COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

MEJORA DEL SISTEMA DE SEGURIDAD DE UN ROBOT INDUSTRIAL BASADO EN VISIÓN ARTIFICIAL

Autor: Álvaro Fernández Blázquez

Director: Jaime Boal Martín-Larrauri

Director: José Antonio Rodríguez Mondéjar

Septiembre 2020

Madrid

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

Mejora del sistema de seguridad de un robot industrial basado en visión artificial

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2019/20 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.: Álvaro Fernández Blázquez Fecha: ……/ ……/ ……

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Firmado digitalmente por BOAL MARTIN LARRAURI JAIME - 05304600H
Fecha: 2020.09.25 11:02:15 +02'00'

Fdo.: Jaime Boal Martín-Larrauri Fecha: ……/ ……/ ……

EL DIRECTOR DEL PROYECTO

Firmado por mondejar@comillas.edu el día 27/09/2020 con un certificado emitido por Communications Server

Fdo.: José Antonio Rodríguez Mondéjar Fecha: ……/ ……/ ……

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

MEJORA DEL SISTEMA DE SEGURIDAD DE UN ROBOT INDUSTRIAL BASADO EN VISIÓN ARTIFICIAL

Autor: Álvaro Fernández Blázquez

Director: Jaime Boal Martín-Larrauri

Director: José Antonio Rodríguez Mondéjar

Septiembre 2020

Madrid

Agradecimientos

En primer lugar, quiero agradecer a D. Jaime Boal, como director de proyecto por haberme ofrecido la realización de este proyecto fin de grado, por su ayuda y dedicación a las numerosas dudas y problemas que han surgido a lo largo de este proyecto. Asimismo, agradecerle a D. Aurelio García por su interés en este proyecto y su disponibilidad.

También me gustaría dedicar este agradecimiento a mis padres por su esfuerzo y apoyo durante estos años. A toda mi familia por ayudarme siempre en los tiempos difíciles durante estos cuatro años del grado, y en especial en este último periodo. Por último, a mis compañeros de Comillas ICAI con los que siempre he contado durante mis estudios, entre ellos, Javier Colinas por su disponibilidad cuando he tenido alguna duda.

MEJORA DEL SISTEMA DE SEGURIDAD DE UN ROBOT INDUSTRIAL BASADO EN VISIÓN ARTIFICIAL

Autor: Fernández Blázquez, Álvaro.
Director: Boal Martín-Larrauri, Jaime.
Director: Rodríguez Mondéjar, José Antonio.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

Este proyecto mejora el sistema de seguridad por control de velocidad y separación de un robot industrial IRB 120, del laboratorio de automatización de Comillas ICAI. La mejora permite incrementar la velocidad de procesamiento de imágenes necesaria para el control del robot utilizando el lenguaje de programación Python en lugar de Matlab, permitiendo que, en un entorno colaborativo, el robot reduzca la velocidad o, si es necesario se pare a tiempo evitando colisiones, aunque una persona u objeto se acerque muy rápido. Además, se trata de un código abierto, para el sistema de seguridad, no siendo necesaria licencia de uso, lo que reduce sus costes.

Palabras clave: Visión artificial, Python, Seguridad industrial, Brazo robótico, Mixtura de Gaussianas, OpenCV.

1. Introducción

En los últimos años los robots industriales han sido mejorados tanto en rapidez como en precisión y disminuido su precio por lo que es más viable su implementación en procesos productivos. Sin embargo, estos robots debían instalarse en zonas de acceso restringido puesto que no estaban diseñados para una interacción segura y, por tanto, era necesario que el robot detuviera su actividad si alguna persona se acercaba a este. El siguiente paso en la actual cuarta revolución industrial, es eliminar la limitación de interactuación hombre-máquina e impulsar la colaboración de forma segura [1].

Estos robots colaborativos, también llamados “Collaborative Robots - Cobots” pueden aprender tareas y movimientos o incluso decidir, según en la situación en la que estén (como paralizar su funcionamiento al instante ante una situación de posible riesgo para las personas). Por tanto, reducir los tiempos necesarios para el procesamiento de datos y la respuesta de los robots a los mismos es de suma importancia para que el entorno de trabajo colaborativo de robots con personas u objetos sea seguro.

Resolver y mejorar la seguridad de este tipo de entornos es especialmente interesante porque cada vez son más las industrias que apuestan por la automatización mediante robótica colaborativa y se espera que el mercado se multiplique por veinte en 2025 [2].

2. Definición del proyecto

Los sensores son esenciales para que los robots sean capaces de reconocer el ambiente en el que se encuentran y lleven a cabo las tareas/movimientos que se les requieran. Además de los sensores, se precisa de una tecnología adecuada de procesamiento de imágenes, así como un lenguaje que trate estas imágenes y las traduzca al lenguaje comprensible por el robot. La elección de uno u otro elemento de los tres mencionados

puede afectar al resultado. Este proyecto se centra en la reducción del tiempo del procesamiento de imágenes y la implementación del software en la industria, utilizándose Phyton, uno de los lenguajes más conocidos y fiables del mercado. Esta reducción de tiempos se compara con otro lenguaje Matlab que fue el utilizado en el anterior proyecto de Concepción Góngora [3].

3. Descripción del sistema

Para cumplir con los objetivos el sistema recoge las imágenes en directo de la cámara D435 del robot IRB 120 del laboratorio y las analiza mientras el brazo robótico y los objetos entran en el área de trabajo. El sistema de seguridad se conecta a la cámara, con un lenguaje operativo Python y enviarle los resultados a Rapid, lenguaje informático que controla el brazo robótico IRB 120 como se indica en la Figura 1.

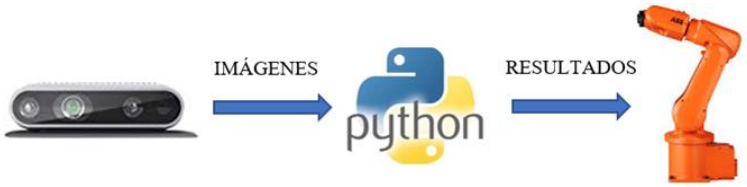

Figura 1. Diagrama de los bloques que componen el sistema desarrollado [4][5][6]

Debido a causas de fuerza mayor el acceso al robot en el periodo de realización del proyecto no ha sido posible. Como opción se ha grabado un vídeo con la misma cámara D435 y con el brazo robótico en funcionamiento, mientras algunos objetos entran y salen del área de trabajo. Las mayores diferencias son que el tratamiento de la imagen no es en vivo y no se envían los resultados desde el lenguaje de programación que ejecuta el sistema de seguridad al Rapid, por lo que el diagrama de flujo es solo lectura de *frames* de un vídeo grabado anteriormente y procesamiento de las imágenes en Python, siendo los resultados en este último lenguaje.
Los métodos utilizados para detectar los objetos son MOG2 y SSIM, para detectar el brazo robótico se segmenta por color. Para limpiar las imágenes de ruido se utilizan transformaciones morfológicas y para poder calcular las distancias se utilizan las funciones que detectan los contornos de los objetos. Estas funciones las utiliza tanto Python como Matlab y ambos lenguajes de programación las utilizan de la librería OpenCV.

4. Resultados

Una comparación entre Matlab y Python con respecto al tiempo de ejecución total de una imagen, representado en un diagrama de caja y bigotes, se observa en la Figura 2. La mediana de tiempo de Matlab es de 42,09 ms mientras que la mediana de tiempo de Python es de 15,01 ms.

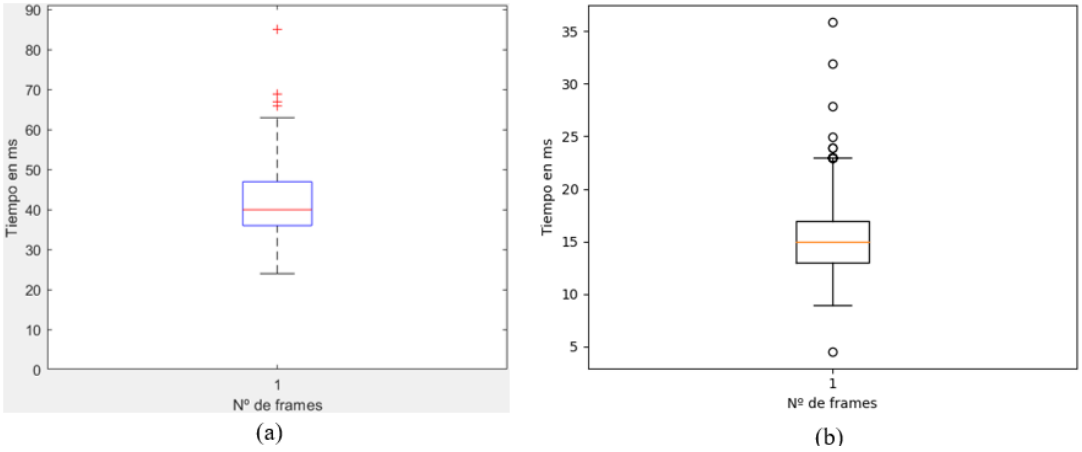

Figura 2. Tiempo de procesamiento de una imagen en Matlab y en Python

Los resultados de los tiempos del sistema de seguridad ejecutado en Python frente a Matlab, tras probarlos en diferentes vídeos con distintas condiciones de iluminación, se representan en la Tabla 1.

	Matlab Tiempo (ms)	Python. Tiempo (ms)	Mejora de Python (%)
Total	42,09	14,85	64,72
Segmentación por color	7,66	0,18	97,65
Sustracción de fondo	21,30	12,07	43,33
Eliminación del brazo	0,83	0,13	84,34
Transformación morfológica	14,68	1,52	89,65
Contornos / Creación de blobs	1,26	1,84	-46,03
Cálculo de distancias y peligro	0,20	0,03	85,00

Tabla 1. Comparación de tiempos entre Matlab y Python

Python supera a Matlab en todos los procesos menos en el contorno de los objetos, aunque, a su vez, es el tiempo que más se asemeja en Matlab y Python. La diferencia de tiempo es de 27,24 ms lo que indica, si bien Matlab podía procesar imágenes a 23fps, Python puede procesar las mismas imágenes con las mismas respuestas, pero a 67 fps.

5. Conclusiones

Las conclusiones que se pueden extraer de este trabajo fin de grado son que, Python es más rápido que Matlab y ofrece los mismos resultados por lo que su uso se prefiere antes que Matlab. Como es más rápido es más seguro, puede procesar y enviar los mensajes al brazo robótico antes de que se produzca una colisión con un objeto o persona. Esto abarata los costes a la hora de implementar este sistema de seguridad a los robots de la industria ya que Python no requiere comprar una licencia para utilizarlo. Se puede mejorar y hacer más complejos los procesos para reconocer los objetos siempre y cuando no se superen los 16 ms en tiempo total de procesamiento de imagen.

6. Referencias

[1] IGAPE. Estado del Arte de Automatización y Robótica. Oportunidades Industria 4.0 en Galicia. http://www.igape.es/es/ser-mas-competitivo/galiciaindustria4-0/estudos-e-informes/item/1529-oportunidades-industria-4-0-en-galicia . "Última visita 30/08/2020".

[2] Cade cobots. Cobots: los nuevos robots con inteligencia artificial. https://cadecobots.com/cobots-robots-con-inteligencia-artificial/. "Última visita: 21/01/2020"

[3] Concepción Góngora Luque. Aplicación de la visión artificial a la seguridad de un robot. Universidad Pontificia de Comillas ICAI, Madrid, Julio 2019.

[4] Camera D435. Intel Realsense. https://www.intelrealsense.com/depth-camera-d435/ "Última visita: 22/01/2020"

[5] LinuxAdictos. Python: los lenguajes también pueden ser de código abierto. https://www.linuxadictos.com/python-los-lenguajes-de-codigo-abierto.html. "Última visita 01/09/2020"

[6] Medusa Robotics. Robot arm irb120 controlled by ROS. https://sites.google.com/site/medusarobotics/abb_irb120. "Última visita 01/09/2020"

IMPROVEMENT OF THE SECURITY SYSTEM OF AN INDUSTRIAL ROBOT BASED ON COMPUTER VISION

Author: Fernández Blázquez, Álvaro.
Supervisor: Boal Martín-Larrauri, Jaime.
Supervisor: Rodríguez Mondéjar, José Antonio.
Collaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

This project improves the safety system by speed control and separation of an industrial robot IRB 120 from the automation laboratory of Comillas ICAI. The improvement will increase the image processing speed required for robot control using the Python programming language instead of Matlab. This will allow the robot to slow down in a collaborative environment or, if necessary, stop in time avoiding collisions, even if a person or object approaches very quickly. In addition, it is an open source for the security system, not requiring a user's license, which reduces its costs.

Key words: Artificial vision, Python, Industrial safety, Robotic arm, Gaussian Mixture, OpenCV.

1. Introduction

In recent years, industrial robots have been improved in both speed and precision and reduced their price, making their implementation in production processes more viable. However, these robots had to be installed in restricted access areas since they were not designed for safe interaction and therefore it was necessary for the robot to stop its activity if any person approached it. The next step in the current fourth industrial revolution is to eliminate the limitation of man-machine interaction and to promote collaboration in a safe manner [1].

These collaborative robots, also called "Collaborative Robots – Cobots" can learn tasks and movements or even decide, depending on the situation they are in (how to stop their functioning instantly in the face of a situation of possible risk to people).Thus, reducing the time required for data processing and the response of robots to them is of paramount importance for the collaborative working environment of robots with people or objects to be safe.

Solving and improving the safety of this type of environment is particularly interesting because more and more industries are boosting automation through collaborative robotics and the market is expected to multiply by twenty in 2025 [2].

2. Project definition

Sensors are essential for robots to be able to recognize the environment in which they are located and carry out the required tasks/movements. In addition to sensors, appropriate image technology is needed, as well as language that manages these images and translates them into a comprehensive language for the robot. The choice of one or the other element of the three mentioned above may affect the outcome. This project

focuses on reducing the time needed for image processing using Phyton and facilitating software implementation in the industry. This is one of the best-known and reliable languages in the market. This time reduction is compared to another Matlab language that was used in Concepción Gongora's previous project [3].

3. System description

To meet the objectives, the system collects live images from the D435 camera of the laboratory's IRB 120 robot and analyses them as the robotic arm and objects enter the work area. The security system connects to the camera with a Python operating language and sends the results to Rapid, a computer language that controls the robotic arm IRB 120 as shown in Figure 1.

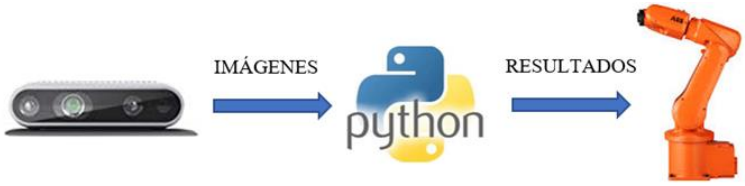

Figure 1.Diagram of the blocks that make up the developed system [4][5][6]

Due to force majeure reasons, access to the robot during the project period has not been possible. As an option, a video has been recorded with the same D435 camera and with the robotic arm in operation, while some objects enter and leave the work area. The biggest differences are that the treatment of the image is not alive and the results are not sent from the programming language that runs the security system to Rapid, so the flowchart is only reading frames from a previously recorded video and processing the images in Python, being the results in the latter language.

The methods applied to detect objects are MOG2 and SSIM while to detect the robotic arm is color segmentation. Morphological transformations are used to clean the noise images and to calculate the distances, the functions that detect the contours of the objects are used. These functions are used by both Python and Matlab and both programming languages use them from the library OpenCV.

4. Results

A comparison between Matlab and Python with respect to the total time execution of an image, represented in a box plot, is shown in Figure 2. The median time of Matlab is 42.09 ms while the median time of Python is 15.01 ms.

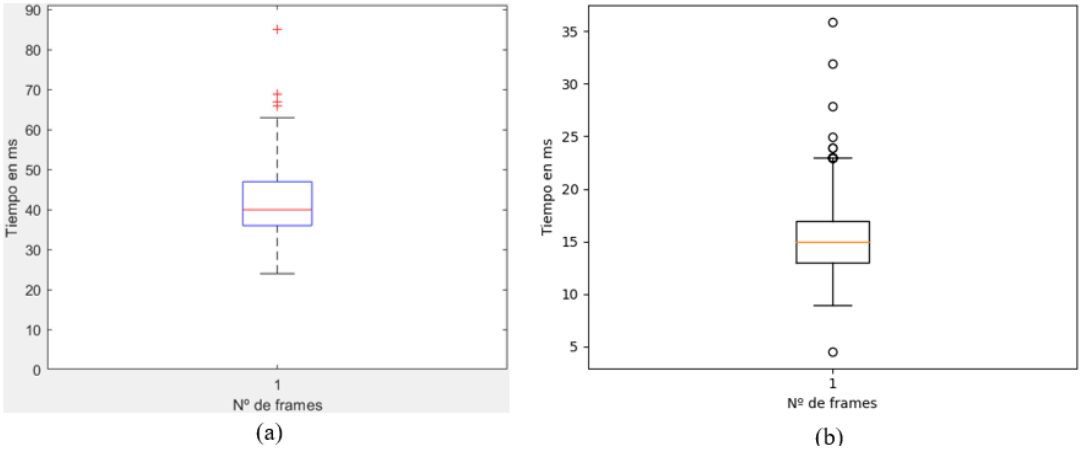

Figura 2. Image processing time in Matlab and Python

The results of times for the security system executed in Python compared with Matlab, after testing them in different videos with different lighting conditions, are shown in Table 1.

	Matlab Time (ms)	Python. Time (ms)	Python improvement(%)
Total	42,09	14,85	64,72
Color segmentation	7,66	0,18	97,65
Background sustraction	21,30	12,07	43,33
Arm removal	0,83	0,13	84,34
Morphological transformation	14,68	1,52	89,65
Blobs creation/contours	1,26	1,84	-46,03
Distances and danger calculation	0,20	0,03	85,00

Table 1. Time comparison between Matlab and Python

Python surpasses Matlab in all processes except in the contour of objects, although, in turn, it is the time that most resembles Matlab and Python. The time difference is 27.24 ms which indicates, although Matlab could process images at 23fps, Python can process the same images with the same answers, but at 67 fps.

5. Conclusions

The conclusions that can be drawn from this final degree project are that, Python is faster than Matlab showing the same results so its use is preferable to Matlab. Python is safer, with a faster processing in sending messages to the robotic arm before collision with an object or person. It also reduces implementation costs of this safety system in industrial robots since Python does not require purchasing a license to use it. You can improve and make more complex processes to recognize objects as long as you don't exceed 16 ms in total image processing time.

6. References

[1] IGAPE. Estado del Arte de Automatización y Robótica. Oportunidades Industria 4.0 en Galicia. http://www.igape.es/es/ser-mas-competitivo/galiciaindustria4-0/estudos-e-informes/item/1529-oportunidades-industria-4-0-en-galicia. "Última visita 30/08/2020".

[2] Cade cobots. https://cadecobots.com/cobots-robots-con-inteligencia-artificial/. "Última visita: 21/01/2020"

[3] Concepción Góngora Luque. Aplicación de la visión artificial a la seguridad de un robot. Universidad Pontificia de Comillas ICAI, Madrid, Julio 2019.

[4] Camera D435. Intel Realsense. https://www.intelrealsense.com/depth-camera-d435 / "Última visita: 22/01/2020"

[5] LinuxAdictos. https://www.linuxadictos.com/python-los-lenguajes-de-codigo-abierto.html "Última visita 01/09/2020"

[6] Medusa Robotics. https://sites.google.com/site/medusarobotics/abb_irb120 "Última visita 01/09/2020"

Índice de la memoria

Índice de figuras

Índice de tablas

Capítulo 1. INTRODUCCIÓN

Desde que empezó la primera revolución industrial por el año 1789, la mayoría de los procesos de fábrica son repetitivos y siempre se ha buscado la optimización de tiempo y costes a la hora de fabricar cualquier producto. En concreto, en la Figura 1 se muestra que todas las revoluciones industriales han ido de la mano de procesos de automatización, así como también, que estas revoluciones se producen cada vez en un menor periodo de tiempo.

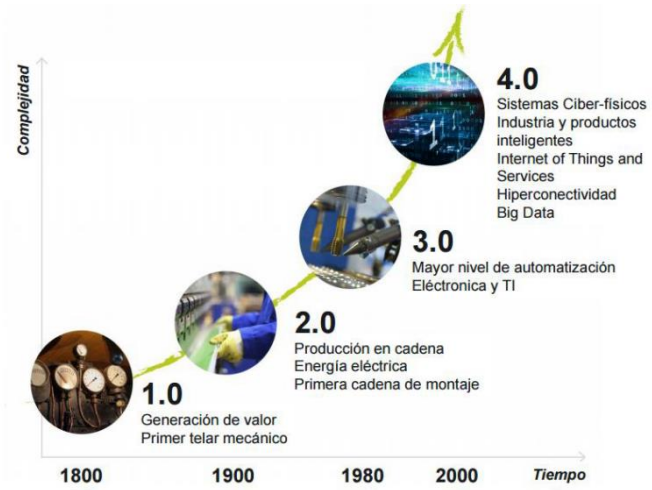

Figura 1. Evolución de las tecnologías industriales en cada una de las revoluciones industriales en el tiempo [1]

Desde que se iniciaron los procesos industriales siempre se usó principalmente mano de obra "humana" pero el paradigma cambió en el año 1959 cuando General Motors decide instalar el primer robot industrial, creado por la empresa Unimation, en su fábrica de Trenton [2][1] y seguidos por FIAT en Europa. Este primer robot industrial consistía en un brazo robótico de 4000 libras que, en una línea de montaje permitía transportar piezas de fundición y soldarlas en partes de los automóviles, lo que suponía una tarea peligrosa para los trabajadores por los gases liberados durante el proceso o por la posible pérdida de un miembro en el menor descuido [3].

