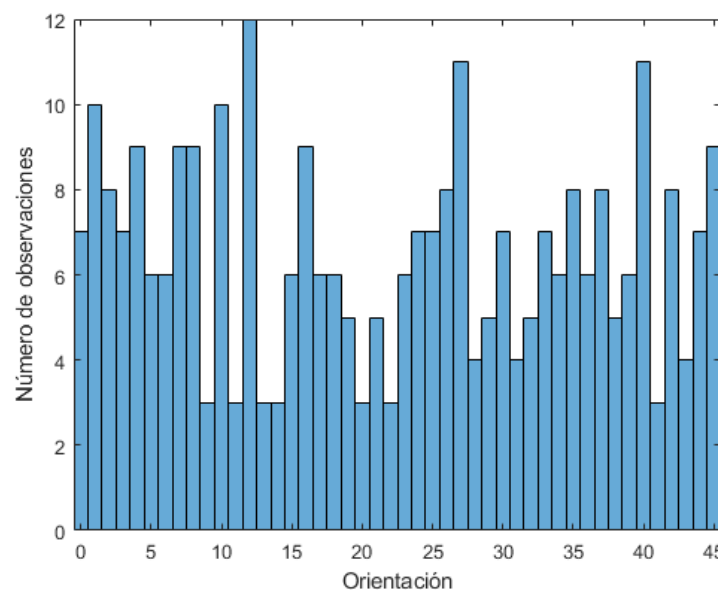


Figura 6.11. Histograma de imágenes empleadas para la evaluación del cálculo de la orientación

Modelo de regresión basado en LEGONet

En el Capítulo 4 se ha desarrollado un clasificador basado en AlexNet y denominado LEGONet, este al igual que AlexNet cuenta con 13 millones de parámetros y 650.000 neuronas. Para poder reutilizar un clasificador y transformarlo, el primer paso consiste en la eliminación de las tres últimas capas. Es decir, la eliminación de la última capa completamente conectada, la eliminación de la capa *Softmax* y la eliminación de la capa de salida del clasificador. En el caso de LEGONet ha sido necesario eliminar más capas ya que las capas *Dropout* interferían con el entrenamiento. Por ello se ha eliminado todo a partir de la capa *relu6* y se ha introducido capas completamente conectadas intercaladas con capas *Batch normalization* y capas *ReLU*. La estructura obtenida se muestra en la Figura 6.12.

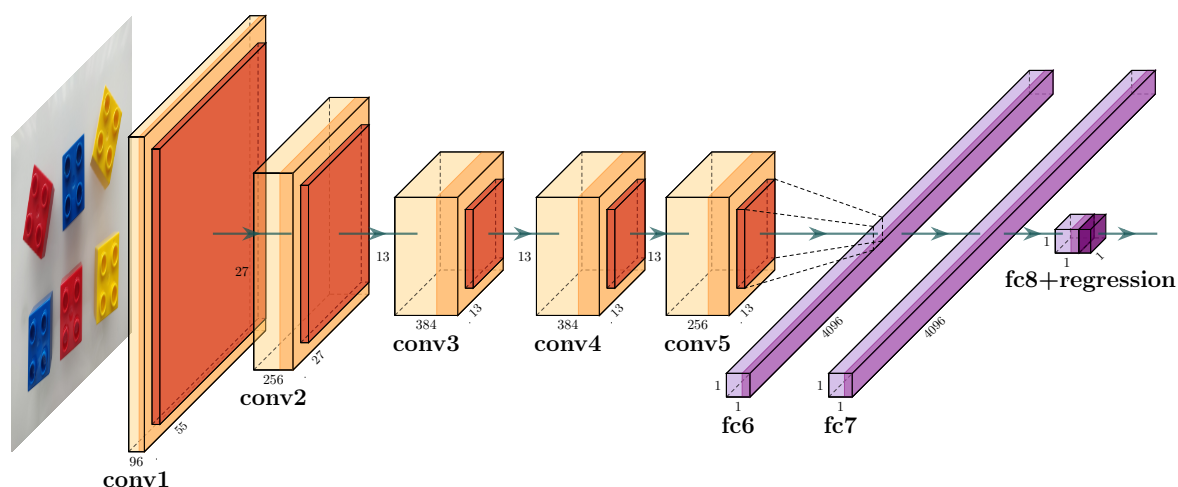


Figura 6.12. Estructura de un modelo de regresión basado en LEGONet

Para el entrenamiento de esta red se ha usado la base de datos diseñada en la Sección 5.1. Esta base de datos consta de un total de 10.350 piezas de LEGO. El 20 % de estas piezas serán empleadas para la validación del entrenamiento. Se han realizado numerosas pruebas de entrenamiento y se ha descubierto que las mejores opciones de entrenamiento son las mostradas en la Tabla 6.2.

Opciones de entrenamineto	
Solver	Stochastic Gradient Descent with Momentum (SGDM)
Momentum	0.9
Initial Learn Rate	1.00E-03
Learn Rate Schedule	piecewise
Learn Rate Drop Factor	0.1
Learn Rate Drop Period	80
L2Regularization	0.004
Max Epochs	100
Mini Batch Size	200
ValidationData	ValidatingData
ValidationFrequency	41
Shuffle data	every epoch

Tabla 6.2. Opciones de entrenamiento del modelo de regresión basado en LEGONet

Con la ayuda de MATLAB se ha desarrollado un script con el objetivo de comprobar la eficacia de este método. Para ello se van a analizar un total de 300 piezas de LEGO de diferentes colores y con diferentes orientaciones. A continuación, se analizan los resultados obtenidos.

- Error medio: Es el error medio cometido al calcular la orientación de todas las piezas.
 $Error\ medio = 0,3327^\circ$
- Error máximo: Error máximo cometido al analizar las 300 piezas. $Error\ máximo = 1,3^\circ$
- Desviación típica: Representa la dispersión del conjunto de datos. $\sigma = 0,4293^\circ$
- Precisión: Se define como el porcentaje de predicciones cuyo error es inferior a un umbral definido. Se ha decidido que el umbral para determinar el nivel de precisión sea 1° .
 $Precisión = 97,67\%$
- Diagrama de cajas: Permite analizar gráficamente el comportamiento del modelo. Se puede observar el diagrama de cajas en la Figura 6.13 Viendo el primer y tercer cuartil podemos ver que la mayoría de las predicciones tienen un error menor a $0,5^\circ$. También se puede observar la pequeña presencia de valores atípicos. Estos son aquellos cuya desviación sea mayor a 1.5 veces la del recorrido intercuartílico. Son pocas las situaciones en las que se ha superado un error de 1° y el mayor error cometido es de $1,3^\circ$

Este método se caracteriza por ser rápido, aunque depende del número de piezas analizar. Esto se debe a que es necesario correr la red neuronal tantas veces como piezas se hayan detectado. Además de rápido también destaca por ser constante y preciso. De esta forma podemos asegurarnos que el brazo robótico nunca tenga problemas para recolectar las piezas de LEGO.