Debido a la mejora en tiempos y resultados de las cadenas de producción muchas otras empresas decidieron adaptar sus fábricas incorporando robots industriales, creciendo exponencialmente por toda la industria hasta día de hoy. Un brazo robótico es, según la Organización Internacional de Normalización (ISO), un manipulador programable en tres o más ejes con varios propósitos, controlado automáticamente y reprogramable [4].

Para el año 2020 ya se estima que se instalarán 1,7 millones robots industriales [5] y ya, en España, había 160 robots por cada 10.000 trabajadores [6]. Durante los últimos años los robots industriales han sido mejorados tanto en rapidez como en precisión y disminuido su precio por lo que es más viable su implementación en procesos productivos. Sin embargo, estos robots debían instalarse en zonas de acceso restringido puesto que no estaban diseñados para una interacción segura y, por tanto, era necesario que el robot detuviera su actividad si alguna persona se acercaba a este. El siguiente paso, que fomenta la competitividad de la industria, y siempre se busca el proceso más eficaz, es desarrollar procesos en los cuales un operario pueda trabajar al mismo tiempo y en el mismo espacio con un robot industrial, a estos robots se les llama robots colaborativos. Por tanto, en la actual cuarta revolución industrial, esta limitación de interactuación hombre-máquina no sólo desaparece, sino que se impulsa la colaboración de forma segura [1]. Con este proyecto se podría aumentar de grado de colaboración del Robot IRB 120 (Figura 2) propiedad de Comillas ICAI mejorando el sistema de seguridad basado en la visión artificial.

Figura 2. Robot IRB 120 [7]

El robot con el que se trabaja en el proyecto es el IRB 120 [8], creado por la empresa ABB en 2009. Un robot compacto, ligero y que se utiliza en la industria, perfecto para realizar el proyecto. Posteriormente la empresa ABB lanzo una versión mejorada, IRB 120T en 2012 [9].

Para su correcto funcionamiento, los robots colaborativos disponen de sensores que recogen muchos datos del entorno en tiempo real, como cámaras o sensores de contacto, entre otros. Todos estos datos e información son procesados por los robots mediante códigos que pueden interpretar el área de trabajo como lo haría una persona. Estos **robots colaborativos**, también llamados "Collaborative Robots - Cobots" pueden aprender tareas y movimientos o incluso decidir según en la situación en la que estén (como paralizar su funcionamiento al instante ante una situación de posible riesgo para las personas). En este proyecto se pretende mejorar el grado de colaboración optimizando en tiempo el procesamiento de datos y la respuesta del robot a los mismos.

En conclusión, los últimos avances tecnológicos han permitido evolucionar los robots industriales, reduciendo los costes de automatización y proporcionándoles nuevas habilidades de percepción, movimiento y aprendizaje. Esto justifica el interés de este trabajo porque cada vez son más las industrias que apuestan por la automatización mediante robótica colaborativa y ya no solo se introducen en grandes compañías: también está aumentando el crecimiento de la robótica colaborativa en PYMES y se espera que el mercado se multiplique por veinte en 2025 [10].

1.1 MOTIVACIÓN DEL PROYECTO

La motivación de llevar a cabo este proyecto se debe a la necesidad de aportar nuevas soluciones al mundo de las empresas, en el campo de la seguridad de las personas en sus puestos de trabajo y para avanzar hacia la meta de industrias modernas y competitivas, con procesos más automatizados y seguros lo que se puede denominar Industria 4.0 o industrias inteligentes.

El desarrollo de proyectos en el campo de la robótica colaborativa, como este, permitirá un mejor conocimiento de las comunicaciones del ser humano con los robots, incrementando el uso de estas soluciones en otros ámbitos. Pudiendo crear una carrera profesional con puestos de trabajo con mucho futuro.

Por último, es una satisfacción que el proyecto pueda aportar mejoras en las condiciones de trabajo e incrementar la productividad de las cadenas de producción de varios sectores industriales.

1.2 OBJETIVOS

El principal objetivo del proyecto fue la mejora del sistema de seguridad por control de velocidad y separación de un robot industrial (IRB 120), instalado en el laboratorio de automatización de Comillas ICAI. El sistema de seguridad estuvo basado en la visión artificial y la mejora de este sistema consistió en incrementar la velocidad de procesamiento de imágenes utilizando el lenguaje Python en lugar de Matlab. Esta mejora consiguió que, aunque la persona o el objeto se acerque muy rápido, el robot reduzca la velocidad o, si es necesario, se detuviera a tiempo para evitar colisionar con un objeto.

Otro de los objetivos del proyecto era reducir los costes, ya que las empresas tienen que pagar por las licencias de las aplicaciones y su mantenimiento para poder implementar este sistema a sus robots industriales en las plantas de producción. Por este motivo se utilizó un código que no precisa de licencia, lo que reduce costes.

1.3 RECURSOS

Para el desarrollo de este proyecto se han utilizado principalmente los siguientes recursos:

- ***IRB 120, robot industrial ABB***

ABB es una empresa líder mundial en ingeniería eléctrica y automatización. La compañía es el producto de la unión en 1988 de Asea y Brown Boveri. El robot IRB 120, adquirido por

Comillas ICAI, es el más ligero y de menores dimensiones, se puede utilizar para tareas que requieran como máximo 3 kg y pesa solamente 25 kg. Tiene un alcance horizontal de 580 mm y de profundidad desde su base de 112 mm. Tiene una relación coste-efectividad con la que es capaz de obtener una alta tasa de producción con una baja inversión.

- ***Intel RealSense Depth Camera D435***

La cámara D435, escogida para el presente trabajo, utiliza visión estéreo para la captación de imágenes de profundidad. Es alimentada por USB el cuál va conectado al ordenador y consiste en un par de sensores de profundidad, sensor RGB y proyector infrarrojo. Puede obtener fotografías con una resolución de 1280x720. Además, tiene incluido un procesador de señal para cambiar los ajustes de la imagen y escala de datos de color. El proyector de infrarrojos lo utiliza para iluminar objetos y mejorar los datos obtenidos en la imagen de profundidad.

- ***Matlab***

Sistema de cómputo numérico que ofrece un entorno de desarrollo integrado con un lenguaje de programación propio. El código está programado para este sistema y las mejoras iniciales investigarán optimizando las funciones de Matlab.

- ***Python***

Es un lenguaje de programación con alta facilidad de aprendizaje, alta legibilidad, robusto y potente. Este lenguaje es de código abierto teniendo librerías de libre acceso sin licencia y con bastantes librerías creadas por los usuarios de este lenguaje lo cuál facilita la creación de cualquier código.

Capítulo 2. ESTADO DE LA CUESTIÓN

2.1 SEGURIDAD EN LA COLABORACIÓN HUMANO-ROBOT

El origen de la palabra "Robot" viene de la palabra checa "robota" creada a principio del siglo 20 y se crean con el objetivo de sustituir a los humanos en tareas duras y rutinarias. Hacia principios de 1940 se hizo necesario estables algunos principios para definir mejor a los robots. Fue Isaac Asimov el que presentó su "Tres Principios para la Ingeniería de Robots" [11] :

1) Los robots no deben herir a los humanos o ignorar a los humanos que hayan resultado heridos.

2) Los robots deben obedecer las órdenes dadas por los humanos, siempre y cuando éstas no vayan en contra del artículo 1.

3) Los robots deben protegerse a sí mismos, siempre y cuando esto no vaya en contra de los artículos 1 y 2.

Las tecnologías aplicadas a la fabricación de robots han ido evolucionando de forma muy rápida a partir de la segunda guerra mundial en sus distintas disciplinas tanto de *hardware*, software, comunicaciones, computación etc. Hoy en día, no podrían existir robots sin el desarrollo de sistemas de computación y tecnología de control. En la década de los 60, fueron fabricados por primera vez los circuitos integrados. Con el desarrollo de circuitos integrados, la velocidad de cálculo de los robots mejoró enormemente, junto con grandes avances en la miniaturización. Estos avances ayudaron a impulsar el desarrollo de los robots y más tarde se plantearon la necesidad, relacionada con la tecnología de los robots, de la búsqueda de dispositivos equipados con funciones equivalentes a los cinco sentidos humanos, es decir, "sensores" [12].

Los robots colaborativos, o cobots, han sido diseñados para trabajar junto a humanos, ayudando a estos en las tareas que resultan más pesadas y de mayor precisión. Hoy en día dadas sus prestaciones, bajos costes y su integración en entornos de trabajos diversos, se están introduciendo de forma rápida no solo en las empresas con cadenas de producción más clásicas, sino también para muchas otras actividades relacionas con los servicios de salud, medioambientales, transporte dentro de plantas industriales etc. Las investigaciones no solo se dirigen a un mejor funcionamiento de los robots, sino principalmente en darles mayores funcionalidades como conducción autónoma, sistemas colaborativos, reconocimiento de objetos y comunicación entre máquinas, entre otras.

En la actualidad el sector de la fabricación de los cobots está dominado por empresas de los países más desarrollados destacando las compañías OMRON, Kuka ABB, FANUC y Universal Robots [13].

La colaboración humano robot a través de los cobots ha supuesto un avance muy importante en los últimos años y según los expertos las previsiones de crecimiento son muy altas. Esto se debe entre otras razones a:

- Pueden hacer tareas repetitivas y pesadas de una forma segura interactuando con humanos.
- Se puede adaptar a varias tareas o procesos de producción de forma rápida y a un bajo coste.
- Para muchas tareas los cobots pueden ser programados de forma sencilla.

La seguridad humano-robot es siempre una prioridad y para eso existen a nivel internacional y nacional una serie de normas de los organismos de normalización que deben cumplir los cobots para considerarlos seguros. La Directiva de Máquinas 2006/42/CE [14] del Parlamento Europeo y del Consejo, es el marco general. A partir de ella se han elaborado distintas normas para la homologación de robots en la industria.

Las principales normas sobre este tema son:

- ISO 10218-1 Robots y dispositivos robóticos. Requisitos de seguridad para robots industriales. Parte 1: Robots [15].
- ISO 10218-2. Robots y dispositivos robóticos. Requisitos de seguridad para robots industriales. Parte 2: Sistemas robot e integración [16].

Para los robots colaborativos está la ISO/TS 15066 [17]. Se trata de una especificación técnica con directrices adicionales que ayudan al integrador a evaluar y adecuar medidas para el uso de robots colaborativos.

La norma ISO 10218-1 [15] en su apartado 5.3.8.3 define 4 modos de trabajo diferentes para los robots diseñados para trabajar de manera colaborativa.

Para desarrollar el robot colaborativo en las cadenas de producción se han ido desarrollando varios métodos. Ahora se expondrán los cuatro métodos principales que tienen las empresas de operación colaborativa entre hombre y robot en una fábrica.

2.1.1 PARADA SEGURA MONITORIZADA

Este método consiste en colocar al robot industrial trabajando en una zona de trabajo la cual está en un habitáculo cerrado (Figura 3), y si alguien quiere entrar para realizar un cambio o mover un objeto, primero se debe de detener el robot, y después realizar la operación. Una vez el trabajador abandone el área de trabajo el robot puede volver a seguir operando. No se permite que humano y robot puedan moverse a la vez permitiendo el movimiento del robot solo cuando el operador se encuentra fuera del espacio de trabajo. El robot se detiene, pero no se corta el suministro de energía de este. En el momento en que el operador abandona el espacio de trabajo el movimiento se reanuda de manera automática. El beneficio que esto ofrece es la facilidad y velocidad para reanudar el funcionamiento automático [18].

El principal problema de este método es que mientras la persona esté dentro del habitáculo, la fábrica pierde tiempo de producción impidiendo que operario y robot trabajen a la vez y es el principal problema que intenta resolver el robot colaborativo.

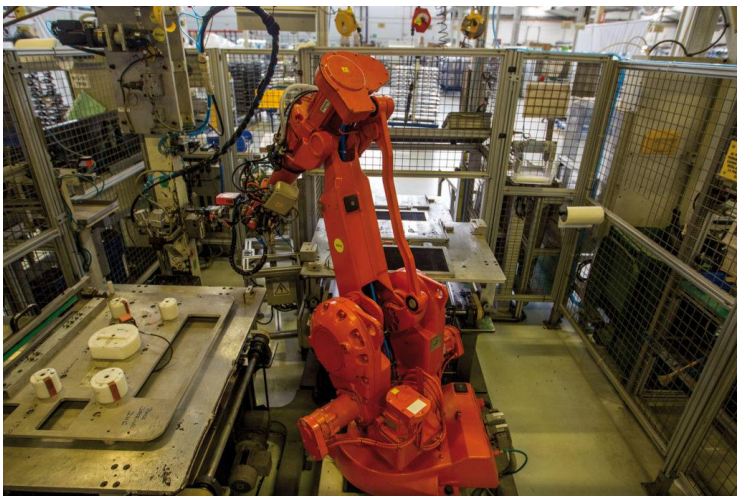

Figura 3. Robot industrial enjaulado [19]

2.1.2 GUIADO MANUAL

El robot trabaja de manera automática hasta que un operario entra en su zona de trabajo. A partir de ese momento se detiene y únicamente se mueve si el operario lo guía manualmente. El operador deberá utilizar un dispositivo manual habilitador para transmitir los comandos de movimiento. Antes de que el operador entre en el espacio de trabajo colaborativo, el robot deberá efectuar una parada monitorizada de seguridad. Para guiar el cobot, el operador dispondrá de un dispositivo de guiado manual acorde con la normativa ISO 10218-1 (5.6.4) [15]. Este dispositivo de guiado deberá situarse en el efector final del cobot o cerca de él. Hoy en día la mayoría de los cobots pueden ser "enseñados" mediante la técnica de guiado manual, es decir, se utiliza esta función para programar de manera sencilla, rápida y sin apenas programación (o ninguna) los puntos de paso o trayectorias que el robot debe describir [18]. El dispositivo de guiado manual debe incorporar un pulsador de paro de emergencia, salvo que se cumplan los requisitos de exclusión de este dispositivo de parada recogidos en el apartado 5.4.5 de la norma ISO/TS 15066 [17].

Ejemplo de guiado manual es el robot desarrollado por Omron, de la TM series [20] se muestra en la Figura 4.

Figura 4. Demostración de robot manual. Omrom TM series [20]

2.1.3 POTENCIA Y FUERZA LIMITADA

En este modo el robot trabaja en automático y en el momento que colisiona con cualquier obstáculo se limita su potencia impidiendo que dañe severamente, ya sea al objeto al que impacta, o al robot mismo. Los resultados con los robots que tienen este método son bastante satisfactorios ya que solo se interrumpen si colisionan con algo o alguien, no si entra un operario en la zona de trabajo. Las principales aplicaciones de este modo de trabajo deben ser para aplicaciones de bajo riesgo, poca carga y velocidades bajas. Un ejemplo de un robot con potencia y fuerza limitada es el robot colaborativo de Yaskawa, el Motoman HC10 [21] (Figura 5).

El problema es que no se puede implementar en la mayoría de los robots industriales ya que requiere de un *hardware* específico con sensores de fuerza que pocos robots tienen en la industria y por lo tanto su implementación sería muy costosa a diferencia de los otros dos métodos comentados previamente.

Figura 5. Límite de potencia y fuerza. Yaskawa Motoman HC10 [21]

2.1.4 CONTROL DE VELOCIDAD Y SEPARACIÓN

El robot opera en modo automático con un sensor constantemente controlando que nada entre en su área de trabajo. En el momento que un trabajador u objeto entre en la zona de trabajo, un programa calcula la distancia que hay entre el robot y el obstáculo, dejando siempre una distancia de seguridad mínima en la cual el robot se detiene y según se aleja va a una mayor velocidad (Figura 6). Con esto se consigue la mayor velocidad posible teniendo en cuenta la seguridad con el operario mientras él trabaja. La distancia entre ambos es calculada por algún sensor externo, que se estudiará más adelante. Se puede aplicar a muchos robots industriales pero el problema es la velocidad de procesamiento ya que está constantemente realizando un análisis de su entorno y necesita una respuesta rápida [22]. Este es el método elegido para llevar a cabo el proyecto de robot colaborativo.

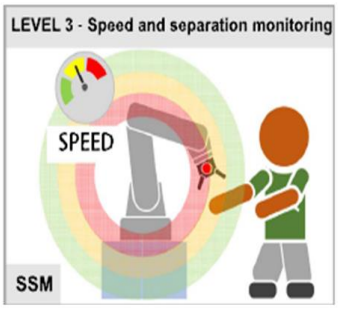

Figura 6. Control monitorizado de la velocidad y separación [23]

Cada modo de trabajo tiene sus distintas aplicaciones y se concluye que no hay una solución única para cada aplicación y que en muchos casos existirán combinaciones de modos de trabajo.

Desde el punto de vista de la seguridad, como se cita en distintos trabajos consultados los dos últimos modos de trabajo: límite de potencia y fuerza y, control monitorizado de la velocidad y separación son los que más riegos pueden producir en los humanos y deben ser desarrolladas nuevas capacidades para evitar accidentes graves [23].

2.2 SENSORES

Los sensores son esenciales para que los robots sean capaces de reconocer el ambiente en el que se encuentran y lleven a cabo las tareas/movimientos que se les requieran. Un robot sin sensores únicamente puede realizar movimientos repetitivos. Los sensores serían los sentidos del robot y por tanto gracias a éstos, se les proporciona visión, escucha, movimiento y tacto. Estas capacidades hacen posible que se pueda incrementar su capacidad. Así, existen distintos tipos de sensores según la información que se quiera trasladar al robot. Principalmente son:

- Visión y proximidad: son similares a los que se utilizan para la conducción de vehículos, pueden incluir cámaras, infrarrojos, sonar, ultrasonidos, radar, láser y LiDAR. Puede darse el caso que se utilicen distintos tipos de estos sensores, pudiendo en la integración de la información, determinar el tamaño e identificar un objeto, así como la distancia a la que se encuentra.
- Fuerza y táctil; permiten recopilar información necesaria para que el robot pueda levantar y recoger objetos de distintos tipos sin dañarlos o dejarlos caer, así como también los sensores de par permiten medir y controlar la fuerza rotacional necesaria para estos movimientos.
- Radiofrecuencia; recopilan información de códigos y permiten adquirir al robot otra información adicional.

- Acústicos; son micrófonos que permiten recibir sonidos y comandos de voz, así como identificar sonidos inusuales en el ambiente donde se encuentre ubicado.
- Temperatura; se utilizan para conocer la temperatura del entorno, lo que permitiría detectar ambientes con temperaturas de riesgo o puntos de temperaturas elevadas para humanos.

Considerando el objetivo del proyecto, donde se necesita información sobre objetos y su cercanía, se indican los sensores de visión y proximidad más comunes y sus principales características.

2.2.1 SENSOR DE ULTRASONIDOS

Este tipo de sensor es muy utilizado para medición de media distancia sin contacto, sobre todo en los sistemas de navegación de vehículos y robots móviles. El sensor ultrasónico utiliza el método de tiempo de vuelo "time of flight" (TOF) para medir la distancia, que es el tiempo que tarda un pulso en ir desde el trasmisor a un objeto observado y volver al receptor. En el caso de detección de objetos cercanos, este tipo de sensores son los más adecuados [24]. Es fiable y robusto cuando se conocen las condiciones de trabajo y se dispone comercialmente de bastantes modelos, como el UB 1000-18GM75 (Figura 7) de la empresa Pepperl+Fuchs [25]. Como el área de trabajo de este proyecto es amplia, utilizar un sensor solo no sería suficiente, y otro inconveniente es que para determinadas formas geométricas el sensor no detecta la distancia, y como es desconocido el tipo de obstáculo que se puede introducir en el área de trabajo no se ha escogido este sensor.

Figura 7. Sensor ultrasónico UB 1000-18GM75 [25]

2.2.2 LIDAR (LIGHT DETECTION AND RANGING)

Es un sensor capaz de obtener una información de sus alrededores en 3D con bastante exactitud y baja carga de datos, evitan problemas de iluminación, sombras geométricas, pero son bastante caros, aunque ya hay empresas como Intel que está lanzando modelos más baratos [10]. Este método de seguridad se ha planteado introducir en las industrias, pero debido a su alto coste, no se ha elegido este sensor. No obstante, se utiliza para detectar los obstáculos en los coches autónomos [26]. La Figura 8 muestra un ejemplo de sensor LIDAR.

Figura 8. Sensor LIDAR de coche autónomo. Velodyne [27]

2.2.3 CÁMARA RGB-D

Los sensores de imagen cada vez son más empleados debido a su reducción en el precio y grandes prestaciones. En los últimos años las cámaras se han ido desarrollando más y más. El laboratorio de Comillas ICAI dispone de las cámaras, con sensores RGB-D [28] similares a los de la Figura 9, las cuales son capaces de tomar imágenes a color y medir la profundidad de los objetos. El sensor infrarrojo tiene problemas con la luz solar [7] , pero si se utiliza en interior esto no es un obstáculo y proporciona información adicional, por lo que no se descarta su uso en futuros proyectos. Este es el dispositivo elegido para detectar si un objeto, o persona entra en el lugar de trabajo.

Figura 9. Cámara RGB-D. Intel RealSense D435 [28]

2.3 TÉCNICAS DE PROCESAMIENTO DE IMÁGENES

En este apartado se comentarán las técnicas existentes y las que se van a utilizar para procesar las imágenes que toma en todo momento la cámara y poder detectar, tanto los objetos dinámicos como estáticos que entren en el área de trabajo. En este proyecto se trabajó con cámara estática, por este motivo este estudio se centra en este tipo. No obstante, se ha de considerar que se pueden producir cambios en el entorno como por ejemplo distinta intensidad de luz, sombras, diferentes objetos que pueden aparecer pero que no se consideren obstáculos (papel, bolígrafo). Estos elementos distintos del entorno esperado se consideran como ruido, perturbaciones para el procesamiento de la imagen que se capte por lo que el algoritmo que se diseñe ha de considerarlas para evitarlas. Este último objetivo aún no se logrado alcanzar a pesar de haber numerosos estudios y trabajos que se comentan a continuación, para resolverlo. Los métodos de detección de movimiento se basan en estos tres principios: el flujo óptico, la diferencia temporal, y la sustracción de fondo [29].

2.3.1 FLUJO ÓPTICO

Hay varias definiciones para el flujo óptico. Uno de los primeros autores que introdujo este concepto fue Von H. Helmholtz [30] en 1925, como el movimiento aparente de un patrón de brillo sobre una superficie y/o borde en una escena, causado por el movimiento relativo entre un observador y la escena. Este autor indicó que se debía a la percepción de las variaciones de la imagen en la retina. Asimismo, otro autor indica que el flujo óptico consiste en la distribución aparente de velocidades en la imagen como resultado del movimiento de la luminosidad entre dos imágenes [31] Esta distribución de velocidades se utiliza para estimar la naturaleza tridimensional y la estructura de la escena, así como el movimiento 3D de los

objetos y la cámara relativa a la escena. La estimación de movimiento del flujo óptico consigue, por tanto, que los objetos puedan ser detectados y seguidos en la escena elegida. Asimismo, se incluye el posible movimiento del observador y de los objetos del ambiente, además de su estructura [32][32].

Para simplificar los cálculos del flujo óptico hay diversas aplicaciones como por ejemplo el método de expansión de Taylor [33]. A diferencia de la mayoría de las aplicaciones de medida de flujo óptico todos los píxeles captan la misma superficie (objeto) que se desplaza, pero estos sensores están programados para calcular una sola dirección de flujo óptico. En la Figura 10 se muestra el funcionamiento de un algoritmo con este aspecto donde la imagen anterior (I^{F-1}) se compara con la actual (I^{F}) en diferentes posiciones de los píxeles. El *offset* entre el centro de la plantilla de la imagen anterior y el punto con mejor coeficiente de correlación, p, indican el desplazamiento producido [35].

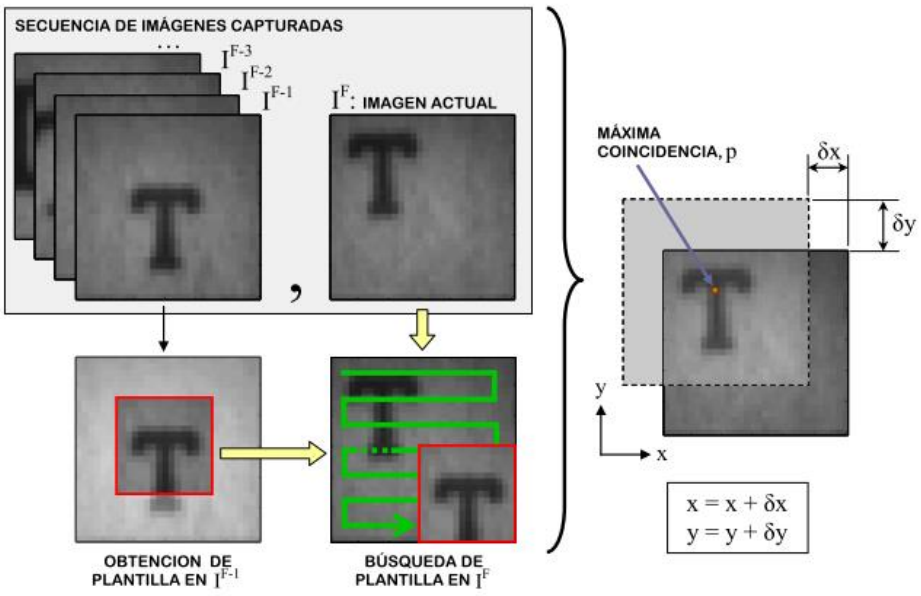

Figura 10. Ejemplo gráfico de algoritmo de navegación del sensor de flujo óptico[35]

Otros métodos son también muy utilizados, como el de Silva Blanco et al. [31]. Otros autores R. Nelson y Y. Aloimonos [36], emplean medidas de divergencia del campo de flujo como señal cualitativa para poder esquivar los objetos.

El flujo óptico si bien puede utilizarse para detectar objetos independientes en movimiento incluso con cámaras no estáticas, la mayoría de los métodos basados en este sistema son computacionalmente complejos y no se podrían aplicar en casos de vídeo completos en tiempo real sin *hardware* especializado [37].

2.3.2 DIFERENCIA TEMPORAL

En este caso, se puede detectar movimiento considerando una diferencia temporal cuando se comparan dos imágenes tomadas en instantes distintos y tratar la diferencia como movimiento. La complejidad del método es simple, y como es de esperar es sensible ante varios factores y, por lo tanto, dando errores. Estos factores pueden ser el umbral de error asignado, ya que si es demasiado bajo aparece demasiado ruido (el píxel de una planta movido una posición, iluminación o sombras) o demasiado alto (el objeto puede que ni se detecte al moverse), o el factor de velocidad de los objetos en movimiento.

La diferencia temporal, considera la intensidad entre dos imágenes como referencia[38]. Sin embargo, en la diferencia temporal se utilizan dos o tres *frame*s (cada una de las imágenes que componen un vídeo), consecutivos en una secuencia de imágenes para detectar los objetos en movimiento. Esto tiene menos eficiencia si el movimiento del objeto es muy lento puesto que siempre se considera el marco anterior como imagen de referencia, lo que puede crear interrupciones en el objeto detectado[39]. No obstante, en la actualidad hay métodos que mejoran la diferencia temporal [40].

2.3.3 SUSTRACCIÓN DE FONDO

El algoritmo de la sustracción de fondo trata de tomar una imagen como fondo, y después comparar todas las que toma con el fondo, así todo aquello diferente que se desvíe del fondo se extrae como un objeto en un primer plano. Después se separa lo que forma parte del primer plano para el procesado posterior de la imagen formando una nueva imagen como se observa en la

Figura *11*.

Un modo sencillo para modelar el fondo es captar la imagen cuando algún objeto no está en ella y considerar esta imagen como modelo de fondo. Los objetos que no sean del fondo se determinan por diferencia absoluta de *frame*s que equivale a comparar cada píxel de la imagen secuencial I^t con la imagen estimada del fondo. Si el valor absoluto de la diferencia está por debajo de algún umbral *Th*,el píxel se clasifica como fondo[41]. Esta es la ecuación siguiente.

$$Etiquetas_{(i,j)} = |\hat{B}_{(i,j)} - I^t_{(i,j)}| < Th.$$

Si el fondo se retira de la imagen, ésta mostrará el primer plano en representación binaria[42]. La Figura 11 muestra la imagen de salida cuando el fondo (primer cuadro a la izquierda) se ha restado del segundo cuadro (centro). En el tercer cuadro (derecha) se muestran los objetos del primer plano, que a menudo son conocidos como manchas. Es fundamental obtener las manchas del primer plano, con una consistencia suficiente que permita identificar fácilmente el tamaño y la zona ocupada [43]. En la Figura 12 se muestra también un ejemplo de cómo funciona la sustracción de fondo [44].

Figura 11. Sustracción de fondo, imagen a procesar [43]

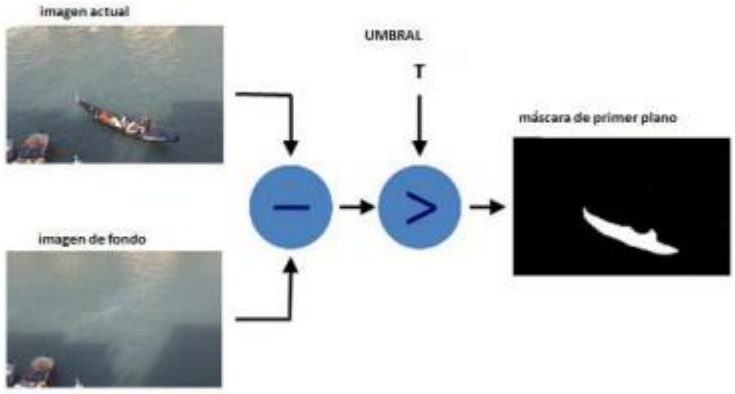

Figura 12. Ejemplo del funcionamiento general de la sustracción de fondo [44]

Como son muchas las perturbaciones que pueden ocurrir al entorno en general (por ejemplo, que atardezca) todo modelo de fondo debe de poder adaptarse a cambios de fondo graduales y a cambios repentinos. Hay dos opciones, tratar cada píxel como si fuesen variables independientes y aleatorias teniendo un modelo de fondo particular para cada uno o tratar la imagen de fondo como si fuese una variable única aleatoria, siendo esta modelada a partir de muchas imágenes (N) durante el periodo de aprendizaje.

La obtención de una sustracción de fondo adecuada puede ser compleja, de hecho algunos de los métodos de sustracción son: Running Gaussian Average, Temporal Median Filter, Mixture of Gaussians, Kernel Density Estimation (KDE), Kalman Filter, y Coocurrencia de variaciones de imagen [45], si bien se están desarrollando mejoras y nuevos métodos de manera continuada. Son de interés para el proyecto los siguientes:

- ***Running Gaussian Average:***

  Cada píxel del modelo de la imagen de fondo se define con una Gaussiana y una función de densidad de probabilidad de color durante el periodo de entrenamiento. La Gaussiana se define como se muestra en la Figura 13:

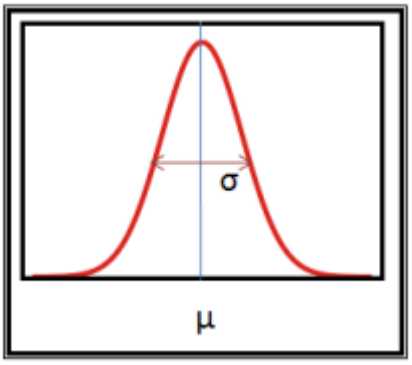

Figura 13. Función Gaussiana y parámetros estadísticos. [46]

La media, que tiene la información de las N imágenes observadas como producto de sus tres canales, y con la varianza (σ^2) que determina si el píxel pertenece o no al fondo. Se sigue el siguiente criterio:

$$|x - \mu| > k\sigma \rightarrow Primer\ plano$$

$$Resto \rightarrow Fondo$$

k es la constante de precisión con un valor entre 3 y 5. Para poder adaptar esta primera estimación a la evolución de la secuencia, se han de actualizar los parámetros de la Gaussiana una vez obtenido el modelo de fondo con estas ecuaciones:

$$\mu_t = \rho \cdot x_t + (1 - \rho) \cdot \mu_t - 1$$

$$\sigma^2 = \rho \cdot (x_t - u_t)^2 + (1 - \rho) \cdot \sigma_{t-1}^2$$

x_t es el valor del píxel de la imagen tomada, μ_t y σ_t son la media y desviación estándar respectivamente que caracterizan la función de densidad de probabilidad de color de cada píxel en ese preciso momento, y ρ que es el parámetro de absorción, que determina la velocidad de actualización del modelo en cada imagen (valor habitual 0.01). Este método precisa de poca memoria y tiene una alta velocidad de procesado. Hay trabajos donde se estudia la silueta y postura de una persona [47], apoyándose en otros anteriores [48] donde se determina la detección de individuos a partir de su contorno con el modelo Gaussiano de sustracción de fondo. Hay que destacar que estos trabajos se orientan a la aparición de un único intruso en el ambiente seleccionado y transforma la imagen de color (RGB) al espacio de

luminancias y crominancias (YVU). Se trabaja con YVU para normalizar los cambios en la iluminación.

- ***Múltiples Gaussianas:***
  Este método tiene como origen el método anterior (running gaussian average). En este se utilizan K Gaussianas por canal de color para estimar la Función de Densidad de Probabilidad del modelo de cada píxel. En este modelo estadístico en cada píxel se determina una Gaussiana por cada canal de color (RGB) obteniendo 3 funciones. Stauffer y Grimson [49] proponen este método para identificar objetos en primer plano mediante la suma ponderada de las tres Gaussianas que tiene cada píxel de la imagen. Su objetivo es que tanto el fondo como el primer plano sean representados por las Gaussianas y que se pueda aplicar un criterio de ponderación para poder discernir cuales modelan el fondo y cuales el primer plano. Así, serán parte del fondo los objetos estáticos y por tanto fuera de la zona de peligro. Como resumen, para adaptarse a distintos cambios de color, cada píxel se identifica con una media de entre 3 y 5 Gaussianas ponderadas según la frecuencia de aparición. Las Gaussianas con mayor frecuencia de aparición serán parte del fondo con la ponderación más elevada y el resto serán del primer plano. Los píxeles se pueden adaptar a cambios de iluminación puesto que cada Gaussiana corresponde a una iluminación particular como se puede observar en la Figura 14.

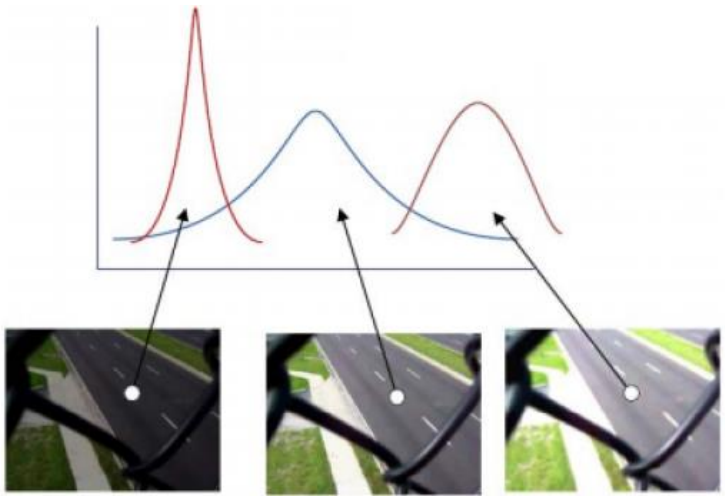

Figura 14. Diferentes Gaussianas dependiendo de la iluminación. [50]

Uno de los inconvenientes de este modelo estadístico es que supone un fondo estático en escalas de tiempo reducido, por lo que no se puede adaptar a imágenes dinámicas en el tiempo. Por ejemplo; girasoles siguiendo la orientación del sol o cualquier otro movimiento periódico de fondo. No obstante, existen estudios como el de Chan et al [50] Figura 14,*Figura 14* que permitirían hacer una adaptación a este tipo de escenas con movimientos relevantes.

- ***Kernel Density Estimation (KDE):***

Este último método consiste en estimar las funciones de densidad de probabilidad de un píxel mediante la suma de funciones Kernel calculadas a partir de algunas imágenes de muestra. La función Kernel es toda aquella función que sea simétrica, integrable, y con área unitaria. La calidad de este modelo está en el ancho de la función Kernel como se puede apreciar en la Figura 15 dónde: en la gráfica de la izquierda, con un ancho de banda pequeño la estimación es demasiado variable, teniendo picos mayores que en cualquier otra función de probabilidad. En la gráfica del medio, un ancho demasiado grande enmascara la función de probabilidad. Finalmente, en la gráfica de la derecha se puede observar el ancho de banda óptimo y sus resultados.

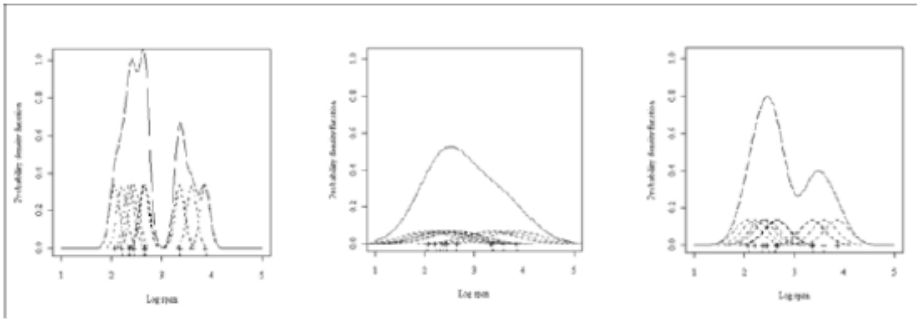

Figura 15. Funciones de Kernel distinto ancho de banda para una misma muestra [51]

Elgammal y Duraiswami [51] [52] emplean el método de KDE pues consideran que, al no realizarse ninguna suposición sobre otras distribuciones subyacentes y partir de una función de probabilidad a partir de datos podrá utilizarse para cualquier caso. Con todo, las densidades de probabilidad de fondo y primer plano no tendrán una forma paramétrica determinada pudiendo cambiar de una imagen a otra. En estos

trabajos el modelo de Kernel Gaussiano es utilizado por simplicidad. Emplean la función gaussiana como núcleo, por lo que no es preciso el ajuste de la distribución a una gaussiana o distribución normal.

2.4 *LENGUAJES PARA PROCESAMIENTO DE IMÁGENES*

Para llevar a cabo los tratamientos de imágenes citados anteriormente y que los entienda un brazo robótico industrial se necesitan lenguajes de programación. Algunos de ellos son más complejos, pero tienen un rendimiento mayor, y otros por el contrario son más fáciles de utilizar, pero tardan más en ejecutar el código. Como en este proyecto se persigue la reducción del tiempo del procesamiento de imágenes y la implementación del software en la industria, se utiliza uno de los lenguajes más conocidos y fiables del mercado. En la Figura 16 se observa el fuerte crecimiento de Python en comparación con otros.

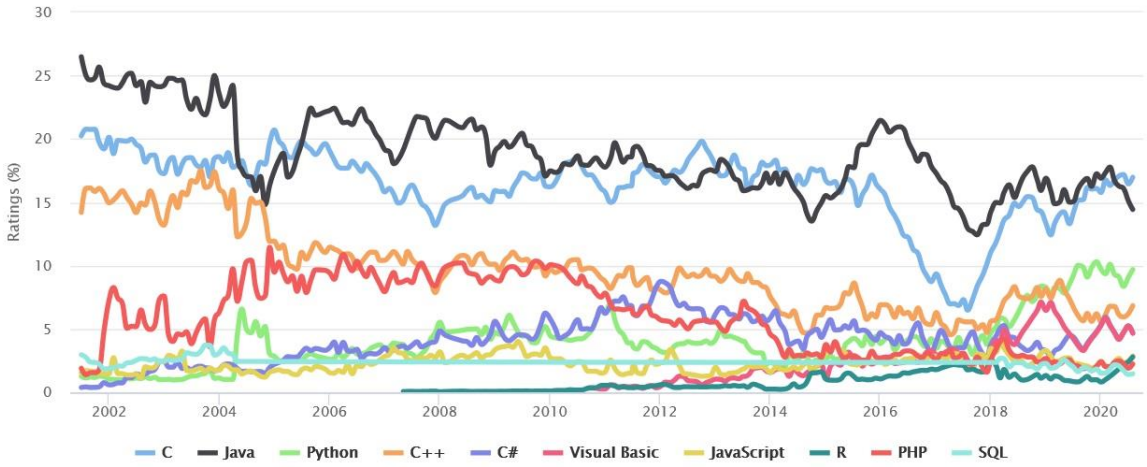

Figura 16. TIOBE Programming Community Index [53]

A continuación, se describen las principales opciones de software valoradas para este proyecto.

2.4.1 MATLAB

Matlab (MATrix LABoratory) fue creado en 1984 por Cleve Moler, fundador de MathWorks Inc.[54] y se usa actualmente en muchas universidades y centros de

investigación como software matemático. A lo largo de estos años la empresa MathWorks [55] ha ido actualizando el lenguaje de programación Matlab incorporando nuevas funciones que permiten que pueda ser utilizado para muchas aplicaciones como por ejemplo el tratamiento de imágenes o análisis estadísticos. En concreto, este software se ha utilizado recientemente para un sistema compuesto capaz de procesar imágenes de color y profundidad para dotar a cualquier robot de inteligencia artificial suficiente, permitiendo una actividad colaborativa entre los alumnos y los robots de la minifábrica de Comillas ICAI [7][10]. No obstante, una de las principales conclusiones es que la velocidad de respuesta no era lo suficientemente rápida para garantizar una mayor seguridad. Otra de las desventajas que este sistema presenta es que para utilizarse se ha de disponer de licencia, además de que en las empresas no se dispone de Matlab en todos los robots, por lo que su uso en este ámbito de la inteligencia artificial es más limitado.

2.4.2 PYTHON

Python fue diseñado a finales de la década de los ochenta por Guido van Rossum [56]. De entre los lenguajes de programación más utilizados, Python ha incrementado tremendamente su popularidad en la última década entre la comunidad científica, en concreto, las más recientes machine-learning y Deep learning libraries están basadas en este lenguaje. Según una encuesta realizada por KDnuggets, Python mantuvo su posición en la primera posición como el lenguaje más ampliamente utilizado en 2019 [57].

Esta popularidad se debe a su legibilidad y facilidad de aprendizaje al mismo tiempo que se trata de un robusto y potente de lenguaje de programación. Aparte de sus beneficios como lenguaje, es particularmente atractivo para el tratamiento de datos, machine-learning y computación [58][57]. Otras ventajas, además de las comentadas, de este lenguaje son [59]; "código abierto", por lo que es transparente y se pueden aportar mejoras, dispone de librerías accesibles, es flexible y versátil (se puede combinar con otros lenguajes, permite de manera rápida aplicar y ver los cambios, se puede ejecutar en cualquier plataforma por ejemplo Windows, MacOS, Linux), permite una potente visualización y representación de datos.

A diferencia de Matlab este lenguaje es de código abierto y aunque su lenguaje pueda ser más complicado que Matlab, al escribir en el código las funciones importantes que ejecutan el tratamiento de imágenes esto se espera que lo haga en menos tiempo. Python cada vez se utiliza más en la industria Figura 16 por lo que la implementación del sistema de seguridad en las empresas será más económica y fácil.

Trabajos realizados por otros autores indican que este lenguaje resulta más complejo para no expertos, indicando que Python es una alternativa mejor para aprender [60][61] y entender, ,pudiendo dedicar más tiempo al propio desarrollo [62]. Asimismo, la velocidad de trabajo mayor con respecto a Matlab (y la no necesidad de licencia) hacen que sea este el lenguaje de programación elegido en el presente proyecto.