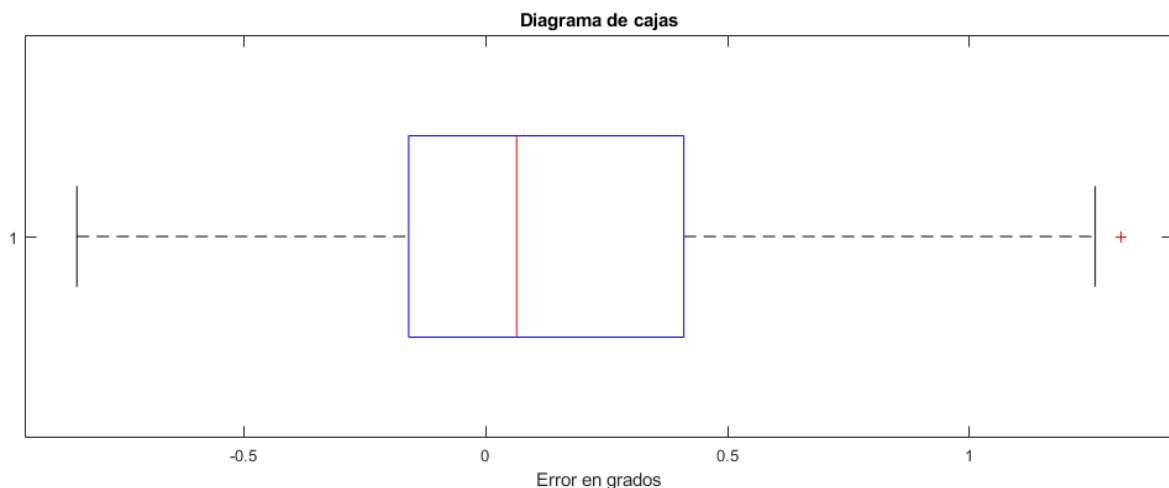


Figura 6.13. Diagrama de cajas de la evaluación del cálculo de la orientación por el modelo de regresión basado en LEGONet

Modelo de regresión basado en LEGO16

En el Capítulo 4 se ha desarrollado un clasificador basado en VGG-16 y denominado LEGO16, este al igual que VGG-16 cuenta con 138 millones de parámetros y 13 millones de neuronas. Para poder reutilizar un clasificador y transformarlo, el primer paso consiste en la eliminación de las tres últimas capas. Es decir, la eliminación de la última capa completamente conectada, la eliminación de la capa *Softmax* y la eliminación de la capa de salida del clasificador. Al igual que con LEGONet, ha sido necesario eliminar más capas ya que las capas *Dropout* impedían el

aprendizaje de la red. Por ello se ha eliminado a partir de la capa relu6. Como se ha cortado la red justo después de la primera capa completamente conectada, las primeras capas en añadirse son una capa *Batch normalization* y una capa *ReLU*. A continuación se añaden dos capas completamente conectadas acompañadas también por capas *Batch normalization* y *ReLU*.

En la publicación original de VGG-16 [SZ14] se menciona que descartan el uso de capas *Batch* debido a que en lugar de aportar a la red, entorpecían su entrenamiento. En una primera instancia se intentó hacer lo mismo al crear el modelo de regresión, es decir, no usar capas *Batch*. Desgraciadamente al intentar entrenar estas redes el sistema se negaba a aprender y por ello se optó por la estructura con capas *Batch*.

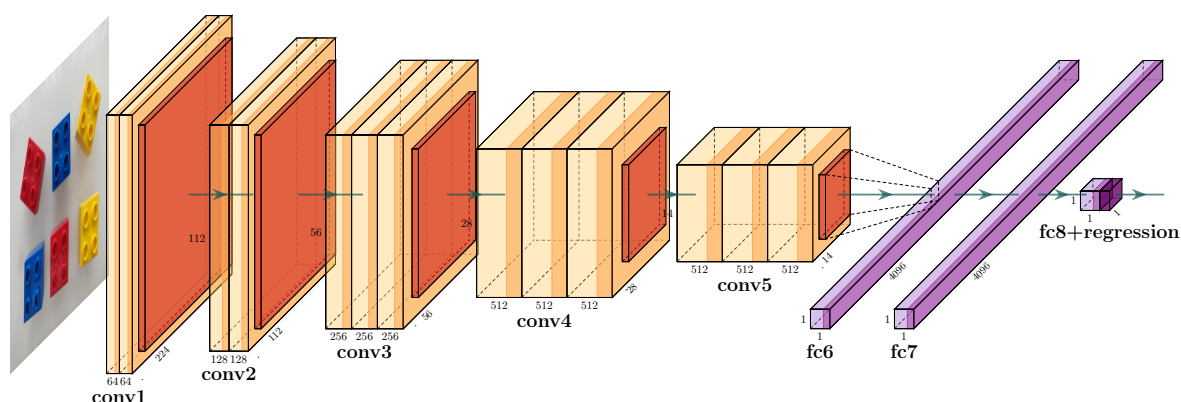


Figura 6.14. Estructura de un modelo de regresión basado en LEGO16

Para el entrenamiento de esta red se ha usado la base de datos diseñada en la Sección 5.1. Esta base de datos consta de un total de 10.350 piezas de LEGO. El 20 % de estas piezas serán empleadas para la validación del entrenamiento. Se han realizado numerosas pruebas de entrenamiento y se ha descubierto que las mejores opciones de entrenamiento son las mostradas en la Tabla 6.3.

Opciones de entrenamineto	
Solver	Stochastic Gradient Descent with Momentum (SGDM)
Momentum	0.9
Initial Learn Rate	1.00E-03
Learn Rate Schedule	piecewise
Learn Rate Drop Factor	0.1
Learn Rate Drop Period	7
L2Regularization	0.004
Max Epochs	20
Mini Batch Size	10
ValidationData	ValidatingData
ValidationFrequency	828
Shuffle data	every epoch

Tabla 6.3. Opciones de entrenamiento del modelo de regresión basado en LEGO16

Con la ayuda de MATLAB se ha desarrollado un script con el objetivo de comprobar la eficacia de este método. Para ello se van a analizar un total de 300 piezas de LEGO de diferentes colores y con diferentes orientaciones. A continuación, se analizan los resultados obtenidos.

- Error medio: Es el error medio cometido al calcular la orientación de todas las piezas. $Error\ medio = 0,7075^\circ$
- Error máximo: Error máximo cometido al analizar las 300 piezas. $Error\ máximo = 15^\circ$
- Desviación típica: Representa la dispersión del conjunto de datos. $\sigma = 1,2336^\circ$
- Precisión: Se define como el porcentaje de predicciones cuyo error es inferior a un umbral definido. Se ha decidido que el umbral para determinar el nivel de precisión sea 1° . $Precisión = 79,33\%$
- Diagrama de cajas: Permite analizar gráficamente el comportamiento del modelo. Se puede observar el diagrama de cajas en la Figura 6.15. Viendo el primer y tercer cuartil podemos ver que la mayoría de las predicciones tienen un error menor a 1° . También se puede observar que son escasos los casos en los que el error cometido es mayor a 2° . Sin embargo, se ha dado una situación en la que el error es de 16° .

Este método se basa en una red neuronal más potente y capaz que LEGONet. Sin embargo, el entrenamiento o la estructura no han sido los adecuados y por ello no llega a superar a LEGONet. Además, al ser una red mayor que LEGONet requiere de más potencia y tiempo de ejecución. Con más tiempo y entrenamiento es posible que se pueda encontrar una red basada en LEGO16 capaz de superar a LEGONet.