2.4.3 C++

C++ fue creado en 1979 por Bjarne Stroustup que quiso extender el lenguaje de programación C (con mucho éxito en ese momento) para que tuviese los mecanismos necesarios para manipular objetos [63]. Hereda su sintaxis del lenguaje C siendo por tanto un lenguaje de alto nivel el cual, si se consigue dominar, se dominan los demás lenguajes como Python o Matlab con gran facilidad. Su rendimiento a la hora de ejecutar un código es el más alto de los 3 (Matlab, Python y C++) y también es un lenguaje que es conocido por la comunidad, top 4 (Figura 16). No obstante, se trata de una opción con un periodo de aprendizaje más largo y por los plazos de tiempo de este proyecto no se ha escogido este lenguaje, pero no se descarta para futuros trabajos.

Capítulo 3. SISTEMA DESARROLLADO

3.1 ARQUITECTURA DEL SISTEMA

En el trabajo fin de grado de Concepción Góngora [7] el sistema recogía las imágenes en directo de la cámara D435 del robot del laboratorio en vivo y las analizaba mientras el brazo robótico y los objetos entraban en el área de trabajo. El sistema de seguridad estaba conectado la cámara, después procesaba las imágenes en Matlab y enviaba los resultados a Rapid, lenguaje informático que controla el brazo robótico IRB 120 como se indica en la Figura 17.

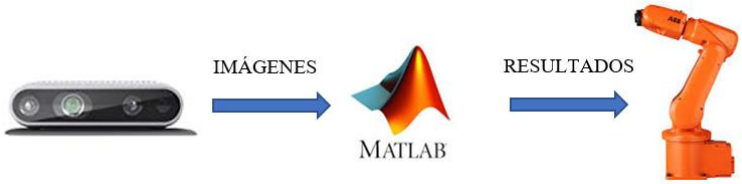

Figura 17. Diagrama de los bloques que componen el sistema desarrollado previamente [28][64][65]

Como el objetivo de este proyecto es optimizar el sistema de seguridad, se tenía planteado mantener la misma arquitectura del sistema en referente a los procesos de coger la imagen de la cámara aplicar el sistema de seguridad en un lenguaje operativo más eficiente, como por ejemplo Python, y enviarle los resultados al lenguaje Rapid para detener o no en caso de peligro al brazo robótico quedando una arquitectura como la siguiente, Figura 18.

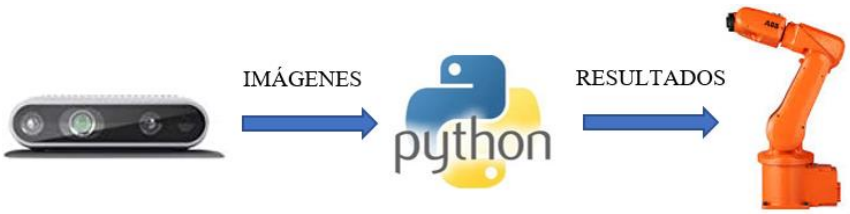

Figura 18. Diagrama de los bloques que componen el sistema desarrollado en este proyecto[28][66][65]

A pesar de tener esta planificación inicial, el acceso al brazo robótico se ha visto interrumpido por causas de fuerza mayor. Por lo que, como solución se ha grabado un vídeo con la misma cámara D435 y con el brazo robótico en funcionamiento, mientras algunos objetos entran y salen del área de trabajo. Las mayores diferencias son que el tratamiento de la imagen no es en vivo y que no se envían los resultados desde el lenguaje de programación que ejecuta el sistema de seguridad al Rapid, por lo que el diagrama de flujo es solo lectura de *frame*s de un vídeo grabado anteriormente y procesamiento de las imágenes en Python, quedándose los resultados en este último lenguaje.

3.2 ANÁLISIS Y TRATAMIENTO DE IMAGEN

El análisis de los *frame*s del vídeo tiene el siguiente diagrama de flujo (Figura 19). Primero se importan todas las librerías necesarias y se toma un *frame* del vídeo, si hay un *frame* disponible realiza todo el procesamiento de la imagen, si no lo hay, es que el vídeo ha finalizado y por lo tanto se detiene.

Una vez se tiene el *frame* para analizar, se toman 2 caminos distintos para el mismo *frame* ejecutándose secuencialmente (primero uno y después el otro).

Por un lado, se segmenta el brazo robótico por color, cuyos valores se han seleccionado previamente con un código de Python. Después, a esa máscara, que solo contiene aquellos píxeles que coincidan con el color del brazo robótico, se le aplica una transformación morfológica para quitar el posible ruido y rellenar los posibles píxeles que el proceso no haya identificado correctamente. Tras tener el brazo robótico en una máscara se calcula su contorno para poder delimitarlo y así saber el espacio que ocupa y su ubicación, guardando esa información en coordenadas del *frame* las cuales están referenciadas a un punto origen que es el mismo para todos los *frame*s del vídeo.

Cuando acaba de segmentar el brazo robótico, empieza el otro camino en paralelo. Este camino se encarga de la sustracción de fondo, es decir, detectar lo que se mueve. Utilizando el proceso de múltiples gaussianas, se obtiene en un *frame* todos aquellos píxeles que el

sistema considera que están cambiando durante el vídeo (objetos en movimiento), después elimina el brazo robótico, restando la máscara que anteriormente ha calculado y así solo quedan los objetos que pueden colisionar con el IRB 120. Después se aplica otra transformación morfológica para eliminar el posible ruido, ya sea de la substracción de fondo o a de la eliminación del brazo robótico, y se calcula el contorno de los objetos que están en el área de trabajo, almacenando también sus coordenadas con el mismo origen que las anteriores.

Por último, se calculan las distancias entre el contorno del brazo robótico y los demás objetos quedándose únicamente con la distancia mínima pudiendo generar la alarma en caso de que esté demasiado cerca el objeto. En caso de ser una distancia menor de la definida como límite se para el robot. Si no hay peligro, analiza el siguiente *frame* del vídeo hasta que no queden más *frame*s.

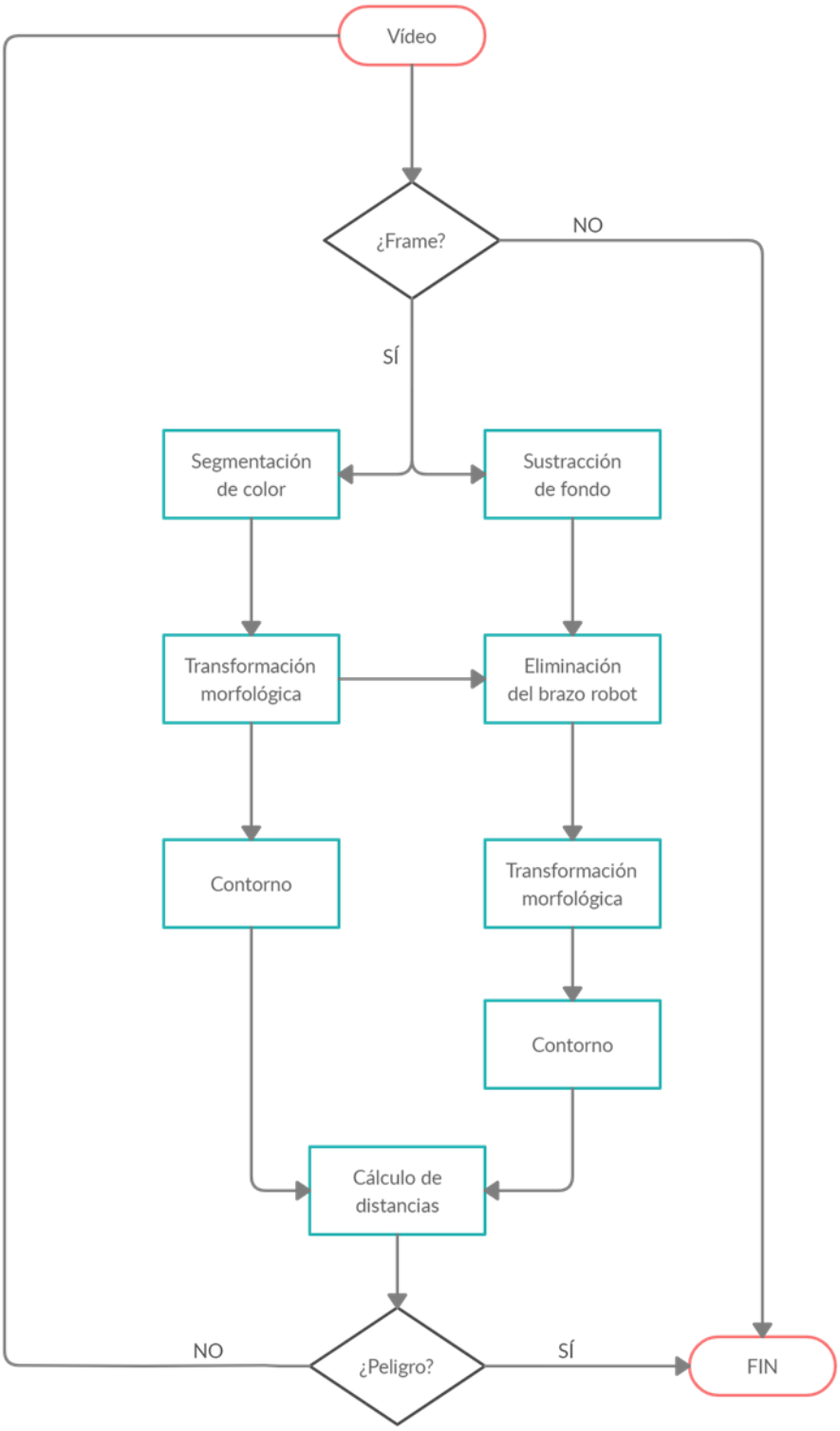

Figura 19. Diagrama de flujo para el procesamiento de imágenes ejecutado por Python

A continuación, se explican cada uno de estos apartados.

3.2.1 TOMA DE *FRAMES*

Para realizar la grabación del vídeo para este TFG, se ha utilizado una aplicación propia de la empresa Intel compatible con la cámara del laboratorio D435. Se trata de Intel RealSense SDK 2.0 [67][64]. Dicha aplicación permite grabar en distintos formatos ya sea teniendo en cuenta la profundidad, RGB, BGR o el tamaño de la imagen, etc. Como en este proyecto se compara el tiempo de procesamiento con el anterior TFG [7] se utilizará el mismo tamaño 424x240 y en formato BGR debido que ese es el que utilizan las librerías de OpenCV.

Los vídeos que genera esta aplicación son formatos ".bag", en vez de los conocidos como ".mp4" o ".avi", debido a que tiene la opción de "grabar" la profundidad. Por ello, se necesita un código en específico para poder leerlos y tratarlos. La librería para poder utilizar estos archivos es "pyrealsense2". Este es el código que se ha utilizado de una fuente de libre acceso [68]. Toda la documentación de la librería anteriormente mencionada se utilizó de fuente de libre acceso [69].

Como hubo un periodo en el cual no se podía acceder a Comillas ICAI para grabar el vídeo, se trabajó algunas partes del código sobre un vídeo de ejemplo de Matlab R2018b. Este vídeo consiste en una cámara estática grabando una autopista mientras algunos coches circulan por ella. Este vídeo se encuentra en los archivos de Matlab en la *toolbox*, de "Computer vision" [70].

3.2.2 SEGMENTACIÓN POR COLOR

Como en el área de trabajo siempre va a estar el brazo robótico y se necesita saber la distancia que hay desde cualquier objeto o persona que entre en su área de trabajo con respecto al brazo robótico, se ha de segmentar por separado. En este proyecto, esta segmentación se realizó por color ya que el brazo robótico tiene un tono muy característico. Una limitación de este método es que no pueden entrar objetos del mismo color que el brazo robótico, ya que los identificaría como tal, en este caso, objetos de color naranja.

La discretización por colores utilizada en el proyecto fue HSV (Hue Saturation Value) puesto que permite separar la luminancia de las imágenes de la información relativa al color. Esto hace más fácil el trabajo ya que la superficie del brazo robótico es metálica, y brilla según se va moviendo. En segmentación de color del tipo RGB todo está relacionado con la luminancia (intensidad), no pudiendo separar, por tanto, la información de color con la de luminancia, siendo este el motivo por el que se elige HSV en lugar de RGB.

Para seleccionar los límites del color del brazo robótico, primero deben seleccionarse los límites manualmente, con código disponible en el tutorial [71], cargando una imagen del área de trabajo con el brazo robótico (Figura 20). Posteriormente se introducen esos límites del HSV al principio del código de seguridad, para que se segmente ese color en todos los *frame*s del vídeo (Figura 20).

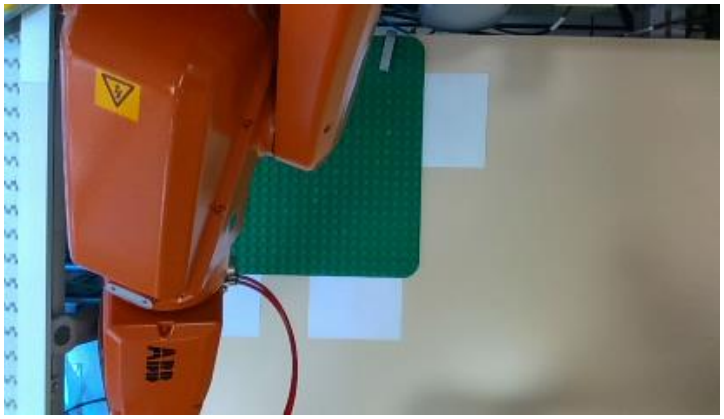

Figura 20. Imagen del robot normal

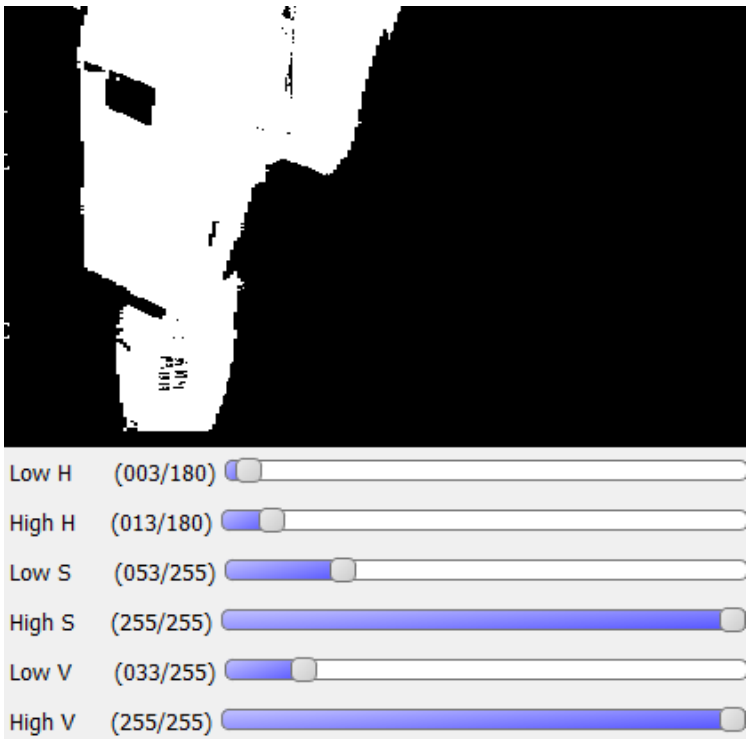

Figura 21. Imagen del robot segmentada por color

Como se observa en la Figura 21 hay algunos píxeles a la izquierda en blanco pero que en realidad no forman parte del brazo robótico, y a su vez, también falta seleccionar algunos píxeles en el interior del brazo robótico dado que esas partes son de distinto color, ya sea la etiqueta amarilla, el metal gris o las letras negras. Este problema se soluciona en transformación morfológica, dejando en la imagen solo el brazo robótico que, a su vez, está completo.

3.2.3 SUSTRACCIÓN DE FONDO

Para la sustracción de fondo se utilizarán dos métodos distintos, que después se combinan. Estos dos métodos son: Mixtura de Gaussianas e Índice de similitud estructural ("SSIM" en inglés). Esto es debido a que, dependiendo de para qué, un método es mejor que el otro.

El método mixtura de gaussianas fue creado en 2004 por Zivkovic [72] y mejorado por este mismo autor en colaboración con van der Heijden en 2006 [73]. Una funcionalidad especial de este algoritmo es que permite seleccionar el número adecuado de distribuciones

gaussianas para cada píxel. Esto proporciona mejor adaptabilidad a escenas variables debidas a cambios lentos en la iluminación, etc.

El modelo utilizado toma como referencia el fondo del vídeo y le adjudica a cada píxel una serie de valores que están dentro de una distribución normal. Cuando el píxel cambia sus valores el algoritmo lo detecta como una cosa distinta del fondo y lo declara como movimiento. Según va a analizando más *frame*s si en todos ellos hay un píxel con un valor distinto que no lo detecta como fondo, pero se queda mucho tiempo igual, el algoritmo declara el valor de ese píxel como nuevo fondo.

En OpenCV para el método de mezcla de gaussianas existen varios algoritmos distintos los cuales todos sustraen el fondo que son: GMG, MOG y MOG2. Para decidir cuál de los tres era el más apropiado se eligió el más rápido con unos resultados seguros. Según un estudio realizado de estas tres funciones en OpenCV con Python por Marcomini et al. [74] el más rápido de ellos con la precisión casi más alta es el "MOG2". Para procesar 10000 *frames* el método que más rápido lo hace es el "MOG2", tardando 150 segundos, en comparación con MOG y GMG que tardan 500 y 1100 segundos respectivamente (Figura 22).

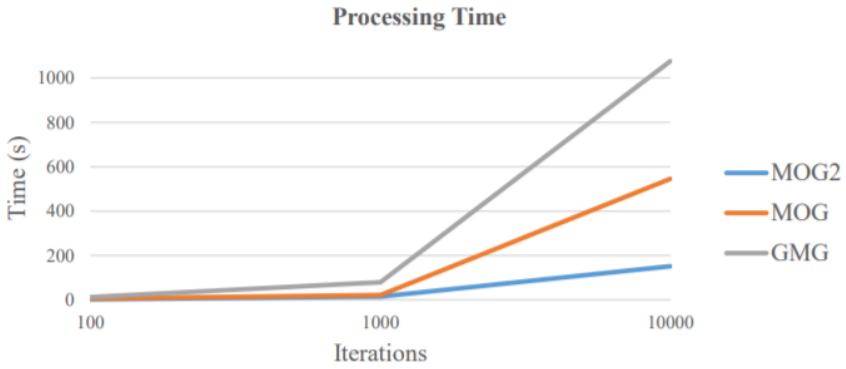

Figura 22. Tiempos de procesado de GMG, MOG, MOG2 [74]

Esta función "MOG2" tiene varios parámetros para adecuar el análisis al vídeo, que son:

- Historia: los *frame*s que se declaran como que van a formar parte del fondo, se supone que no hay nada en movimiento en estos *frame*s.

- Var threshold: Umbral en los cuadrados Mahalanobis (squared Mahalanobis), distancia entre el píxel y el modelo para decidir si un píxel está descrito adecuadamente por el modelo de fondo. Este parámetro no afecta a la actualización del fondo.
- Detector de Sombras (Detect Shadows): Si es verdadero, el algoritmo detecta sombras y las marca como grises. Esto reduce un poco la velocidad, por lo que, si no se necesita esta funcionalidad se ha de establecer el parámetro como falso, como es el caso en este proyecto.

Un ejemplo de este método, con las sombras desactivadas, es el que se observa en la Figura 23. A la izquierda está el modelo de fondo, el cual ha estado el tiempo suficiente con el brazo robótico y la esquina de la cartera negra quieta para que el método MOG2 lo considere como fondo (izquierda). Después se introduce la mano en el área de trabajo, la cual detecta como movimiento junto a su sombra (medio y derecha).

En este caso la sombra está demasiado lejos de la mano para poder ignorarla y supondría una situación falsa de peligro para el brazo robótico por lo que es necesario el segundo método SSIM para evitar este problema. El código fue realizado con un tutorial para substracción de fondo de Open Source Computer Vision [75].

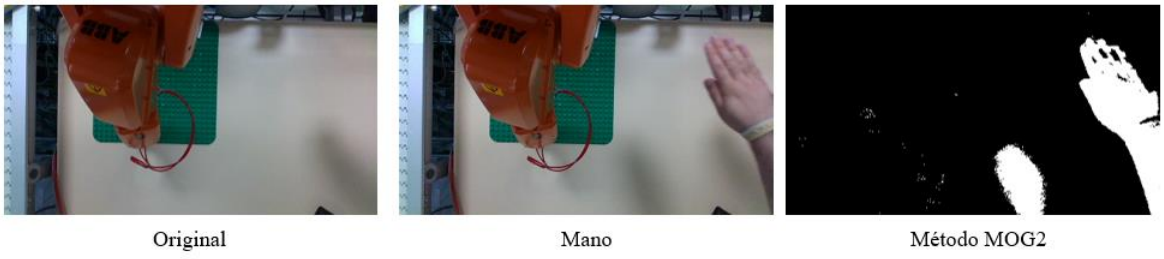

Figura 23. Modelo fondo (izquierda), Mano introducida (medio) y Máscara con MOG2 aplicado (derecha)

El método SSIM consiste en medir la similitud entre dos imágenes, la principal diferencia respecto a otros métodos, como el MSE (Error Cuadrático Medio), es que estima errores absolutos. El método SSIM fue desarrollado por Wang [76]. Este método compara dos imágenes, una es siempre la referencia, que en este caso es el fondo de la zona de trabajo sin nada moviéndose y la segunda es cada *frame* del vídeo. Tras comparar las imágenes con la

referencia, devuelve dos cosas, un valor con rango [-1, 1] el cual cuanto mayor sea el valor, mayor similitud existe entre las dos imágenes y otro que es la imagen con los píxeles que son diferentes. Para tratar las imágenes primero las transforma en blanco y negro, teniendo así un único valor para cada píxel y calcular sus diferencias, por esto es mejor para analizar vídeos con sombras o brillos.

Un ejemplo del método SSIM es el siguiente de la Figura 30.