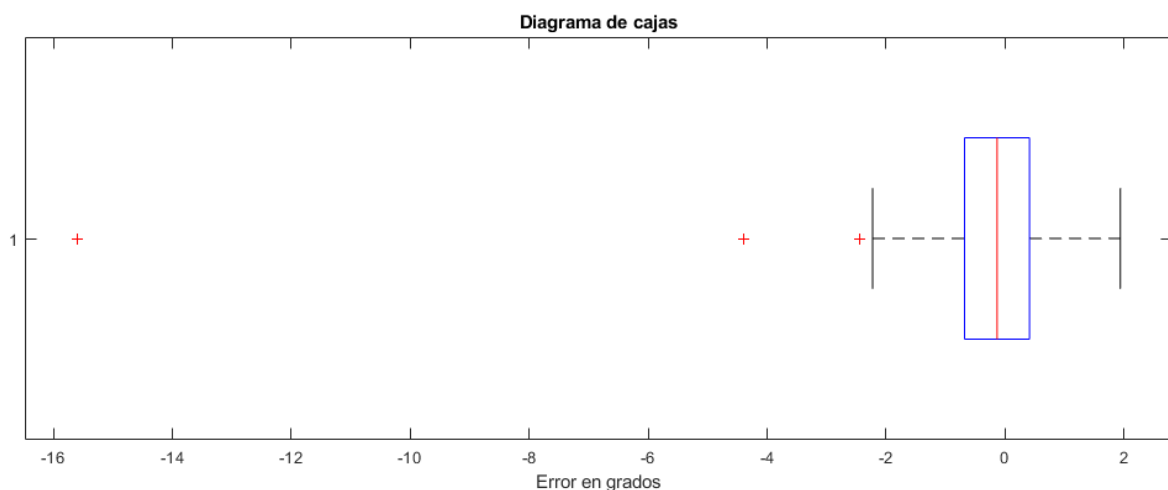


Figura 6.15. Diagrama de cajas de la evaluación del cálculo de la orientación por el modelo de regresión basado en LEGO16

Análisis de los resultados

En este capítulo se llevará a cabo un comparativo de todos los métodos creados para la detección de piezas así como los creados para la estimación de la orientación de piezas. Se realizará un estudio de precisión, robustez y rapidez de los sistemas creados.

7.1. Análisis de la segmentación

Como ya se ha mencionado, se han desarrollado múltiples métodos para llevar a cabo el proceso de segmentación. Y todos tienen sus ventajas y desventajas y es importante conocerlas para así poder seleccionar que método o combinación de métodos emplear para la implantación con el brazo robótico. Es por ello que a continuación se van a comparar estos métodos tanto en precisión y tasa de fallos como en velocidad. Para ello se van a usar los resultados ya mostrados en este documento al desarrollar cada uno de estos métodos.

7.1.1. Precisión, Exhaustividad, Tasa de fallos y FPPI

Si se analizan los resultados se pueden observar dos patrones:

- Segmentación por color: Al comparar los resultados de este método frente a las redes neuronales se muestra la superioridad de estas. Este método solo puede competir al detectar piezas amarillas y esto es debido a que en las fotos analizadas no hay ningún otro objeto de color amarillo.
- AlexNet y VGG-16: al comparar estas redes en igualdad de condiciones se observa claramente la superioridad de VGG-16. Al tratarse de una red más compleja es capaz de aprender más y por ello ha dado mejores resultados siempre que ha podido ser entrenada.

Si nos centramos solo en las redes neuronales, también se pueden extraer múltiples conclusiones:

- R-CNN: Destaca por ser uno de los métodos más precisos y seguros, solamente es superado por YOLO basado en VGG-16. Al hacer una comparativa justa frente al resto de métodos basados también en LEGONet se puede apreciar el verdadero potencial de R-CNN. Se trata de un método muy capaz, aunque muy poco eficiente.

- **Faster R-CNN:** Se trata de una variante de R-CNN cuyo objetivo es mejorar la eficiencia del mismo reduciendo bastante el procesamiento, pero al hacerlo también pierde en capacidades. Ambas redes del tipo Faster R-CNN pierden frente a R-CNN basado en LEGONet pero a cambio son bastante más eficientes. Se puede observar en ellas la diferencia entre AlexNet y VGG-16 ya que esta segunda ha dado resultados mucho más positivos y se han aproximado bastante a R-CNN basado en AlexNet.
- **YOLO:** Este tipo de redes se caracterizan por dar buenos resultados con un entrenamiento muy rápido además de ser bastante eficientes. Al analizar los resultados podemos ver que esto es cierto, las redes han aprendido rápidamente y siendo muy eficientes consiguen dar muy buenos resultados. Aun así, se puede observar que R-CNN basado en LEGONet es superior a YOLO basado en LEGONet aunque queda por detrás de YOLO basado en VGG-16. Esto muestra la superioridad de R-CNN al detectar piezas y la superioridad de VGG-16 frente a AlexNet.

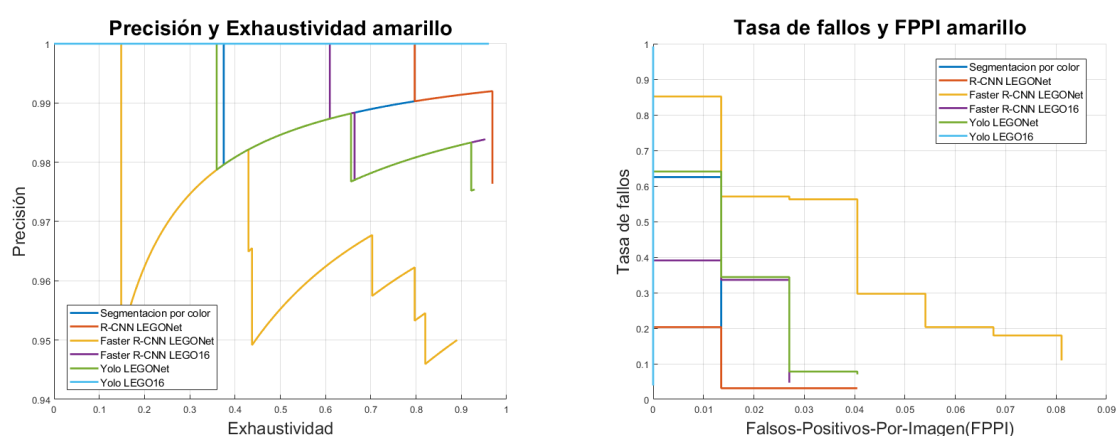


Figura 7.1. Comparativa de los métodos de segmentación al detectar piezas amarillas