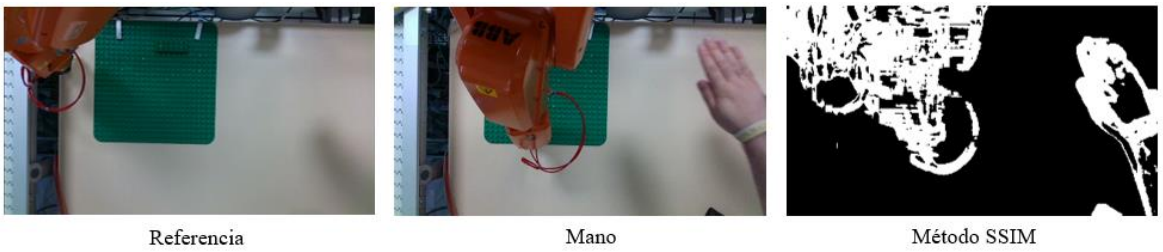

Figura 24. Modelo fondo (izquierda), Mano introducida (medio) y Máscara con MOG2 aplicado (derecha)

Como se puede observar en la imagen de la derecha, marca como movimiento el exterior de todas aquellas cosas que son diferentes con respecto la referencia, ya sea el brazo robótico, la esquina de la cartera o la mano. En este caso, sin embargo, ya no detecta la sombra de la mano, lo cual suponía un problema anteriormente. Como este modelo no ajusta el fondo según pasa el tiempo y el objeto no se mueve, da igual que la cartera este quieta durante una hora, que lo seguiría identificando como un objeto en movimiento y lo mismo para el brazo.

Por lo dicho anteriormente, se hace necesario el método MOG2 ya que este sí que lo hace, así que combinando ambos resultados de las máscaras del MOG2 y del SSIM se obtiene la imagen de la (Figura 25). En ella no está ni el brazo, ni la cartera, ya que son fondo, ni la sombra de la mano ya que no es un objeto.

Figura 25. Máscara del MO2 y SSIM Combinadas

Después las operaciones morfológicas se encargarán de formar bien los objetos. El código desarrollado para realizar el procesamiento del método SSIM fue siguiendo el código [77].

3.2.4 TRANSFORMACIÓN MORFOLÓGICA

Como se puede ver en la imagen anterior (Figura 25) el modelo puede dar problemas y detectar como movimiento algunos píxeles que en realidad son estáticos. Pasa lo mismo en la Figura 26, donde lo que se busca discretizar con la segmentación no se consigue del todo por algunos píxeles.

Esto puede ser debido a los reflejos, a las sombras o a la iluminación. Una manera de solucionar este problema es aplicándole una transformación morfológica. Las transformaciones morfológicas son operaciones simples basadas en la forma de la imagen. Normalmente se aplica sobre imágenes binarias. Necesita de dos entradas de información; la primera es la imagen original y la segunda se denomina elemento estructural o kernel, que decide la naturaleza de la operación [78].

Dos operadores morfológicos básicos son la erosión y la dilatación. La erosión consiste en eliminar píxeles que estén sueltos, o en reducir el área de un grupo de píxeles. La forma de la erosión y su tamaño dependerá del elemento de kernel previamente definido (Figura 26– izquierda). La dilatación consiste en exactamente lo contrario, amplía el área de un grupo de píxeles, pero no crea nuevos píxeles independientes. Solo puede crear nuevos si al lado tenía uno previamente (Figura 26- derecha).

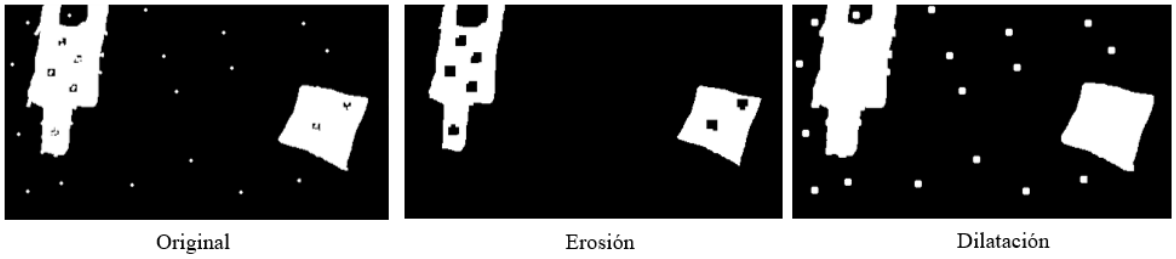

Figura 26. Imagen Original (izquierda), Erosión (medio) y Dilatación (derecha)

Como es común aplicar uno de estos dos métodos, pero luego hay que deshacer la operación para dejar la imagen como estaba previamente, surgen dos operaciones también conocidas: el *opening* y el *closing*.

En este caso, la imagen original (Figura 27– izquierda) se erosiona para destruir los píxeles sueltos que detecta como movimiento y después se dilató para dejar los objetos que están de verdad en movimiento como estaban (Figura 27– derecha). A esta operación se le llama *opening* (apertura). En el proyecto para la aplicación del *opening* se utilizó un tutorial de OpenCV [79].

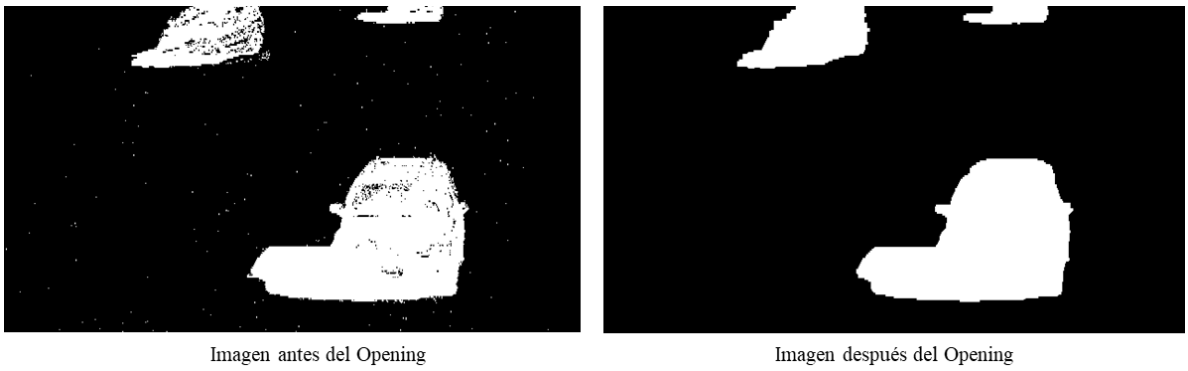

Figura 27. Imagen sin Opening (izquierda) e Imagen con Opening (derecha)

También hay otro tipo de operaciones como *gradient* que consiste en vaciar el área de un objeto, dejando solo los bordes de este (grupo de píxeles).

3.2.5 CREACIÓN DE BLOBS

Para identificar todos los elementos que se mueven en una imagen como objetos, y tener información de ellos, una opción es clasificarlos como blobs. Un blob es un conjunto de píxeles agrupados que comparten una serie de propiedades, éstas dependen de los parámetros que se escojan en el código. Los parámetros pueden ser los siguientes:

- Distancia entre blobs: si un grupo de píxeles se encuentra muy cerca de otro grupo de píxeles, si se ha definido una distancia mínima que tiene que haber entre dos blobs para que los identifique como dos diferentes y la distancia que les separa es menor, lo considera como uno solo.
- Color: se pueden coger grupos de píxeles que sean de un determinado color, pero como en este proyecto llegan solo imágenes en blanco o negro, no es de utilidad.
- Tamaño: para que identifique un grupo de píxeles como un blob es necesario que ese grupo de píxeles tenga un determinado tamaño. Si no lo tiene, se considera como píxeles de ruido y por lo tanto no es ningún objeto.
- Forma: también se puede decir que coja solo objetos con una determinada forma, ya sea una determinada circularidad, convexidad o radio de inercia. Como cualquier objeto puede entrar en la zona de trabajo del brazo robótico, tampoco se utiliza este parámetro.

En conclusión, los blobs de este sistema de seguridad en un principio se caracterizaban por ser píxeles de color negro y de un determinado tamaño. En el proyecto se utilizó este método basándose en [80]. El resultado de la aplicación de blob se muestra en la Figura 28.

Figura 28. Aplicación del Blob

Debido a una serie de problemas que presentaba este método, como a la hora de identificar los blobs con los parámetros requeridos, y a que algunas funciones daban la información en un formato poco manejable, se decidió no utilizarlo para obtener información útil de los objetos y del brazo robótico y se optó por utilizar el cálculo de contornos.

3.2.6 CONTORNOS

Para poder calcular las distancias, primero se tienen que calcular los límites del brazo robótico y de los objetos que entran en el área de trabajo. Para ello se va a utilizar una función de OpenCV llamada "Contour Features"[81], la cual calcula perímetros, el centro de los grupos de píxeles blancos de una imagen. A diferencia de los blobs, en esta función no se definen previamente una serie de parámetros. Esta función calcula primero los perímetros de todos los píxeles y luego el código creado por el usuario decide qué es un objeto y qué no.

Como se puede ver en la imagen, se encuentran el brazo robótico y dos objetos (Figura 29–Izquierda) tras haberle aplicado las transformaciones morfológicas necesarias.

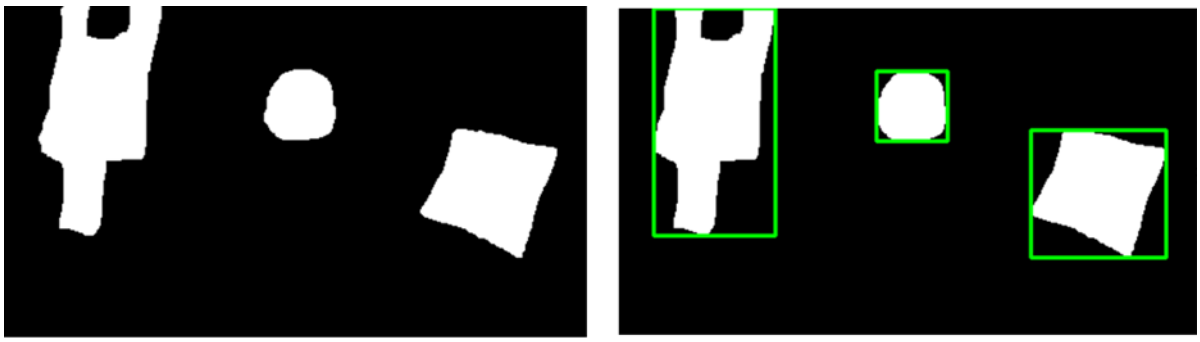

Figura 29. Imagen con tres objetos. Izquierda objetos y Derecha reconocimiento de contornos

Al aplicar el reconocimiento de contornos (Figura 29 – Derecha) crea cuadrados no orientados a los objetos como se ven en la figura, siempre con los lados de los cuadrados paralelos a los ejes de la foto. Pero esto puede ser muy impreciso ya que, si se introduce un objeto en diagonal, como el cuadrado de la derecha, las esquinas de dicho contorno no están

donde en realidad están las del objeto por lo que el cálculo de distancias entre objetos no sería el real.

Por ello se puede aplicar otro contorno de mayor precisión el cual crea rectángulos orientados como lo están los objetos (Figura 30) coincidiendo así con mayor precisión los objetos con lo que calcula el código.

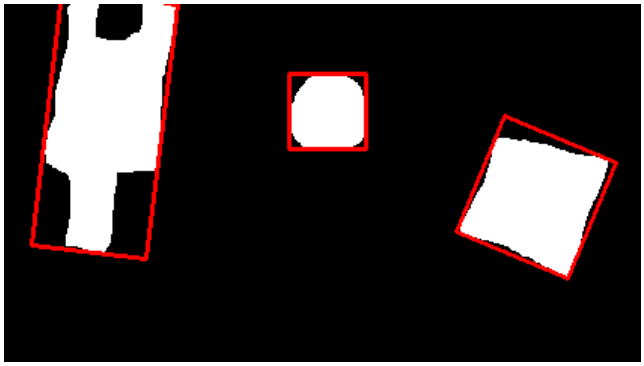

Figura 30. Imagen con los tres objetos. Reconocimiento de contornos con cuadrados orientados

Para la realización del código e implementación se utilizó el tutorial [82]. Como es posible que tras las transformaciones morfológicas se haya filtrado algún tipo de ruido no deseado, que el sistema detecta como movimiento, se considera como requisito para que sea un objeto que tenga un determinado número mínimo de píxeles. Tras calcular los contornos se almacenan en variables las coordenadas de cada uno referenciadas al mismo punto.

3.2.7 CÁLCULO DE DISTANCIAS Y PELIGRO

Para mantener una distancia de seguridad entre el brazo robótico y los objetos, hay que diferenciar el contorno y las coordenadas que crea para el brazo robótico y el resto de los objetos. Este se realiza con la segmentación de color, ya que en esa máscara solo está el brazo robótico. En la Figura 31 se muestra cómo el robot es diferenciado de los dos objetos.

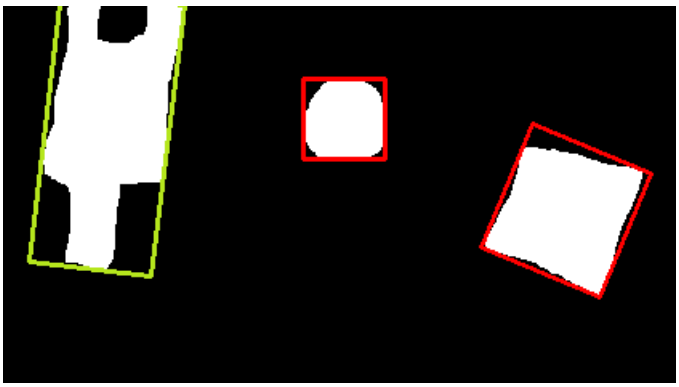

Figura 31. Imagen con brazo robótico y dos objetos. Contorno del brazo diferenciado

Para asegurar que coge bien el contorno del brazo, aparte del requisito de un área mínima solo puede haber un único contorno definido y ese contorno es el que tenga el área más grande de todos los que encuentra en un mismo *frame*, tras haber aplicado la máscara de segmentación de color.

Ahora que se tienen las coordenadas del robot y de los objetos diferenciadas, se calcula la distancia de cada esquina del robot con todas las esquinas de los demás objetos. Se considera únicamente la distancia mínima de todas esas distancias y después se valora si hay peligro de colisión o no.

Para calcular las distancias, como se tienen las coordenadas de todas las esquinas, y todas ellas están referenciadas al mismo origen, que es la esquina superior izquierda (0,0), se utiliza el cálculo de la distancia euclídea al cuadrado. Es al cuadrado ya que para realizar la raíz cuadrada, Python necesita importar la librería "math" y llamar a la función "math.sqrt()". Como lo que se busca en este proyecto es la optimización de tiempo se considerarán las distancias, a la hora de decidir si está demasiado cerca o no, al cuadrado.

$$distancia(A,B)^2 = (x_A - x_B)^2 + (y_A - y_B)^2$$

Si la distancia es inferior o igual a un cierto límite de distancia en píxeles, salta la alarma y se detiene el robot por seguridad. En este proyecto sale un mensaje en el vídeo indicando que hay peligro, simulando ser el mensaje que pararía el brazo robótico (Figura 32). Si la distancia es superior, continua con la siguiente imagen.

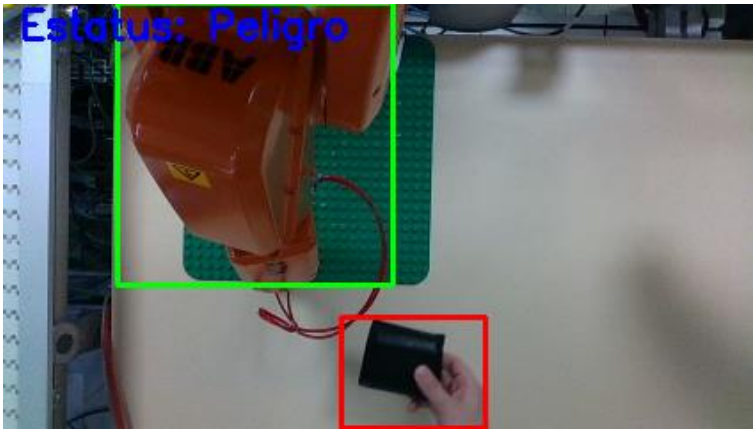

Figura 32. Peligro por proximidad

Capítulo 4. ANÁLISIS Y RESULTADOS DE TIEMPOS

En este capítulo se analizan los tiempos que tarda en procesar Python y Matlab el código del sistema de seguridad. Para comprobar la robustez del sistema de seguridad, se ha probado con cuatro vídeos distintos en Python. Estos cuatro vídeos se grabaron a las 17:00 sin nubes en el laboratorio de Comillas ICAI, cada uno de estos vídeos tienen condiciones lumínicas distintas.

El primer vídeo, designado como (a) en todas las figuras, fue con las cortinas bajadas y con las luces de la sala encendidas. El segundo vídeo, designado como (b) en todas las figuras, fue con las cortinas subidas y con las luces de la sala encendidas. El tercer vídeo, designado como (c) en todas las figuras, fue con las cortinas subidas y con las luces de la sala apagadas. El cuarto vídeo, designado como (d) en todas las figuras, fue con las cortinas bajadas y con las luces de la sala apagadas.

En todos ellos se partía con la misma posición del brazo robótico y entraban y salían los mismos objetos del área de trabajo.

4.1 ESPECIFICACIONES DE HARDWARE Y SOFTWARE

Para poder analizar y decidir si el sistema de seguridad diseñado en Python es más eficiente con respecto al anterior sistema diseñado en Matlab, hay que especificar las características del ordenador en el que se han ejecutado los códigos, y en qué versiones de los lenguajes de programación se utilizaron.

El ordenador es un portátil HP Omen 15-AX201NS con un procesador Intel(R) Core (TM) i7–7700HQ CPU @ 2.80GHz 2.81GHz, con una memoria RAM instalada de 8 GB, un disco duro de 1TB y el sistema operativo Windows 10 Home de 64 bits [83].

La versión de Matlab es la R2018b, la de Python es la 3.7.4 y la de OpenCV 4.2.0.

4.2 TIEMPOS EN PYTHON

En Python hay una librería llamada "time", la cual llama a dos variables, y dependiendo de donde se coloquen estas dos variables, la librería mide el tiempo en milisegundos desde que se crea la primera variable, llamada "ts" (*time start*), hasta que se crea la segunda variable, llamada "te" (*time end*). La parte del código o las funciones de las que se quiere medir el tiempo de ejecución se encuentran entre estas dos variables.

Para medir el tiempo, no solo se mide lo que tarda en procesar un *frame*, sino que se miden todos los *frames* durante el vídeo, dado que puede haber algunos *frames* con valores atípicos que den resultados que no representan la eficacia del código. Estos resultados se representan en una gráfica. Por último, para poder comparar un valor de tiempo, se calcula la mediana o la media de todos los *frames*, ya que es posible que haya valores atípicos en algunos de ellos.

En primer lugar, está la medición de lo que tarda en procesar un fotograma completamente. Después, cada apartado del código, los cuales están en el diagrama de flujo Figura 19. No se incluyó en la medida de tiempo lo que tarda el proceso de toma de *frames*, debido a que el código toma estos *frames* de un vídeo en vez de la cámara en directo. En una situación real sí que se ha de considerar lo que tarda en recibir la imagen de la cámara.

Para todos los procesos se toma el mismo número de *frames*, los primeros 2700, la función de medida de tiempo de la librería "time", tiene como máxima resolución de 1 ms. Las unidades y la precisión con la que se presentan las medias de todos los *frames* es en ms y de dos decimales.

El tiempo en procesar todo el código de los cuatro vídeos se observa en la Figura 33. En estas gráficas se encuentran varios valores atípicos. Sin embargo, son muy pocos, del orden de la decena con respecto a los 2700 totales, por lo que se puede considerar que el código es seguro. La mediana de estos valores es de:

- Vídeo 1. Figura 33 (a), mediana de 15,01 ms.
- Vídeo 2. Figura 33 (b), mediana de 15,34 ms.

- Vídeo 3. Figura 33 (c), mediana de 15,91 ms.
- Vídeo 4. Figura 33 (d), mediana de 13,15 ms.