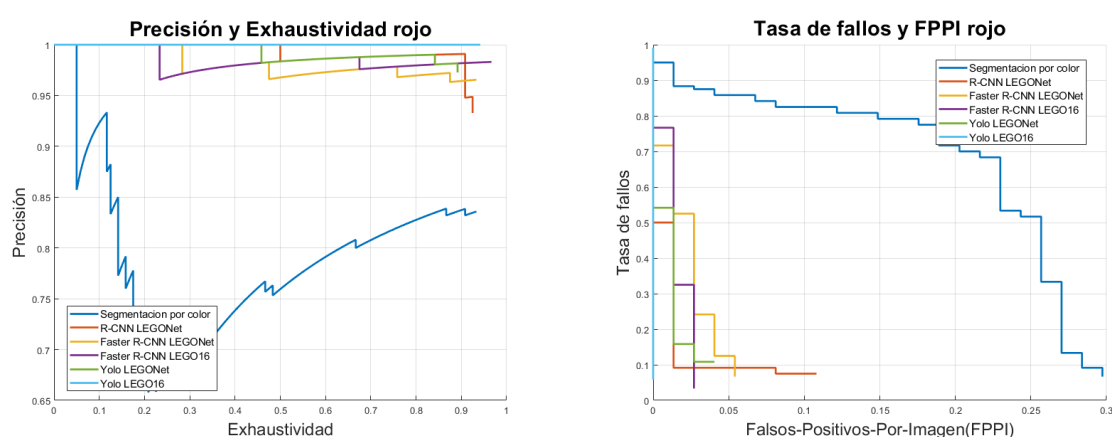


Figura 7.2. Comparativa de los métodos de segmentación al detectar piezas rojas

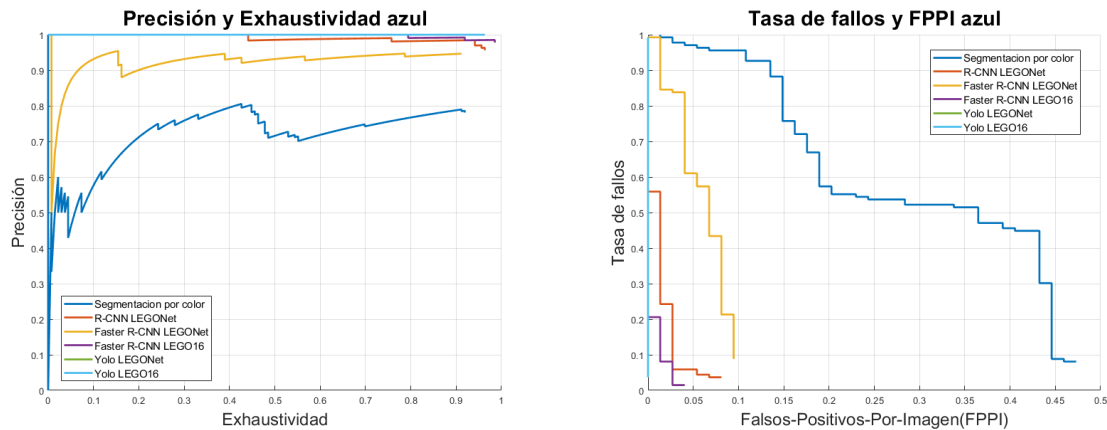


Figura 7.3. Comparativa de los métodos de segmentación al detectar piezas azules

7.1.2. Velocidad

En la situación actual del proyecto no se depende en gran medida de la velocidad en el procesamiento de la imagen ya que no se ha implantado un sistema de seguimiento en tiempo real. En su lugar, desde una posición predefinida se toma una imagen y se analiza y calcula las posiciones de las piezas. A continuación, estas coordenadas son mandadas al robot para que recolecte las piezas y luego volver a la posición original y repetir el proceso. Sin embargo, con la ayuda de estas redes neuronales, el análisis es lo suficientemente rápido como para plantearse la implantación de un sistema de seguimiento en tiempo real. Es por ello por lo que a continuación se va a hacer una comparativa de la velocidad de todos los métodos desarrollados para la segmentación.

- Segmentación por color: este método en su estado actual debe de ser descartado como candidato a ser implantado ya que no consigue resultados suficientemente buenos y además tampoco destaca por su eficiencia y rapidez.
- R-CNN: Como ya se ha comentado, es uno de los métodos más capaces para segmentar, pero a cambio se caracteriza por ser el más lento. Para poder conseguir buenos resultados con este método se debe de sacrificar la velocidad.
- Faster R-CNN: Presenta una mejora muy notable en velocidad. En nuestro caso Faster R-CNN basado en AlexNet ha conseguido una velocidad 50 veces superior a la obtenida por R-CNN basado en AlexNet. Esto demuestra la mejora sustancial en velocidad que se consiguió con este método. Aunque para ello se ha sacrificado en capacidad de segmentación.
- YOLO: Yolo ha sido uno de los métodos que mejores resultados ha obtenido a la vez de ser el más rápido. Esto demuestra la verdadera capacidad de esta tecnología que ha conseguido superar la barrera de las treinta imágenes por segundo.

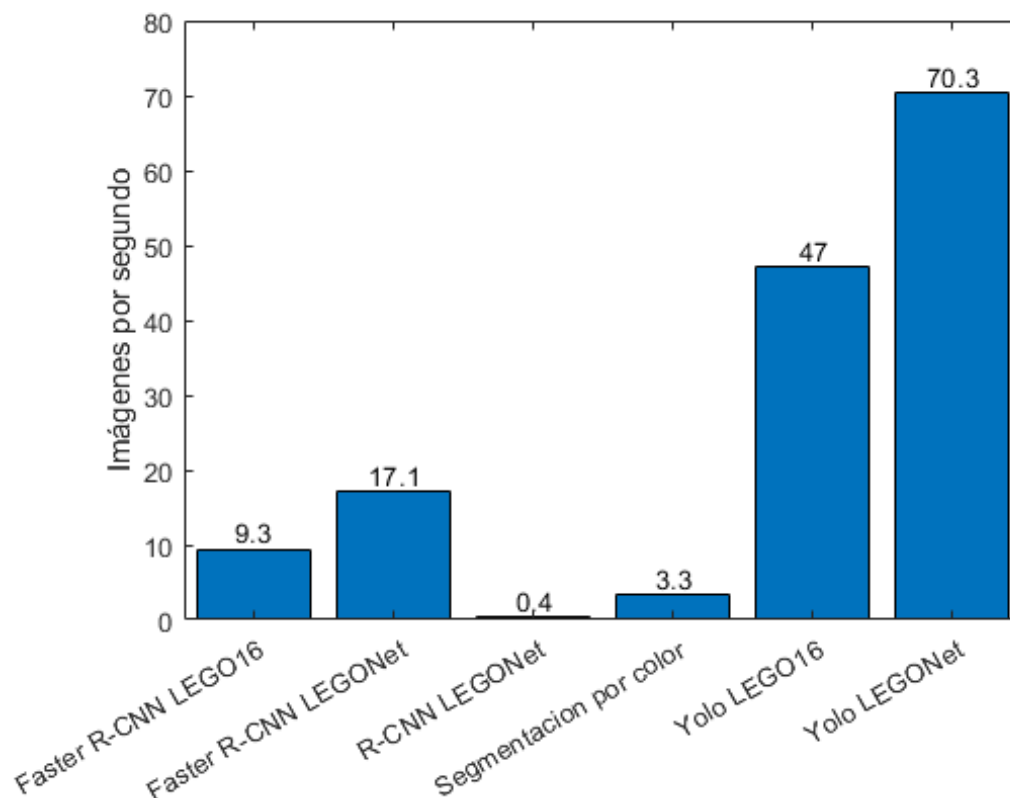


Figura 7.4. Diagrama de barras: Comparación de la velocidad de diferentes métodos para la segmentación (más alto es mejor)

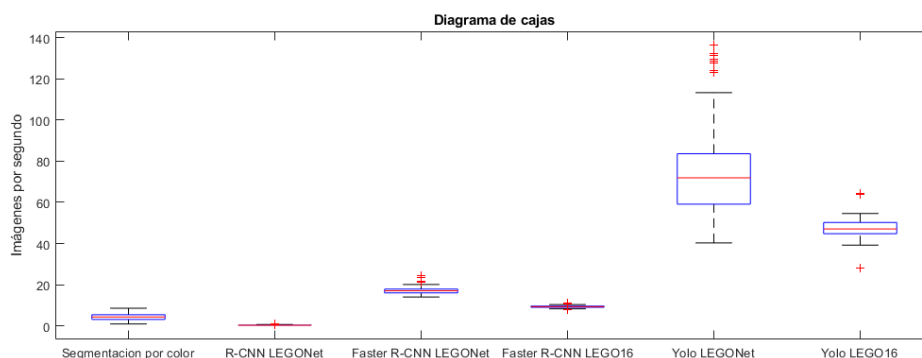


Figura 7.5. Diagrama de cajas: Comparación de la velocidad de diferentes métodos para la segmentación (más alto es mejor)

7.2. Análisis del cálculo de la orientación

En este proyecto también se han desarrollado múltiples métodos para el cálculo de la orientación es por ello que en esta sección se van a mostrar y comparar los resultados de los tres métodos.

7.2.1. Precisión

Los tres métodos destacan por ofrecer buenos resultados, aunque hay un método claramente superior al resto. A continuación, se muestran los resultados de los tres métodos para su comparación.

- Transformada de Hough: Se observa que es un método bastante preciso ya que el error medio no es elevado. Sin embargo, no es un método robusto ya que hay varias situaciones donde el error cometido es demasiado grande.
- LEGONet: Este método es el que mejor resultados ha dado con un error prácticamente nulo y una robustez clara. Demuestra el claro potencial de las redes neuronales.
- LEGO16: Este método ha dado resultados también bastante precisos, aunque ha cometido varios errores demasiado grandes y por ello falla en robustez. Se cree que esta red no ha sido bien diseñada o entrenada y que tiene un potencial mayor al mostrado.

	Transformada de Hough	LEGONet	LEGO16
Error medio	0.98°	0.33°	0.71°
Error máximo	26°	1.3°	16°
Precisión	83.67 %	97.67 %	79.33 %
Desviación típica	1.82°	0.43°	1.23°

Tabla 7.1. Comparación del error medio, error máximo, precisión y desviación típica de diferentes métodos para el cálculo de la orientación

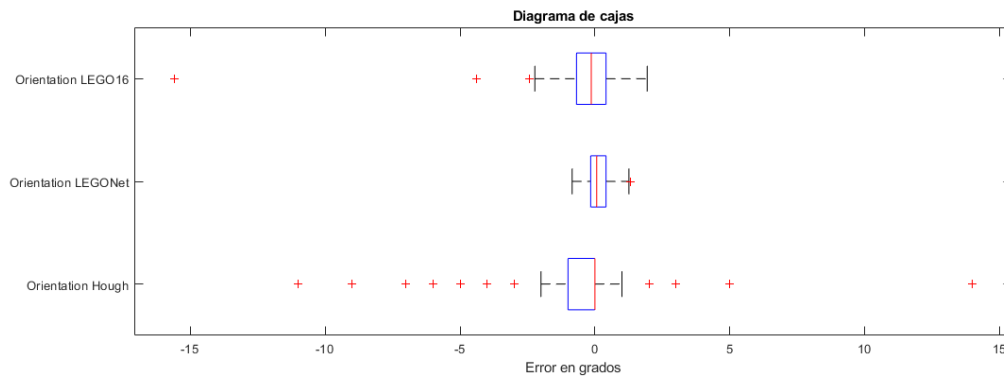


Figura 7.6. Diagrama de cajas: Comparación de la precisión de diferentes métodos para el cálculo de la orientación (cuanto más centrado respecto al cero y menos disperso mejor)

7.2.2. Velocidad

Al igual que con la segmentación, en este caso también se va a realizar un estudio de la velocidad de los diferentes métodos ya que esta tendrá un impacto en el rendimiento del sistema en su conjunto.

- Transformada de Hough: Se caracteriza por ser un método rápido y constante. El tiempo de análisis de cada pieza es muy similar y apenas hay variaciones.
- LEGONet: Este método es el más rápido de los tres, aunque es menos constante en su rapidez. También hay que tener en cuenta, que al ir más rápido un pequeño retraso al analizar una pieza tiene un mayor impacto.
- LEGO16: Este método ha dado resultados también bastante rápido aunque presenta numerosas situaciones en los que la velocidad ha reducido bastante.

En general los tres métodos son muy rápidos y todos han dado resultados superiores a las sesenta piezas por segundo, aunque es importante tener en cuenta que estos métodos se deben de correr tantas veces como piezas haya en la imagen.

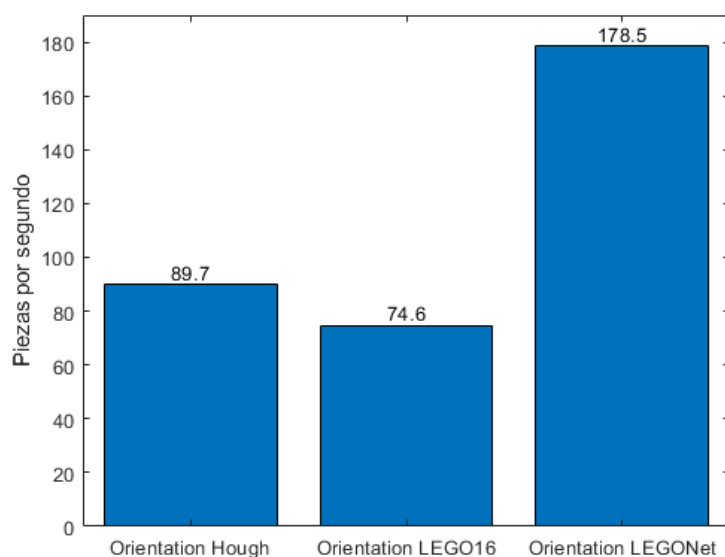


Figura 7.7. Diagrama de barras: Comparación de la velocidad de diferentes métodos para el cálculo de la orientación (más alto es mejor)

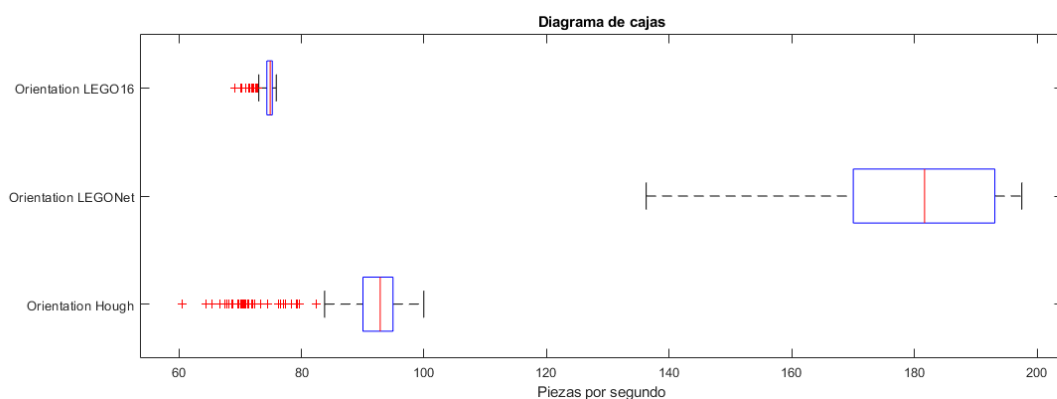


Figura 7.8. Diagrama de cajas: Comparación de la velocidad de diferentes métodos para el cálculo de la orientación (más a la derecha es mejor)

8

Conclusiones

El presente capítulo expone las conclusiones finales del proyecto extraídas a partir de los resultados obtenidos de las diferentes pruebas realizadas.

Durante la elaboración de este proyecto se ha perfeccionado un sistema compuesto por una cámara de color y profundidad, un brazo robótico y un ordenador con MATLAB. El sistema ha sido diseñado para identificar, detectar y recoger piezas de LEGO colocadas sobre la mesa de trabajo de un brazo IRB120.

El objetivo final de este proyecto es el perfeccionamiento del sistema previamente desarrollado por Ana Berjón Valles [Ber] mejorando la capacidad de reconocimiento y la conexión con el brazo robótico. Para ello se plantearon múltiples objetivos para cumplir a lo largo del desarrollo del proyecto, estos se pueden ver en la Sección 1.2. 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 subsidiar 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 se ha mitigado el problema de falsos positivos o piezas no detectadas. 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 es por ello que con solo modificar esta se puede usar el sistema con cualquier cámara. Además, el proceso de segmentación ya no depende tanto del calibrado de color de la cámara empleada. Para el procesado de la imagen de profundidad se ha creado un sistema fácil de calibrar y reutilizar para diferentes circunstancias. Se puede emplear para cualquier tipo de pieza y el único parámetro que se debe de cambiar es la altura de la pieza medida para determinar el número de piezas apiladas. 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.