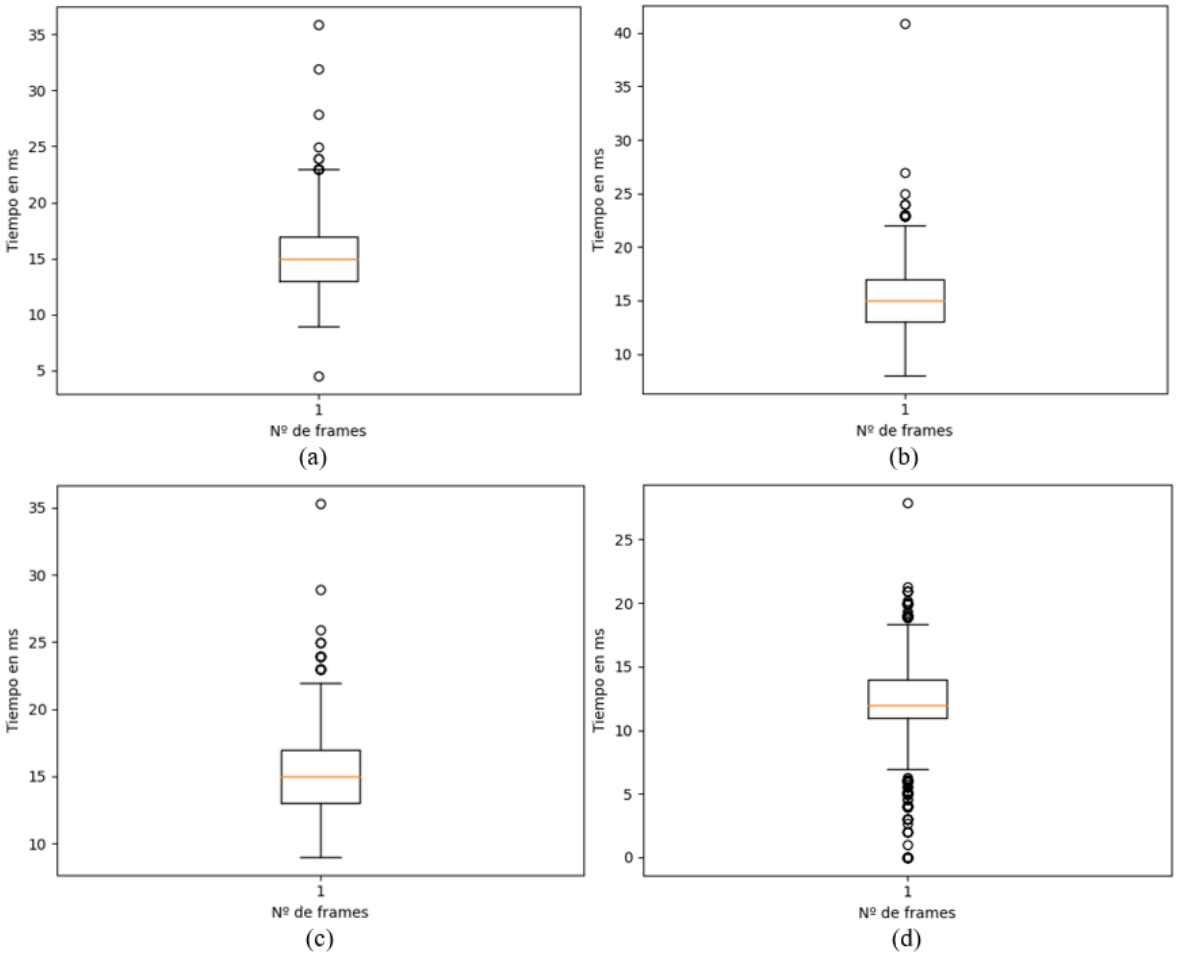

Figura 33. Tiempo total de procesamiento en Python

4.2.1 TIEMPO DE SEGMENTACIÓN POR COLOR

La Figura 34 representa el tiempo que tarda en segmentar el color. La media de estos tiempos es de:

- Vídeo 1. Figura 34 (a), media de 0,19 ms.
- Vídeo 2. Figura 34 (b), media de 0,18 ms.
- Vídeo 3. Figura 34 (c), media de 0,18 ms.
- Vídeo 4. Figura 34 (d), media de 0,15 ms.

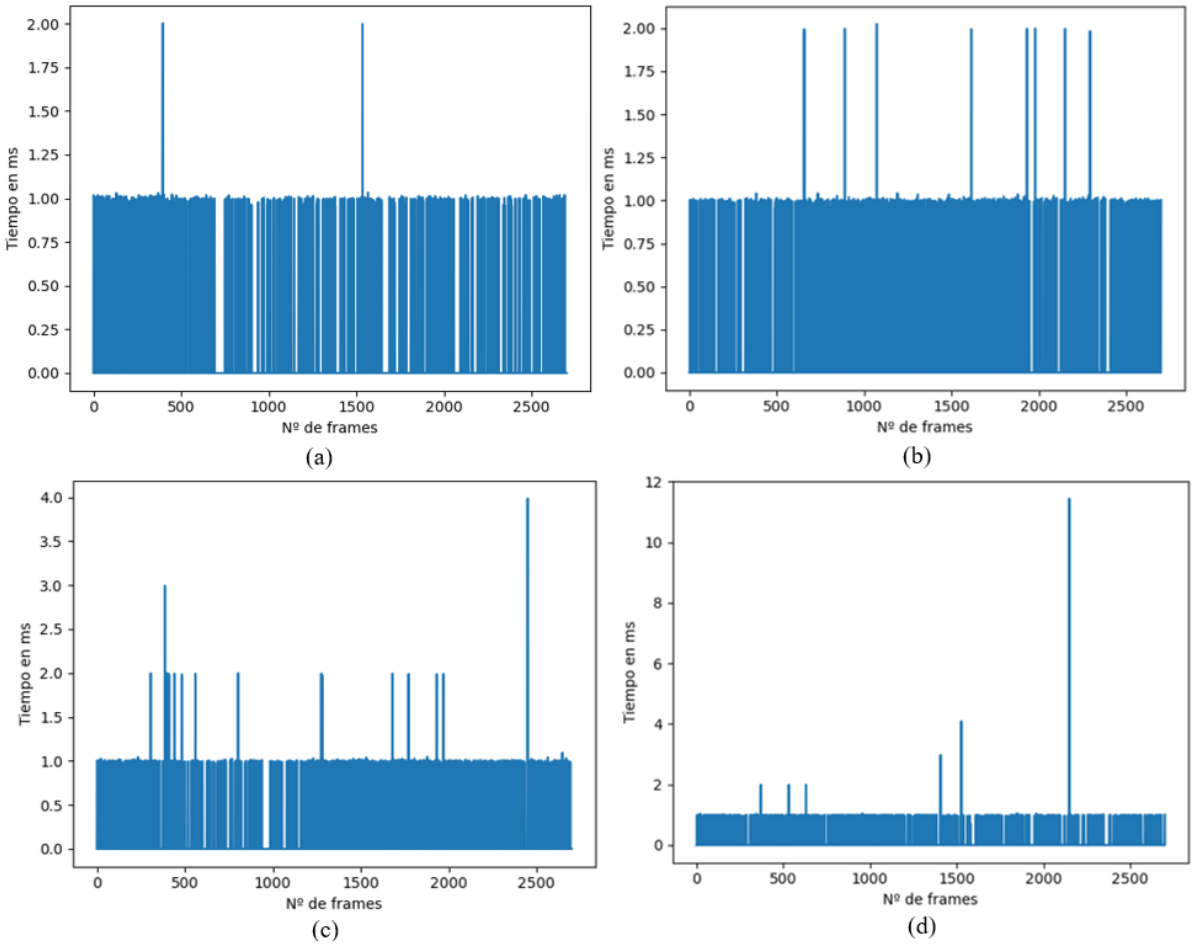

Figura 34. Tiempo de segmentación por color en Python

En esta gráfica no se considera el tiempo en ejecutar los valores iniciales del color que se quiere segmentar, ya que eso solo se ejecuta una sola vez, y no en todos los *frames*.

4.2.2 SUSTRACCIÓN DE FONDO

En la sustracción de fondo se han medido tres fases por separado: los dos métodos distintos, el MOG2 y el SSIM, y después, la operación "AND" entre las dos máscaras.

La Figura 35 representa los tiempos de ejecución del método MOG2, con la media de tiempos:

- Vídeo 1. Figura 35 (a), media de 2,17 ms.

- Vídeo 2. Figura 35 (b), media de 2,15 ms.
- Vídeo 3. Figura 35 (c), media de 2,12 ms.
- Vídeo 4. Figura 35 (d), media de 2,08 ms.

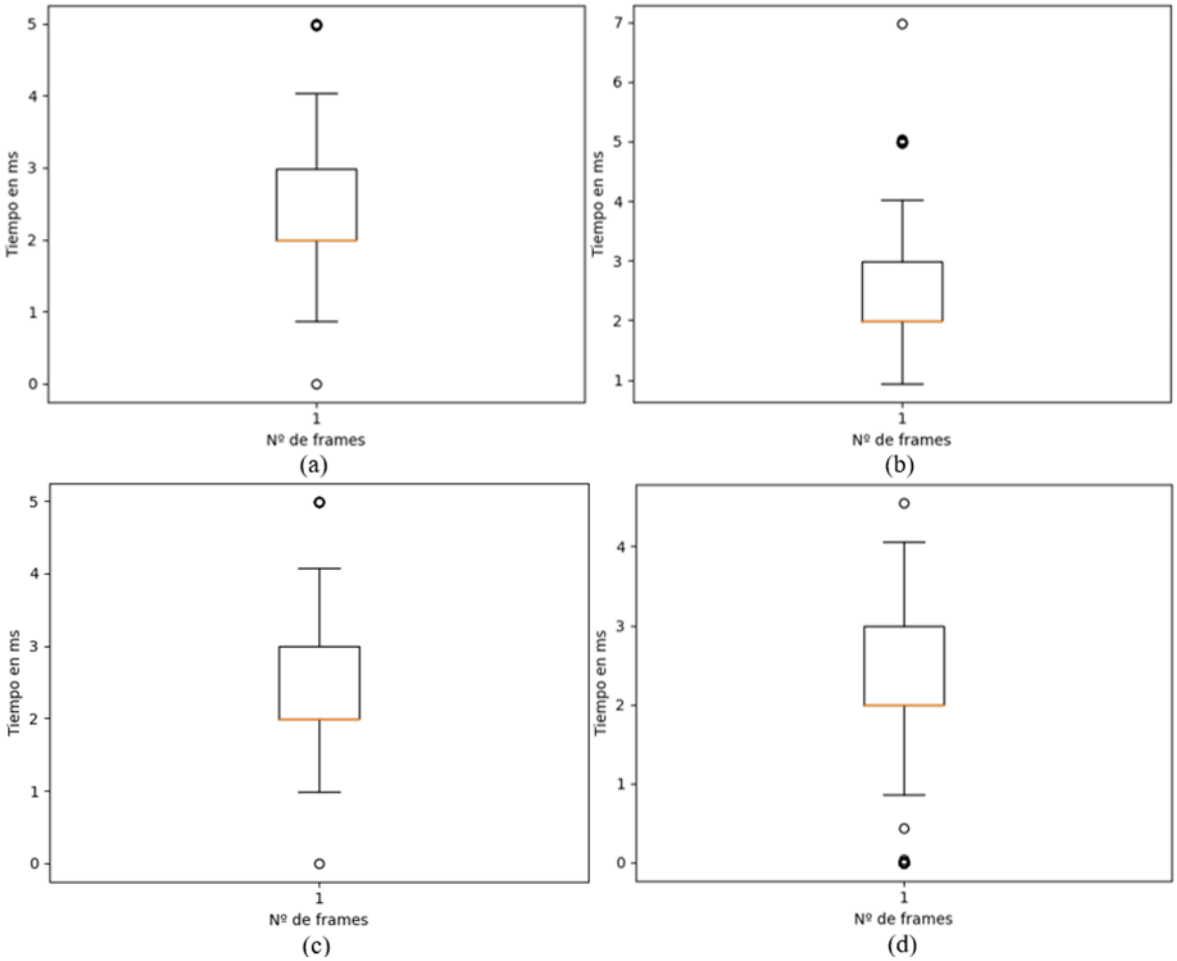

Figura 35. Tiempo de sustracción de fondo MOG2 en Python

La Figura 36 representa los tiempos de ejecución del método SSIM, con la mediana de tiempos:

- Vídeo 1. Figura 36 (a), mediana de 10,24 ms.
- Vídeo 2. Figura 36 (b), mediana de 10,39 ms.
- Vídeo 3. Figura 36 (c), mediana de 10,41 ms.
- Vídeo 4. Figura 36 (d), mediana de 8,23 ms.

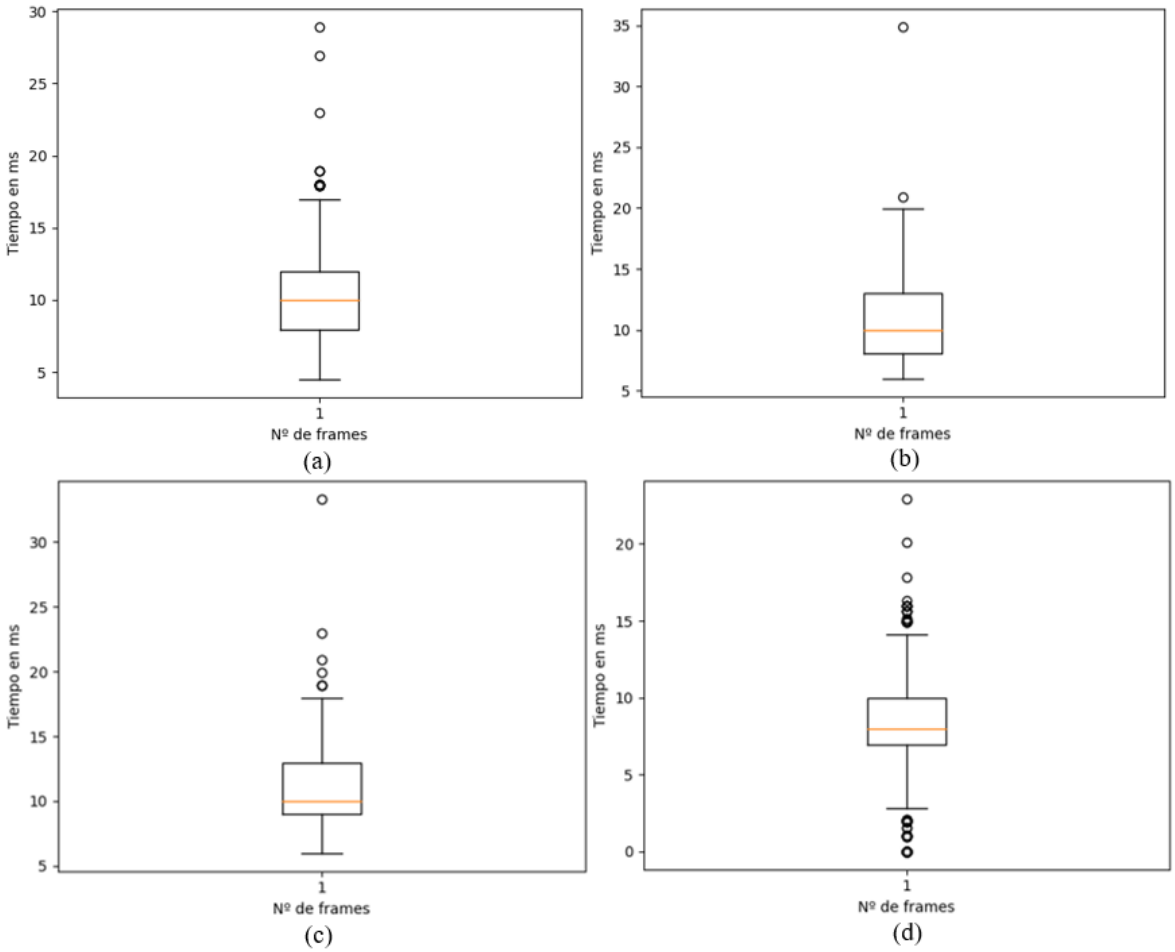

Figura 36. Tiempo de sustracción de fondo SSIM en Python

La Figura 37 representa los tiempos de ejecución de unir ambos métodos, con la media de tiempos:

- Vídeo 1. Figura 37 (a), media de 0,14 ms.
- Vídeo 2. Figura 37 (b), media de 0,12 ms.
- Vídeo 3. Figura 37 (c), media de 0,13 ms.
- Vídeo 4. Figura 37 (d), media de 0,11 ms.

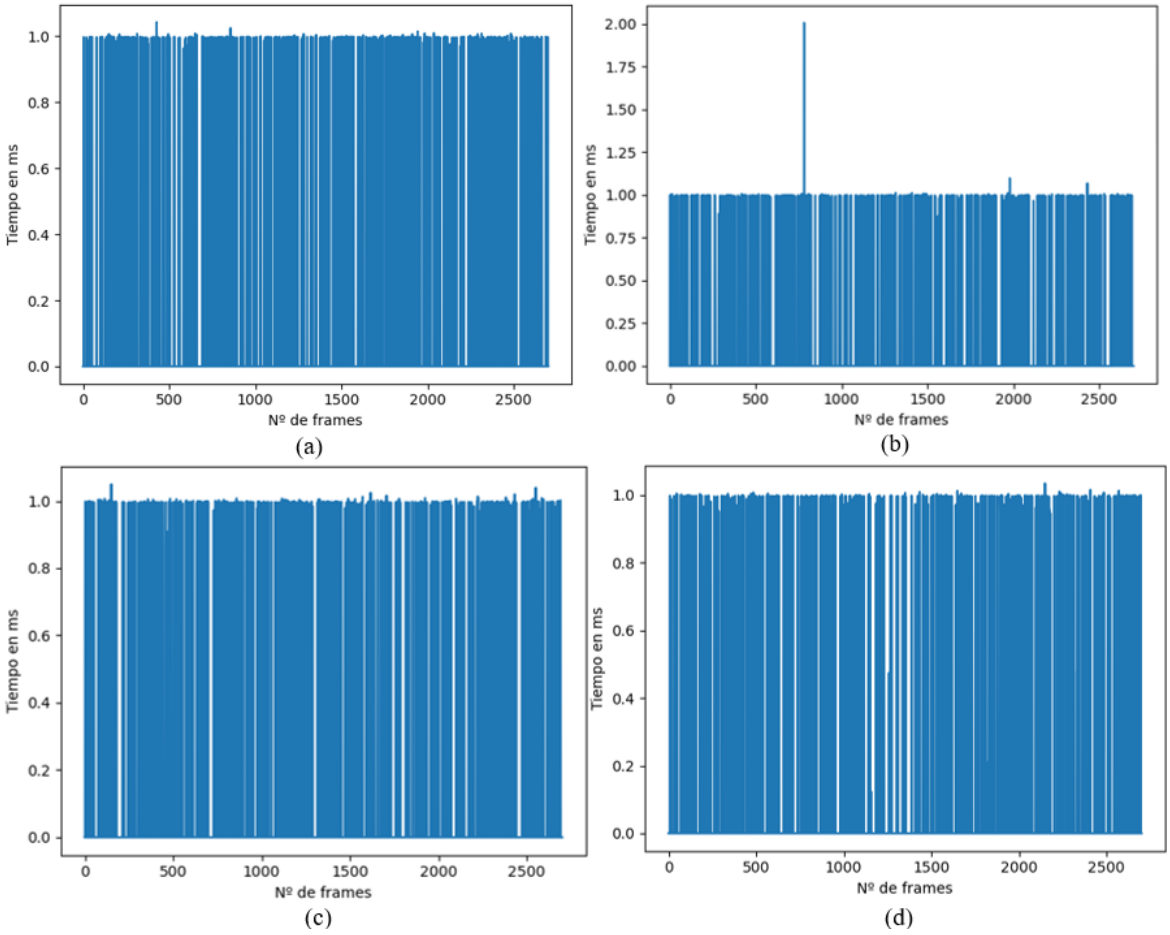

Figura 37. Tiempo de unión de MOG2 y SSIM en Python

4.2.3 ELIMINACIÓN DEL BRAZO DEL ROBOT

La Figura 38 representa el tiempo que tarda en quitar los píxeles que están en la máscara de segmentación de color a la sustracción de fondo. La media de estos tiempos es de:

- Vídeo 1. Figura 38 (a), media de 0,14 ms.
- Vídeo 2. Figura 38 (b), media de 0,13 ms.
- Vídeo 3. Figura 38 (c), media de 0,13 ms.
- Vídeo 4. Figura 38 (d), media de 0,11 ms.

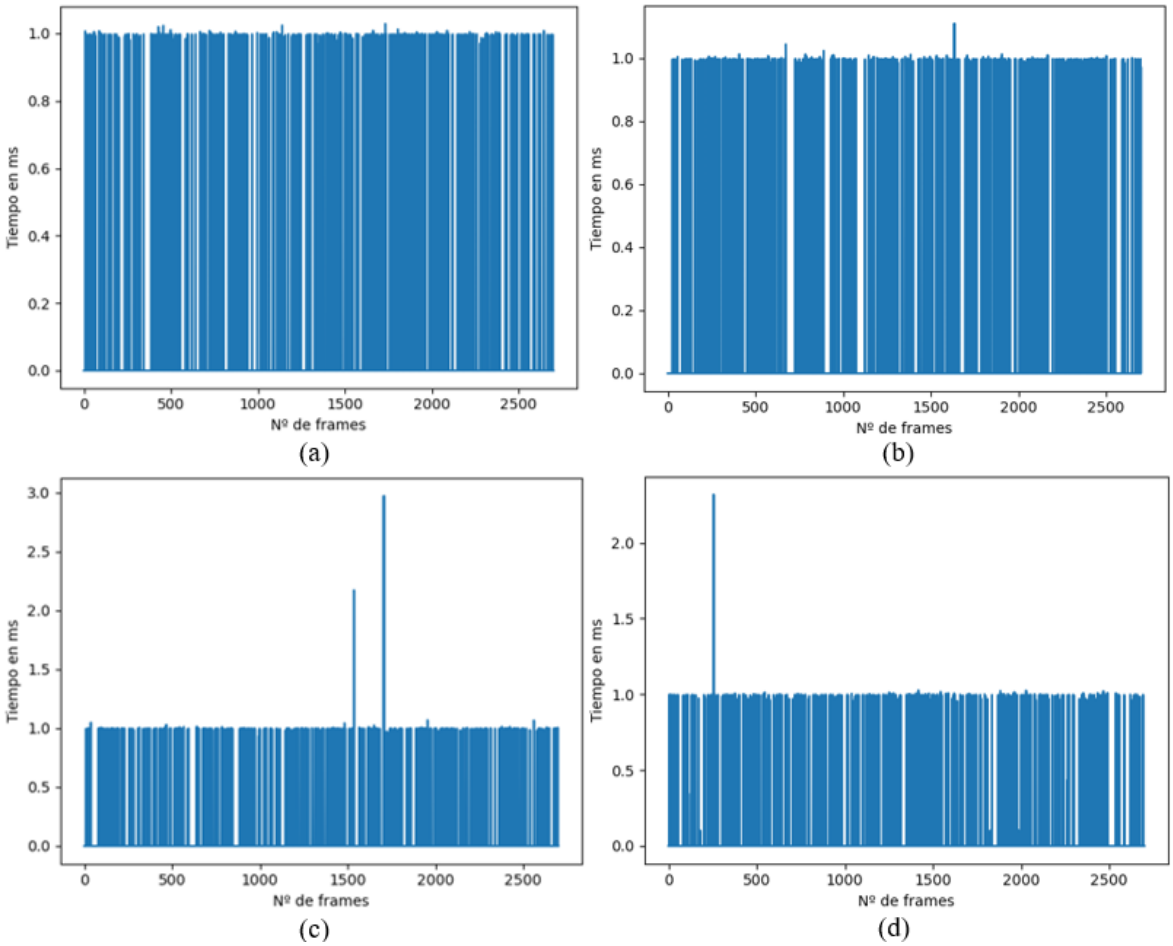

Figura 38. Tiempo de eliminación de brazo robótico en Python

4.2.4 TRANSFORMACIÓN MORFOLÓGICA

La transformación morfológica se aplica en dos sitios distintos, y no tienen las mismas operaciones, por lo que se han medido por separado, por un lado, la transformación morfológica de la segmentación por color y, por otro, la transformación morfológica de los objetos.