9

Futuros desarrollos

En este capítulo se desarrollan las posibles mejoras que se pueden llevar a cabo partiendo del trabajo desarrollado en este proyecto. Se plantean posibles objetivos y posibles formas para llevarlos a cabo.

Este proyecto es la evolución de un proceso amplio y ambicioso, por lo que todavía hay numerosos puntos de mejora. El área de la robótica y la inteligencia artificial presenta infinidad de salidas, es por ello que en este apartado solo nos vamos a centrar en el sistema actual dentro de sus capacidades y limitaciones. Este capítulo se va a dividir en dos secciones, en primer lugar se va a desarrollar las posibles mejoras en el reconocimiento de imagen y en segundo lugar se hablará de mejoras en el agarre y colocación de piezas a través del brazo robótico.

Con este proyecto se ha mejorado notablemente el reconocimiento de imagen y se ha evolucionado el anterior sistema con la inclusión de nuevas tecnologías como las redes neuronales. Sin embargo, este sistema no es perfecto y puede ser ampliamente mejorado.

- Reentrenar las redes: debido a limitaciones por *hardware*, algunas redes no han podido mostrar su potencial o no han podido ser entrenadas. Por ello, es recomendable reentrenarlas con mayor potencia y a ser posible con una mayor base de datos.
- Piezas dobles: para mejorar la capacidad del sistema se puede plantear el uso de piezas de LEGO dobles. Para ello es necesario reentrenar las redes neuronales y obtener una nueva base de datos que incluya las piezas dobles. Además, se requiere de un sistema que sea capaz de distinguir una pieza doble de dos piezas simples del mismo color muy próximas. Para ello una posible opción es un análisis detallado de la pieza mediante la detección de borde. En caso de ser dos piezas simples debe de existir un borde en medio que las separe.
- Reflejos: el sistema ha mejorado frente a los efectos de los reflejos gracias al nuevo análisis de profundidad. Sin embargo, todavía se cree que se puede mejorar más. Para ello se pueden plantear varias alternativas. Se puede usar un filtro de polarización que ayude a reducirlos aunque en ese caso habría que analizar el impacto que tiene sobre los sensores de la cámara. O se puede plantear la alternativa de tomar múltiples fotos desde diferentes ángulos.

Durante el desarrollo de este proyecto se ha trabajado poco en el desarrollo de las capacidades del brazo robótico y su conexión con MATLAB. Es por ello por lo que este todavía puede ser ampliamente mejorado.

- **Automatización:** el sistema actual necesita que una persona indique cuando mandar la información al robot y como. Esto hace que el sistema no pueda ser independiente y sea más lento de lo necesario. Es por ello por lo que se aconseja rediseñar el sistema de comunicación entre MATLAB y el brazo robótico para que no sea necesaria la intervención humana. Esto implica la creación de una comunicación bidireccional.
- **Sistema anticolidión:** en el estado actual no se hace ningún análisis del proceso que debe de llevar a cabo el robot para coger una pieza. Esto implica que se puede dar el caso en el que este colisione con otra pieza. Por ello se recomienda la elaboración de un sistema que tenga en cuenta las limitaciones físicas del robot y la disposición de las piezas para poder maniobrar sin colisionar. Es necesario también tener en cuenta estas consideraciones a la hora de depositar las piezas ya que en caso de que haya más piezas azules que rojas es muy probable que la cámara colisione contra estas.
- **Variabilidad de las coordenadas:** el sistema de coordenadas presenta imperfecciones derivadas de la variabilidad de las coordenadas respecto al punto de referencia. Esta contrariedad se podría resolver de varias maneras, una de ellas es modificar el diseño de la pieza de manera que las tuercas quedaran encajadas impidiendo así su rotación. La otra posibilidad planteada, que no es incompatible con la anterior, sería realizar un proceso iterativo, que se podría combinar con el problema de la automatización. En este caso, la señal que debería enviar el robot contendría la posición del mismo en ese preciso instante, información obtenida con la instrucción CRobT y que permitiría que MATLAB calculara en ese momento la información de la siguiente pieza. Esto reduciría la incertidumbre del conjunto y lo haría más resistente a posibles imprevistos. Para que esto funcione es necesario primero poder establecer una comunicación bidireccional con el robot.
- **Piezas rotadas apiladas:** con el sistema actual se asume que si una pieza esta rotada no puede tener ninguna pieza más apilada. Para llevar esto a cabo es necesario modificar el sistema de alturas con el que trabaja el sistema actual de forma que pueda reconocer estas piezas y añadirles una altura de 3mm respecto a una pieza de la misma altura pero enganchada. Una vez reconocida la pieza se debería de recolocar en la hilera más cerca para de esta forma poder separar las piezas y recolocar las donde corresponda.

Por último, otro posible desarrollo del proyecto es la traspaso de este a otro lenguaje de programación. MATLAB es un lenguaje muy potente y con muchas ayudas que han sido empleadas para el desarrollo del proyecto. Sin embargo, no destaca por su rapidez y carece de un buen grado de control ya que no permite el control de los hilos de ejecución. Se recomienda el uso de C++ o Python ya que estos se caracterizan por ser más rápidos y dan un mayor control sobre la ejecución. Python destaca por ser uno de los lenguajes más empleados para el trabajo con redes neuronales y disponer de amplias librerías con las que poder trabajar.

Comunicación con la cámara

En este anexo se desarrollará la conexión con la cámara Intel realsense D435

Para el desarrollo de este proyecto se ha decidido emplear la cámara intel Realsense D435 [Intc] ya que dispone de todos los sensores necesarios, tiene buenas prestaciones y se ha desarrollado un wrapper para permitir la conexión directa a través de MATLAB.



Figura A.1. Cámara Intel Realsense D435 [Intc]

Para la conexión con la cámara se ha desarrollado una clase en MATLAB 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 ha sido creada como una herencia de la clase pipeline del wrapper de MATLAB. Pipeline se encarga de establecer la conexión entre la cámara y MATLAB y al crear nuestra clase como herencia de pipeline, nos permite tener acceso a todas las configuraciones y capacidades de pipeline y basarnos en estas para desarrollar una clase más simple e intuitiva de cara al usuario.

También se ha dado uso de la clase align del wrapper de MATLAB. Esta permite alinear la imagen de color y de profundidad de forma que ambas pasan a tener la misma resolución y

tienen la misma perspectiva. Este paso es clave para mejorar la precisión del sistema ya que es la única forma de saber con certeza donde se encuentran las piezas detectadas en la imagen de color en la imagen de profundidad. El proceso se muestra en la Figura A.2.

La clase se ha denominado como camera y cuenta con un total de seis funciones.

- Constructor: Se ha desarrollado un constructor personalizado que permiten 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 reiniciar MATLAB.
- status: Comprueba la conexión con la cámara para asegurarse de que esta sigue activa y preparada para ser usada.
- get_images: función para la captura de ambas imágenes de color y profundidad. Devuelve la imagen de color y una imagen de profundidad con una resolución de 640x480 píxeles. La imagen de profundidad ha sido alineada con la de color.
- get_colour: función para la captura solo de la imagen de color. Devuelve la imagen de color con una resolución de 640x480 píxeles
- get_depth: función para la captura solo de la imagen de profundidad. Devuelve la imagen de profundidad alineada con la de color y con una resolución de 640x480 píxeles.

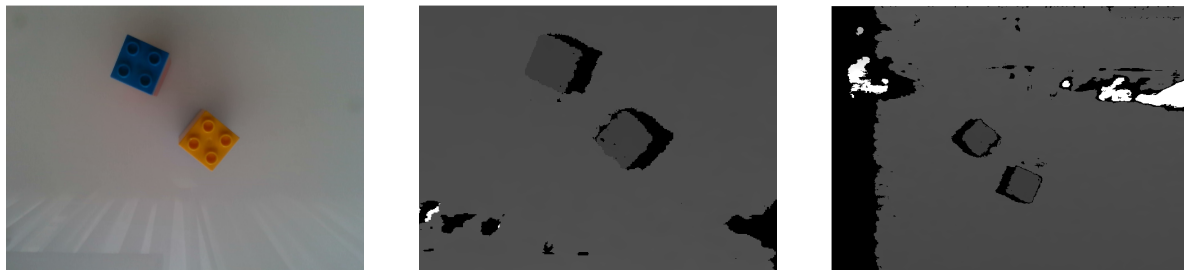


Figura A.2. Alineamiento de las imágenes de color y profundidad tomadas con la cámara Realsense D435

B

Interfaz gráfica de usuario

En este anexo se va a explicar el proceso de desarrollo de una interfaz gráfica de usuario desarrollada con MATLAB para el control del robot y las redes neuronales.

Para facilitar el control del sistema y mejorar la interacción con el robot, se ha desarrollado una interfaz gráfica que permita a un usuario sin conocimientos de MATLAB, de redes neuronales o del robot, ser capaz de controlarlo. Esta interfaz ha sido desarrollado con la ayuda de App Designer de MATLAB (ver Figura B.1). Se trata de una herramienta de MATLAB que facilita el desarrollo de interfaces gráficas de forma visual y sencilla, aunque también permite un mayor control a través de código.

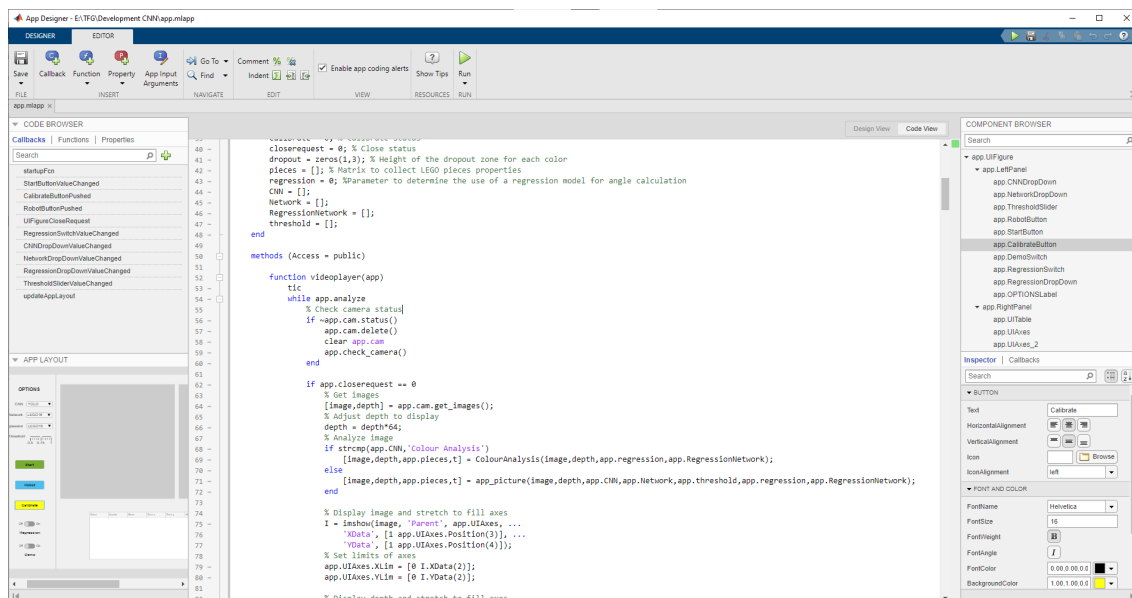


Figura B.1. Desarrollo de una interfaz gráfica con App Designer

B.1. Interfaz

La interfaz se ha decidido dividir en dos secciones para facilitar el manejo. La sección izquierda es la zona de control en la que el usuario puede determinar las opciones que desea

emplear. Y la sección de la derecha es la zona en la que el programa muestra toda la información obtenida y los resultados del proceso de segmentación.

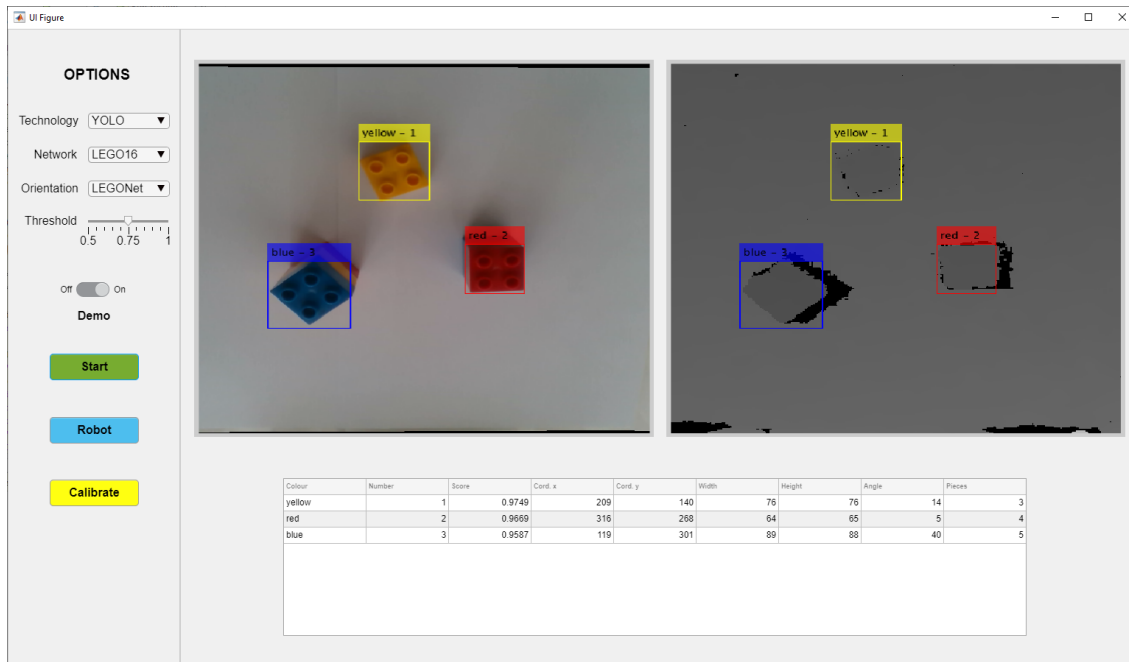


Figura B.2. Interfaz gráfica desarrollada con MATLAB

B.2. Controles

Para facilitar el control y manejo del sistema se ha decidido que la interfaz sea fácil de usar y de entender, es por ello que son pocos los controles y opciones. Las opciones principales se dividen en tres categorías:

- **Technology:** Menú desplegable con el que se permite elegir que tipo de tecnología se desea emplear para el proceso de segmentación. Existen cuatro opciones entre las que elegir: YOLO, R-CNN, Faster R-CNN y Colour Analysis. Esta última se trata de la segmentación con máscaras de color desarrollada en Capítulo 3.
- **Network:** Menú desplegable con el que se permite elegir qué tipo de red neuronales se desea emplear para el proceso de segmentación. En caso de haber escogido algún tipo de tecnología basada en redes neuronales, esta opción determina el tipo de red con la que se desea trabajar. Es decir, da la opción de escoger entre LEGONet y LEGO16.
- **Orientatoin:** Menú desplegable con el que se permite elegir qué tipo de tecnología se desea emplear para el proceso de cálculo de la rotación. Se permite escoger entre tres opciones, dos modelos de regresión basados en las redes neuronales LEGO16 y LEGONet y un sistema basado en la transformada de Hough.
- **Threshold:** Control deslizante para ajustar el valor umbral de confianza para las redes neuronales.
- **Demo:** Botón de activación para desconectar la conexión con el brazo robótico. Con el fin de poder simular el comportamiento del programa con el brazo robótico, se ha diseñado este botón que permite realizar todo a excepción de la conexión con el robot. De esta forma se puede ver el proceso del cálculo de coordenadas respecto al brazo robótico.