La Figura 39 representa los tiempos de ejecución de la transformación morfológica de la segmentación por color. La media de los tiempos es:

- Vídeo 1. Figura 39 (a), media de 1,06 ms.
- Vídeo 2. Figura 39 (b), media de 1,05 ms.

- Vídeo 3. Figura 39 (c), media de 1,06 ms.
- Vídeo 4. Figura 39 (d), media de 0,88 ms.

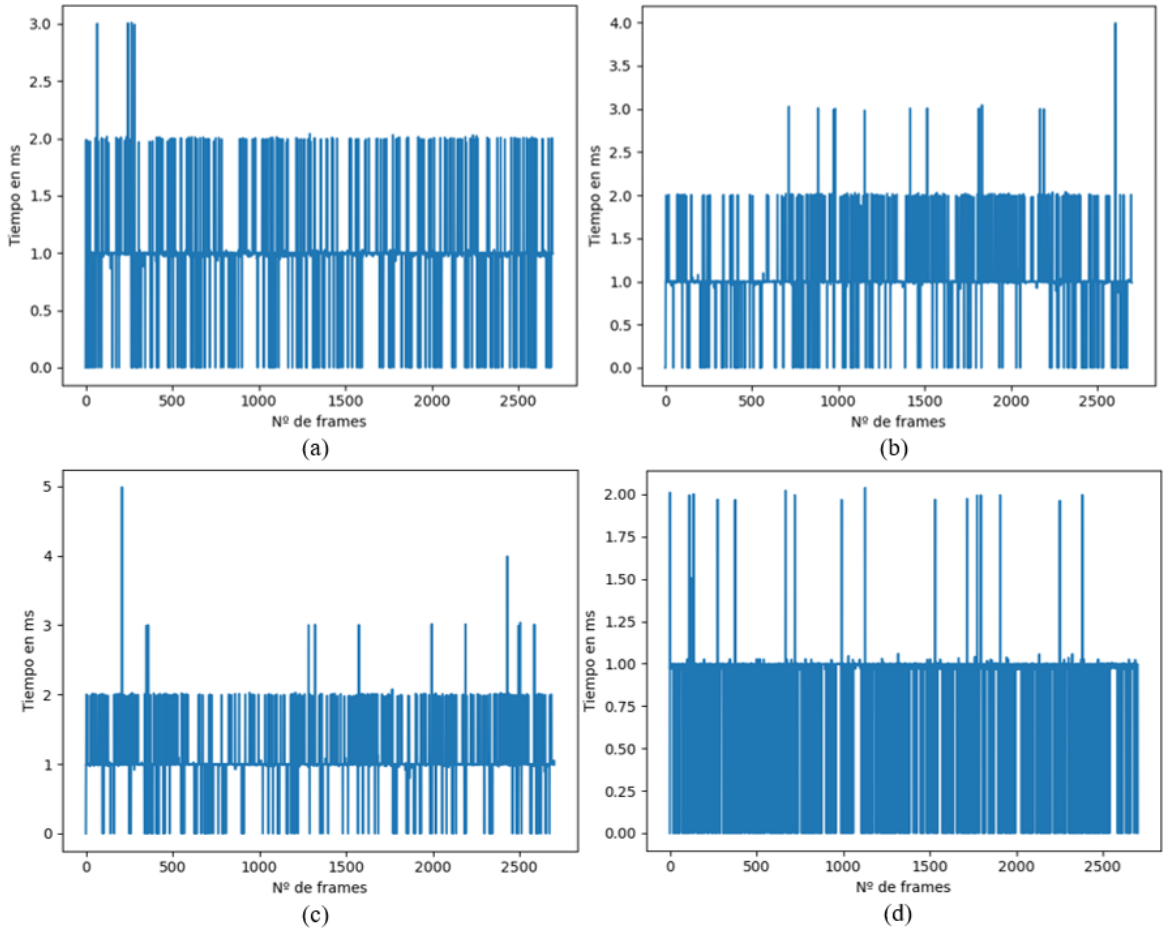

Figura 39. Tiempo de transformación morfológica de segmentación por color en Python

La Figura 40 representa los tiempos de ejecución de la transformación morfológica de los objetos. La media de los tiempos es:

- Vídeo 1. Figura 40 (a), media de 0,52 ms.
- Vídeo 2. Figura 40 (b), media de 0,53 ms.
- Vídeo 3. Figura 40 (c), media de 0,53 ms.
- Vídeo 4. Figura 40 (d), media de 0,45 ms.

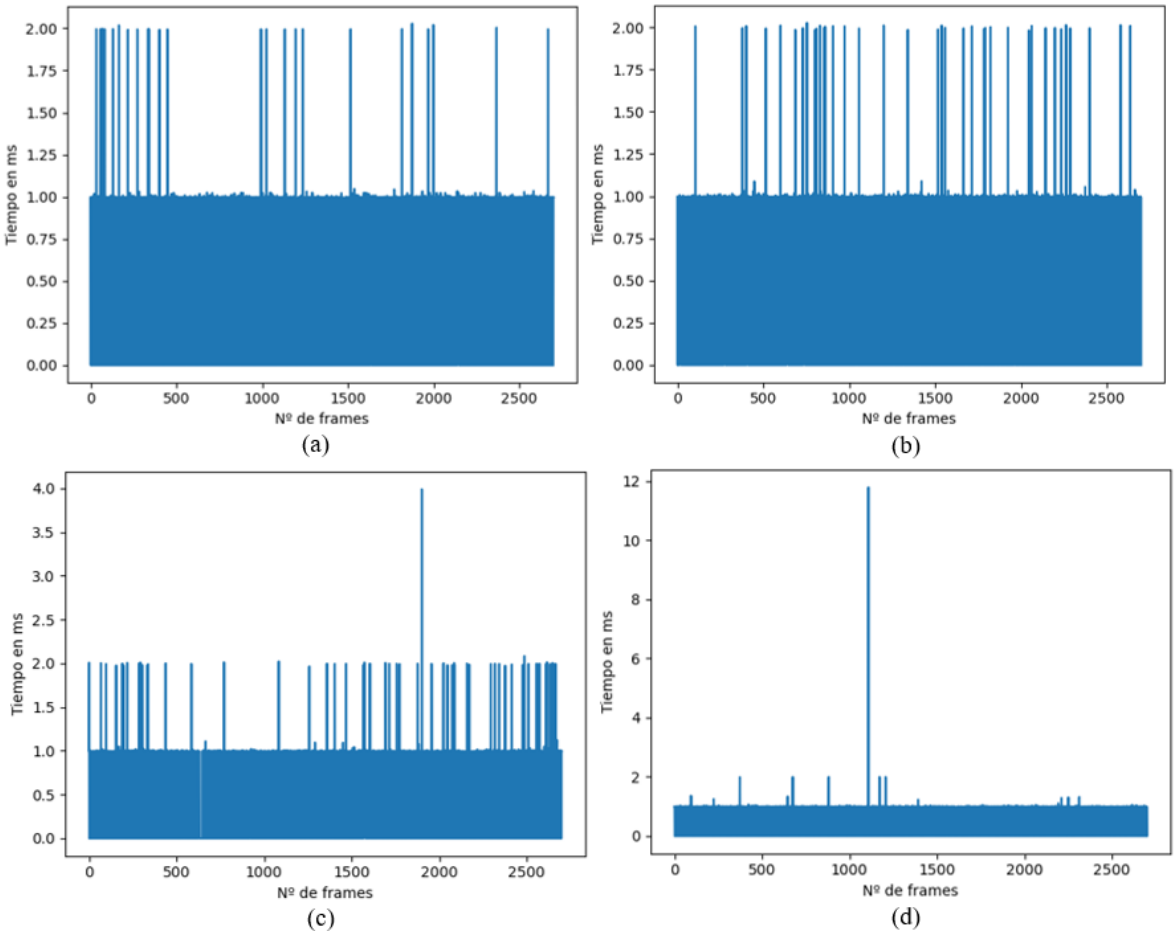

Figura 40. Tiempo de transformación morfológica de objetos en Python

4.2.5 CONTORNOS

Al igual que en la transformación morfológica, también se calculan contornos para segmentación por color y para los objetos, por lo que también se miden aparte.

La Figura 41 representa los tiempos de ejecución de los contornos de segmentación por color. La media de los tiempos es:

- Vídeo 1. Figura 41 (a), media de 0,98 ms.
- Vídeo 2. Figura 41 (b), media de 1,03 ms.
- Vídeo 3. Figura 41 (c), media de 1,00 ms.
- Vídeo 4. Figura 41 (d), media de 0,87 ms.

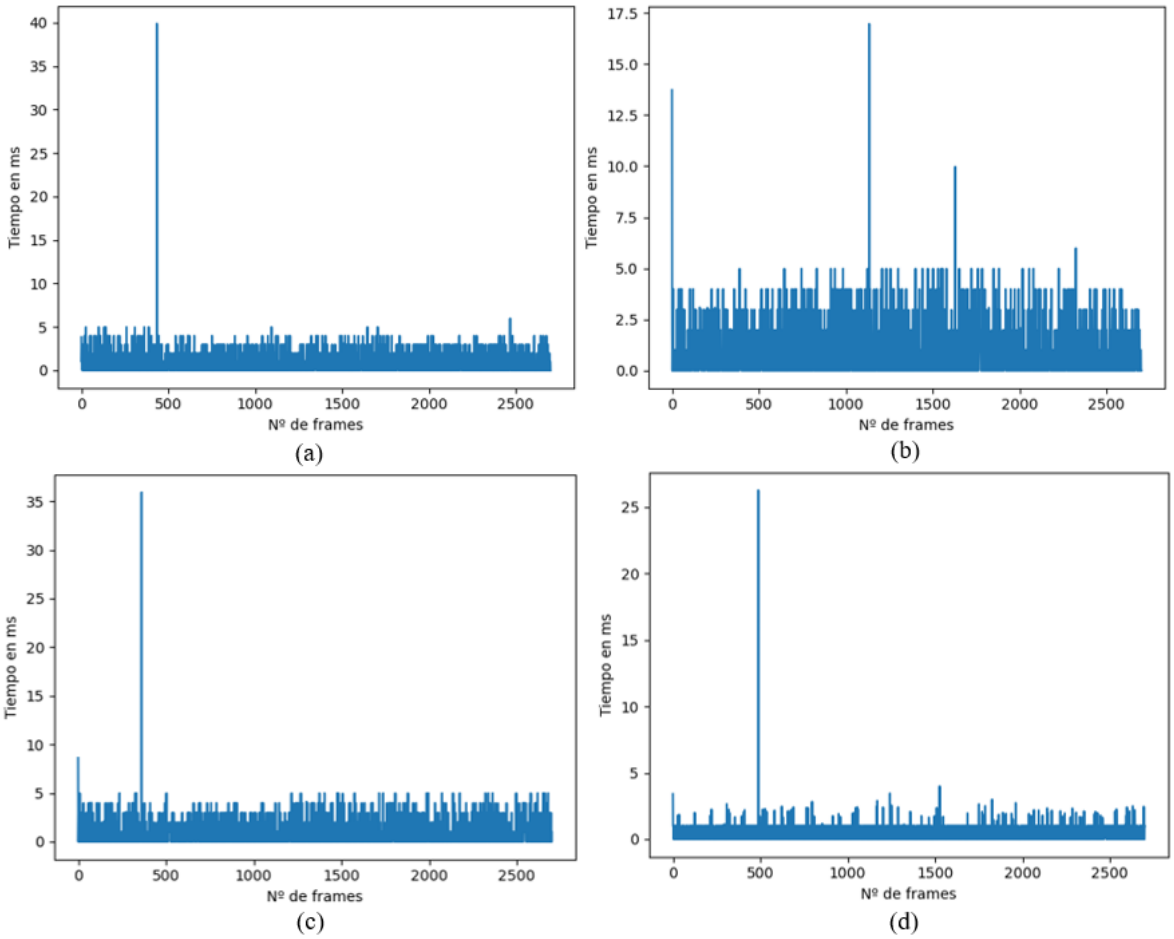

Figura 41. Tiempo de contornos de segmentación por color en Python

La Figura 42 representa los tiempos de ejecución de los contornos de los objetos. La media de los tiempos es:

- Vídeo 1. Figura 42 (a), media de 0,92 ms.
- Vídeo 2. Figura 42 (b), media de 0,89 ms.
- Vídeo 3. Figura 42 (c), media de 0,95 ms.
- Vídeo 4. Figura 42 (d), media de 0,73 ms.

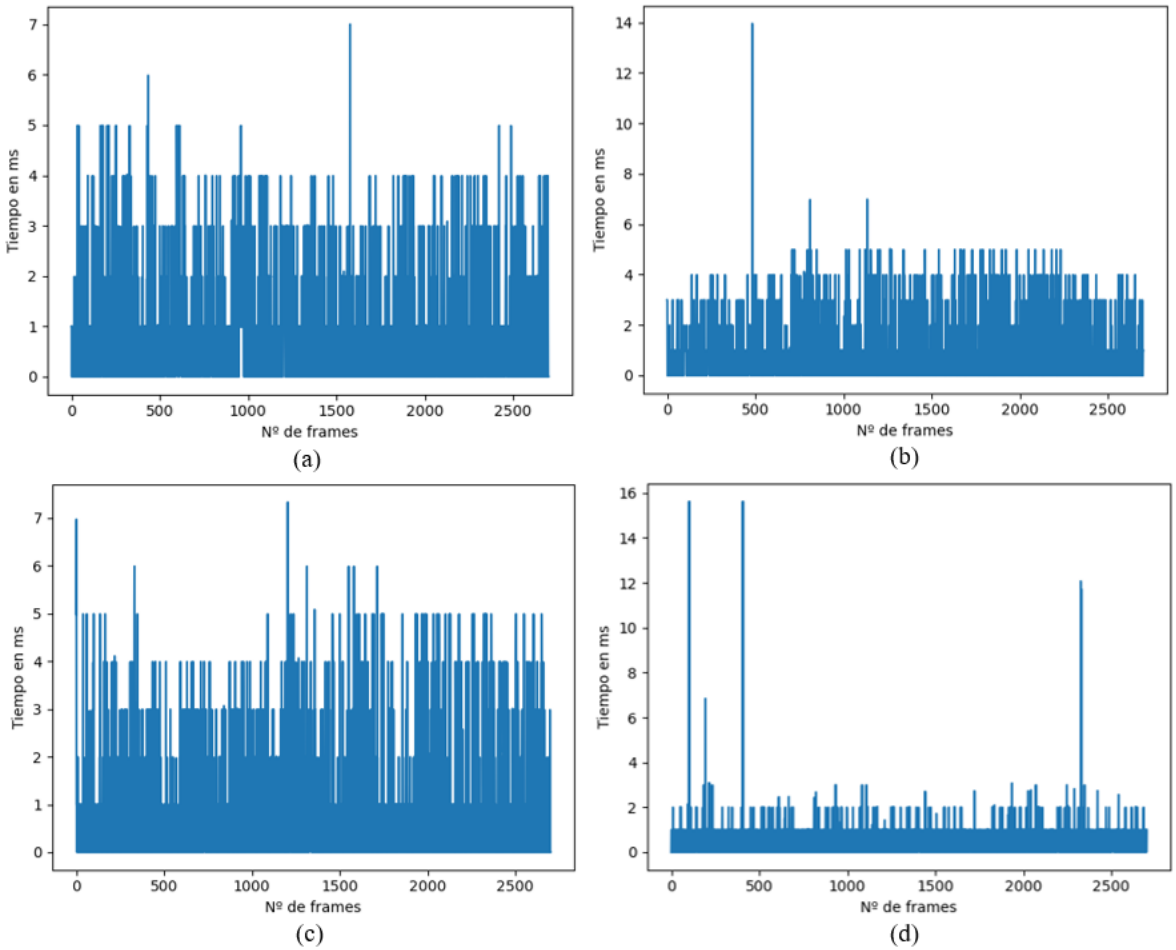

Figura 42. Tiempo de contornos de los objetos en Python

4.2.6 CÁLCULO DE DISTANCIAS Y PELIGRO

La Figura 43 representa los tiempos de ejecución tanto del cálculo de distancias como en decidir si genera la alarma. La media de los tiempos es:

- Vídeo 1. Figura 43 (a), media de 0,03 ms.
- Vídeo 2. Figura 43 (b), media de 0,02 ms.
- Vídeo 3. Figura 43 (c), media de 0,03 ms.
- Vídeo 4. Figura 43 (d), media de 0,03 ms.

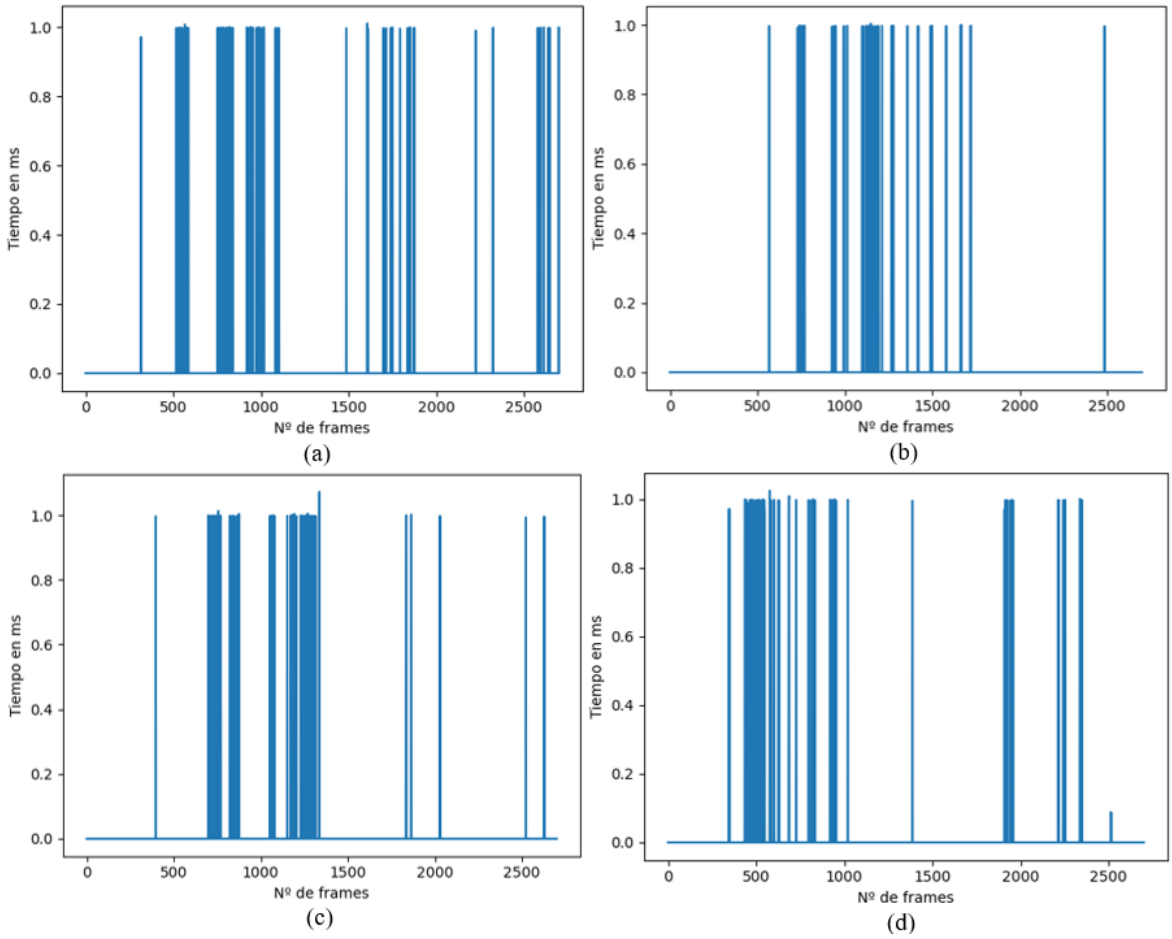

Figura 43. Tiempo cálculo de distancias y alarma en Python

4.3 TIEMPOS EN MATLAB

En Matlab se midieron los tiempos utilizando un método parecido al de Python. Se han utilizado dos variables, las cuales están al principio y al final de lo que se quiere medir, y guardan el tiempo en milisegundos en un vector. En vez de cuatro vídeos, como el código en Matlab ya se probó que funcionaba correctamente en el TFG de Concepción Góngora [7], solo se estudiará el tiempo de procesamiento del primero de ellos. Como también hay atípicos, se utiliza la mediana o la media según proceda.

Para todos los procesos se toma el mismo número de *frames*, los primeros 1300, las medidas de tiempo tienen como máxima resolución de 1 ms. Las unidades y la precisión con la que se presentan las medias de todos los *frames* es en ms y de dos decimales.

El tiempo en procesar todo el código, sin contar con el aprendizaje de fondo del principio (solo se ejecuta una vez), se observa en la Figura 44. La mediana del tiempo de ejecución es de 42,09 ms.