- Start: Arranca el programa con los ajustes seleccionados en Technology, Network, Orientation y Threshold. Analiza de forma continua las imágenes recibidas por la cámara y muestra el resultado a través de la interfaz gráfica.
- Robot: Manda las coordenadas obtenidas de la última imagen analiza al robot.
- Calibrate: Permite recalibrar el mapa de profundidad para determinar la altura de las piezas.

B.3. Medidas de seguridad

Con el fin de ofrecer la mejor experiencia de usuario posible, se ha desarrollado varios sistemas de seguridad que comprueban constantemente el correcto funcionamiento del programa y la correcta conexión entre los diferentes dispositivos. Estas medidas de seguridad se pueden dividir en dos categorías:

- Comprobación de conexiones: El programa ha sido diseñado para asegurar de que en caso de que alguna conexión falle con el brazo robótico o la cámara, este sea capaz de recuperarse y continuar trabajando. Para ello antes de comunicarse con cualquiera de estos dispositivos primero comprueba su estado. El proceso es bastante rápido por lo que no perjudica mucho el rendimiento del programa.
- Comunicación entre procesos: El sistema es capaz de realizar varios procesos de forma simultanea, pero esto puede suponer un problema ya que se pueden quedar procesos en cola que no puedan llegar a correrse nunca o que para cuando se intente correr el sistema ya no este preparado para correrlos. Es por eso por lo que se ha diseñado un sistema de control que evita que los procesos se almacenen o interrumpan entre si cuando no es debido,

B.4. Problemas causados por la interfaz

Una de las mayores carencias de MATLAB como lenguaje de programación es la falta de control sobre los hilos de ejecución. Durante el desarrollo de este proyecto esto no ha supuesto ningún problema ya que todo el proceso es bastante lineal y directo. Sin embargo, al desarrollar la interfaz se ha visto como el rendimiento del sistema ha reducido bastante ya que no puede separar la carga por hilos y hay múltiples sistemas que interrumpen constantemente la ejecución principal para hacer comprobaciones o actualizar valores. Si se trabaja con otro lenguaje de programación es muy probable que este problema pueda ser solventado.



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 C.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 por 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 desmostado 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.

Bibliografía

- [ABBa] ABB. «IRB 120», dirección: <https://new.abb.com/products/robotics/es/robots-industriales/irb-120> (visitado 18-06-2020).
- [ABBb] —, «RobotStudio 2019.5», dirección: <https://new.abb.com/products/robotics/robotstudio/downloads> (visitado 22-06-2020).
- [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).
- [Aga18] A. F. Agarap, «Deep Learning using Rectified Linear Units (ReLU)», *CoRR*, vol. abs/1803.08375, 2018. arXiv: 1803.08375. dirección: <http://arxiv.org/abs/1803.08375>.
- [Alo+18] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, B. C. V. Esesn, A. A. S. Awwal y V. K. Asari, *The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches*, 2018. arXiv: 1803.01164 [cs.CV].
- [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.
- [ART15] L. Apvrille, Y. Roudier y T. Tanzi, «Autonomous Drones for Disasters Management: Safety and Security Verifications», mayo de 2015.
- [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.
- [Ber] A. Berjón Valles, «Integración de un Sistema de Visión Artificial en la Mano de un Robot Industrial», Trabajo de Final de Grado, ICAI,
- [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, R. Pons, Xiaolong Feng, R. Patel, D. Wäppling, J. Weström y H. Andersson, «Towards high performance robotic solutions in press automation - An ABB view», en *IEEE ISR 2013*, 2013, págs. 1-3.
- [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>.
- [Gar] F. Garcia. «Lego Brick Sorting - Image Recognition», dirección: <https://www.kaggle.com/pacogarciam3/lego-brick-sorting-image-recognition> (visitado 29-06-2020).
- [Gir+13] R. Girshick, J. Donahue, T. Darrell y J. Malik, *Rich feature hierarchies for accurate object detection and semantic segmentation*, 2013. arXiv: 1311.2524 [cs.CV].
- [Gir15] R. Girshick, *Fast R-CNN*, 2015. arXiv: 1504.08083 [cs.CV].
- [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.

- [Haz] J. Hazelzet. «Images of LEGO bricks», dirección: <https://www.kaggle.com/joosthazelzet/lego-brick-images> (visitado 29-06-2020).
- [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. «Intel image classification», dirección: <https://www.kaggle.com/puneet6060/intel-image-classification/version/2> (visitado 29-06-2020).
- [Intb] —, «MATLAB wrapper for Intel Realsense», dirección: <https://github.com/IntelRealSense/librealsense/tree/master/wrappers/matlab> (visitado 22-06-2020).
- [Intc] —, «Realsense D435», dirección: <https://www.intelrealsense.com/depth-camera-d435/> (visitado 18-06-2020).
- [Kat19] A. Kattepur, «Workflow composition and analysis in Industry 4.0 warehouse automation», *IET Collaborative Intelligent Manufacturing*, vol. 1, n.º 3, págs. 78-89, 2019.
- [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)». English, dirección: <http://www.hitachi.com/corporate/about/history/1961.html>.
- [Lec+89] Y. Lecun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard y L. Jackel, «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, A. W. Tow, M. McTaggart, R. Smith, N. Kelly-Boxall, S. Wade-McCue, J. Erskine, R. Grinover, A. Gurman, T. Hunn, D. Lee, A. Milan, T. Pham, G. Rallos, A. Razjigaev, T. Rowntree, K. Vijay, Z. Zhuang, C. Lehnert, I. Reid, P. Corke y J. Leitner, *Cartman: The low-cost Cartesian Manipulator that won the Amazon Robotics Challenge*, 2017. arXiv: [1709.06283](https://arxiv.org/abs/1709.06283) [cs.R0].
- [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. arXiv: [1506.02640](https://arxiv.org/abs/1506.02640) [cs.CV].
- [Ren+15] S. Ren, K. He, R. Girshick y J. Sun, *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*, 2015. arXiv: [1506.01497](https://arxiv.org/abs/1506.01497) [cs.CV].
- [SZ14] K. Simonyan y A. Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition*, 2014. arXiv: [1409.1556](https://arxiv.org/abs/1409.1556) [cs.CV].
- [Sze11] R. Szeliski. (2011). «Computer vision algorithms and applications», dirección: <http://dx.doi.org/10.1007/978-1-84882-935-0>.
- [Tsa] S. H. Tsang. «Review R-CNN (Object Detection)», dirección: <https://medium.com/coinmonks/review-r-cnn-object-detection-b476aba290d1> (visitado 02-07-2020).
- [Vid19] A. Vidhya. (2019). «Brief History of Neural Networks», dirección: <https://medium.com/analytics-vidhya/brief-history-of-neural-networks-44c2bf72eec>.

- [WMZ12] C. Wang, W. Miao y J. Zhao, «Vision-Based Curvature Model for Artificial Intelligence in Vehicles», en *2012 International Conference on Control Engineering and Communication Technology*, 2012, págs. 245-248.
- [YC05] Yanhui Guo y Cong Wang, «Autonomous decentralized network security system», en *Proceedings. 2005 IEEE Networking, Sensing and Control, 2005.*, 2005, págs. 279-282.