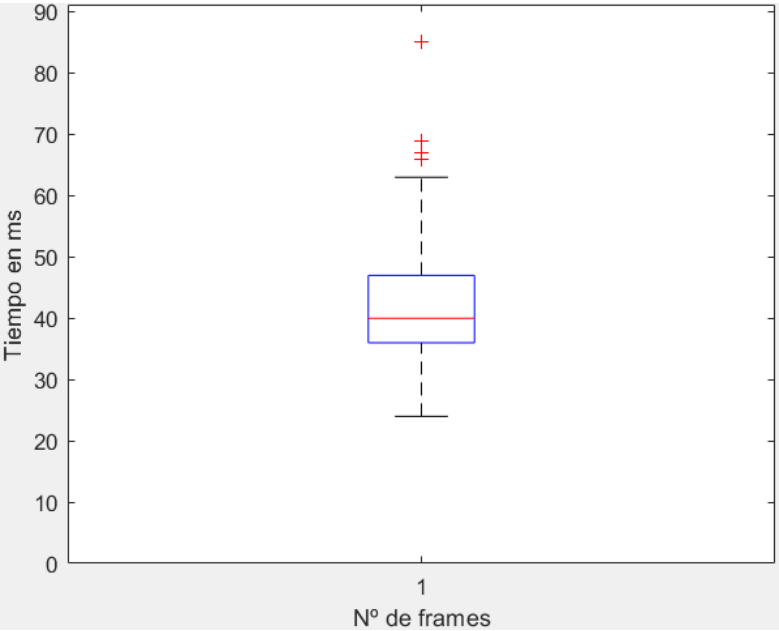

Figura 44. Tiempo total de procesamiento en Matlab

4.3.1 TIEMPO DE SEGMENTACIÓN POR COLOR

La Figura 45 representa el tiempo que tarda en segmentar el color. La media del tiempo es de 3,83 ms.

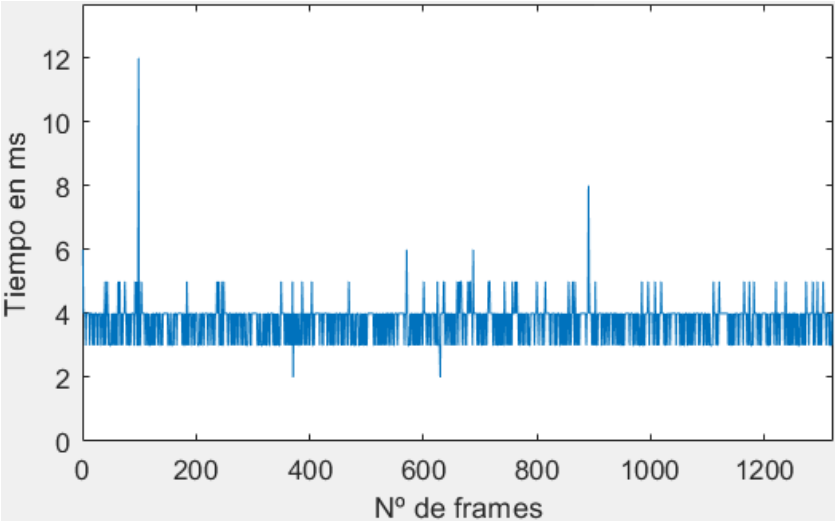

Figura 45. Tiempo de segmentación por color en Matlab

En Matlab, cuando se realiza la sustracción de fondo, el brazo robótico se segmenta directamente en ese paso. Como se han utilizado dos métodos de sustracción de fondo, se segmenta dos veces por color, por lo que el tiempo de este proceso es el doble, tardando 7,66 ms.

4.3.2 SUSTRACCIÓN DE FONDO

En el código de Matlab también se utilizan dos métodos, pero estos son MOG y SSIM. También se han medido por separado.

La Figura 46 representa los tiempos de ejecución del método MOG. El tiempo medio de ejecución es de 7,82 ms.

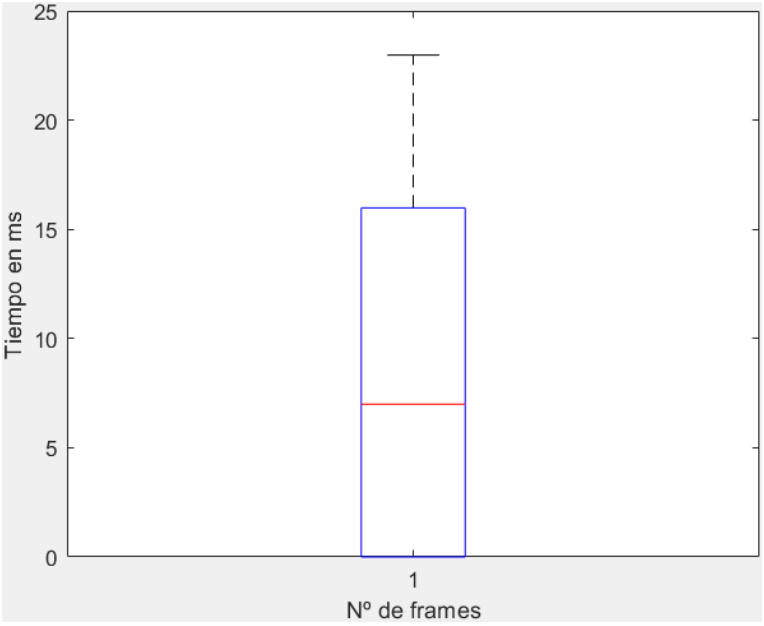

Figura 46. Tiempo de sustracción de fondo MOG en Matlab

La Figura 47 representa los tiempos de ejecución del método SSIM. El tiempo medio de ejecución es de 13,67 ms.

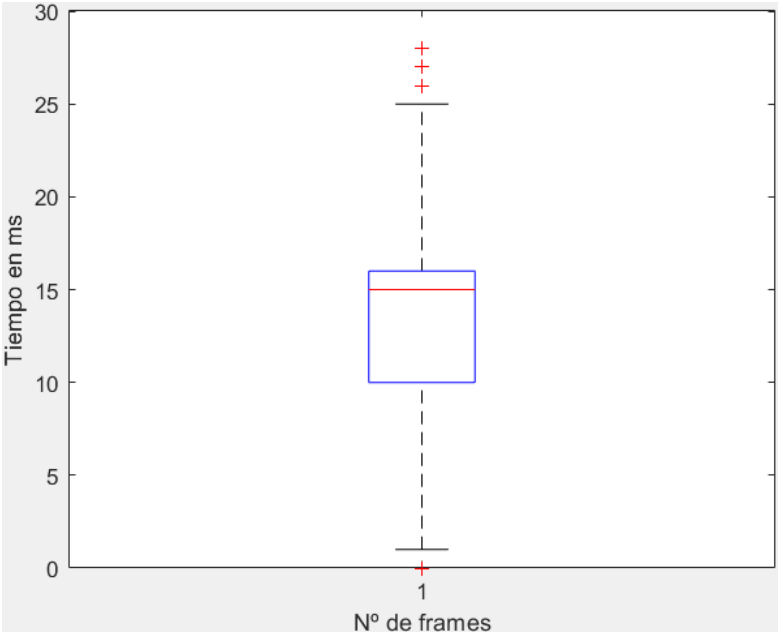

Figura 47. Tiempo de sustracción de fondo SSIM en Matlab

La Figura 48 representa los tiempos de ejecución de unir ambos métodos. El tiempo medio de ejecución es de 0,81 ms.

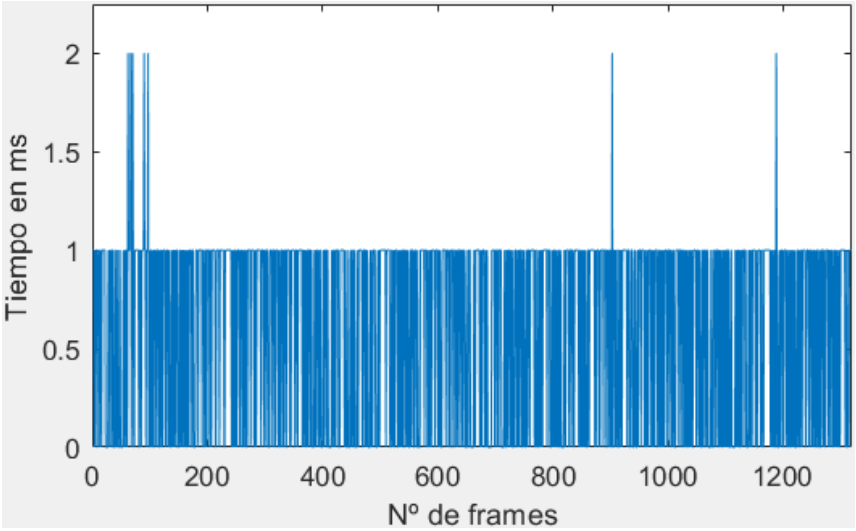

Figura 48. Tiempo de unión de MOG y SSIM en Matlab

4.3.3 ELIMINACIÓN DEL BRAZO DEL ROBOT

La Figura 49 representa el tiempo que tarda en hacer la resta de la máscara de segmentación por color con la máscara de sustracción de fondo. La media de tiempo es de 0,83 ms.

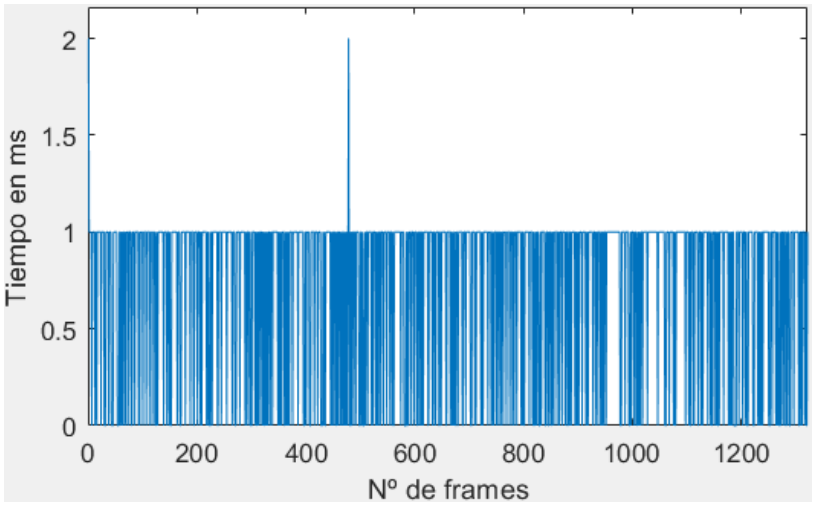

Figura 49. Tiempo de eliminación del brazo robótico en Matlab

Igual que segmenta dos veces, también elimina dos veces por lo que este tiempo también es el doble, siendo 1,66 ms.

4.3.4 TRANSFORMACIÓN MORFOLÓGICA

En Matlab se aplica la transformación morfológica, tanto en la sustracción de fondo SSIM, como en la segmentación de color (esta última dos veces), por lo que se repite tres veces el mismo proceso.

La Figura 50 representa los tiempos de ejecución de la transformación morfológica de la sustracción de fondo SSIM, la cual se asemeja a las otras dos veces que se ejecuta. La media de los tiempos es de 3,67 ms.

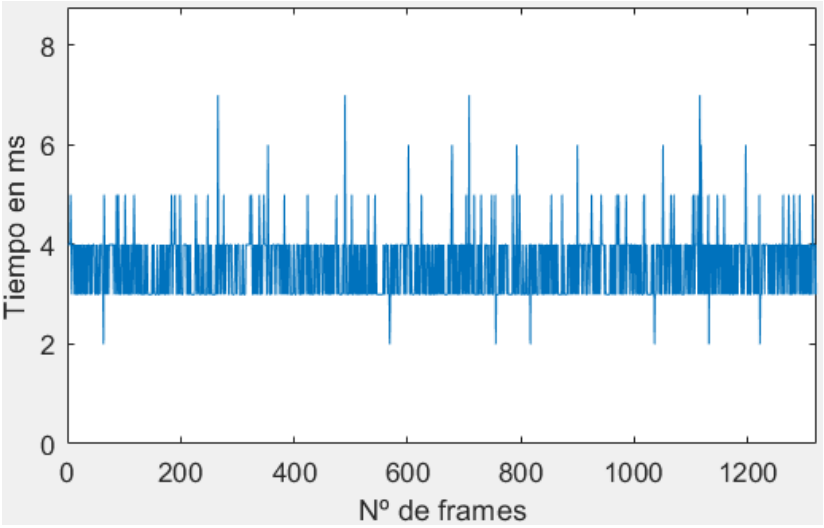

Figura 50. Tiempo de transformación morfológica en Matlab

Como se ejecuta otras tres veces el tiempo total es de, 11,01 ms.

4.3.5 CREACIÓN DE BLOBS

Las funciones que se han utilizado en Matlab son los “blobs” en vez de “Contour Features”. A esta función se ejecuta tres veces, dos en segmentación por color y una para los objetos.

La Figura 51 representa los tiempos de ejecución de los contornos de segmentación por color. La media es de 0,42ms. El tiempo total es de 1,26 ms.

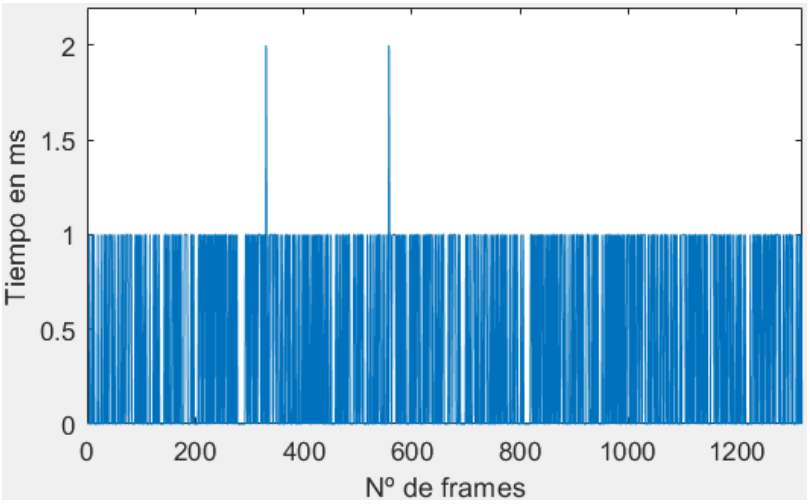

Figura 51. Tiempo de blobs en Matlab

4.3.6 CÁLCULO DE DISTANCIAS Y PELIGRO

La Figura 52 representa los tiempos de ejecución del cálculo de distancias y de decisión para generar la alarma. La media de los tiempos es 0,20 ms.

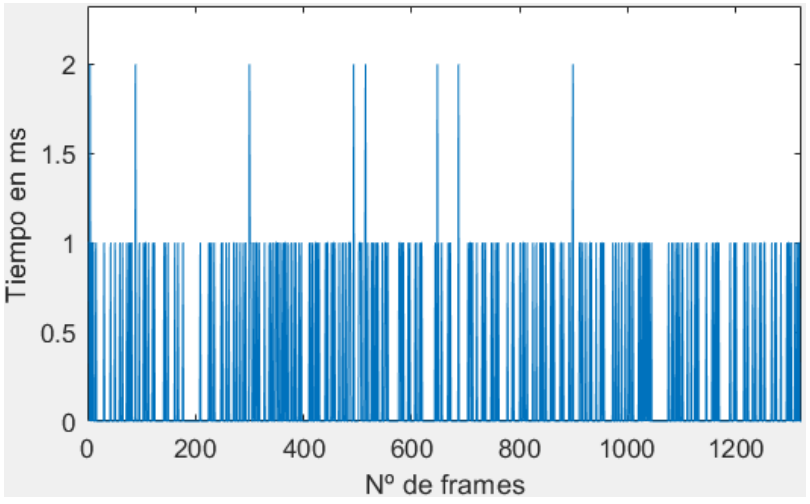

Figura 52. Tiempo cálculo de distancias y alarma en Matlab

4.4 RESULTADOS DE PYTHON Y MATLAB

Con los resultados de las gráficas anteriores ya se pueden discernir algunas conclusiones, pero para verlas más claras, están colocadas en tablas con sus debidos porcentajes. Las medidas de los cuatro vídeos de Python se han resumido en una tabla haciendo la media de cada uno de los apartados.

4.4.1 RESULTADOS DE PYTHON

Todos los resultados del Capítulo 4.2 están resumidos en la Tabla 1.

	Tiempo (ms)	Porcentaje (%)
Total	14,85	100,00
Segmentación por color	0,18	1,21
Sustracción de fondo	12,07	81,27
Eliminación del brazo	0,13	0,87
Transformación morfológica	1,52	10,23
Contornos	1,84	12,39
Cálculo de distancias y peligro	0,03	0,20

Tabla 1. Tiempos de ejecución de Python

Lo primero que llama la atención es el tiempo que tarda en ejecutar la sustracción de fondo, el cual es un 81,27% del total. Si se analiza más en detalle la sustracción de fondo, se tiene que: el método MOG2 son 2,13 ms, el SSIM son 9,81 ms y la unión de las máscaras solo un 0,13 ms. Estos valores son la media de los valores indicados para cada caso en el Capítulo 4.2.2. El método SSIM es el que más tarda, por lo que cabe la posibilidad de cambiar éste por otro método de sustracción de fondo más eficiente, o procesar las sombras y los brillos de otra manera.

Otro dato interesante es que el cálculo de distancias lo ejecuta bastante rápido (solo el 0,20% del total), por lo que se puede implementar un algoritmo más complejo para calcular las distancias entre el brazo robótico y los objetos.

Tanto las transformaciones morfológicas como los contornos tienen un tiempo de ejecución razonable, y tanto la transformación morfológica como los contornos se aplican a la máscara

segmentada por color como a los objetos sin el brazo robótico. Sin embargo, si se observa el tiempo de cada uno de ellos, es distinto. La transformación morfológica tarda 1,01 ms en segmentación por color, y 0,51 ms en los objetos, lo que es la mitad de tiempo. Esto puede ser debido a que en segmentación por color tiene una operación más. Los contornos tardan lo mismo en ambos, 0,97 ms y 0,87 ms respectivamente.

4.4.2 RESULTADOS DE MATLAB

Todos los resultados del Capítulo 4.3 están resumidos en la Tabla 2.

	Tiempo (ms)	Tiempos parciales de cada proceso (%)
Total	42,09	100,00
Segmentación por color	7,66	18,20
Sustracción de fondo	22,30	52,98
Eliminación del brazo	0,83	1,97
Transformación morfológica	11,01	26,16
Creación de blobs	1,26	2,99
Cálculo de distancias y peligro	0,20	0,48

Tabla 2. Tiempos de ejecución de Matlab

En Matlab lo que tarda más tiempo en ejecutar es también la sustracción de fondo. Si se analiza con más detalle, igual que en Python, los tiempos de los métodos son, 7,82 ms para el MOG, 13,67 ms para el SSIM y 0,81 ms para la unión de las máscaras. Estos datos se incluyen en el apartado 4.3.2. Otra vez el SSIM vuelve a ser lo que más tarda en procesar el lenguaje de programación. Sin embargo, el método SSIM mantiene más o menos el mismo tiempo que en Python a diferencia de todos los demás tiempos, que son mayores que en Python, por lo que seguramente ambos usen la misma función.

Lo que menos tarda en el código de Matlab es el cálculo de distancias, pudiendo implementarse otro método para calcular las distancias.

Las transformaciones morfológicas tardan bastante tiempo porque están repetidas varias veces en el código de Matlab, lo que lo hace un poco más seguro a la hora de filtrar las máscaras de sustracción de fondo. La creación de blobs tarda bastante menos que las demás

funciones, y el resto de los apartados no comentados tienen un tiempo normal en comparación con los demás.

4.4.3 COMPARACIÓN DE TIEMPOS PYTHON Y MATLAB

Los tiempos, tanto en Python como en Matlab se representan en la Tabla 3. En la tercera columna, "Mejora de Python", se calcula el porcentaje de la diferencia de tiempo entre Python y Matlab en realizar los procesos con respecto al tiempo de Matlab.

	Matlab Tiempo (ms)	Python. Tiempo (ms)	Mejora de Python (%)
Total	42,09	14,85	64,72
Segmentación por color	7,66	0,18	97,65
Sustracción de fondo	21,30	12,07	43,33
Eliminación del brazo	0,83	0,13	84,34
Transformación morfológica	11,01	1,52	89,65
Contornos / Creación de blobs	1,26	1,84	-46,03
Cálculo de distancias y peligro	0,20	0,03	85,00

Tabla 3. Comparación de tiempos entre Matlab y Python

Como se puede observar en la Tabla 3, Python es más rápido que Matlab en todos los apartados, excepto en reconocer cuáles son los límites de los objetos y el brazo robótico, pero esta desventaja es despreciable con respecto a todas las demás ventajas. La diferencia total de tiempos es de 27,24 ms. Matlab puede procesar vídeos a 23 fps y Python puede a 67 fps, que para la aplicación en la que se usa, es más que aceptable.

En ambos lenguajes, el cálculo de distancias es rápido, aunque simple, por lo que se puede mejorar. La segmentación por color y la transformación morfológica en Python es mucho más rápida.

No hay tanta diferencia en la sustracción de fondo debido al método SSIM. No obstante, si se compara el método MOG de Matlab, que tarda 7,82 ms, con el método MOG2 de Python, que tarda 2,13 ms, se ejecuta tres veces más rápido, lo cual sí que mejora el tiempo de sustracción de fondo de la imagen.

Cabe destacar que, de todos los procesos, el de segmentación por color es el que más diferencia porcentual hay, la diferencia de tiempo entre Python y Matlab es del 97,65%.

La mayoría de estas mejoras de tiempo es debido a que Matlab ejecuta algunas de las funciones cogiéndolas de la librería OpenCV. Python, realiza esto mismo pero su proceso es más eficiente, siendo casi siempre más óptimo ejecutar funciones que estén en OpenCV desde Python antes que desde Matlab.

4.5 DETECCIÓN SEGÚN LA ILUMINACIÓN

La activación de la alarma cuando el objeto está demasiado cerca del brazo robótico depende de que el sistema de seguridad detecte bien o no el objeto. Si se comparan los tiempos del procesamiento total de la Figura 33 se observa que los tres primeros vídeos tienen tiempos parecidos, mientras que el cuarto tiene menos tiempo. Esto se puede relacionar con la detección de objetos ya que a la hora de generar la alarma de seguridad el cuarto vídeo es el que más problemas da, mientras que los otros tres generan la alarma cuando deben hacerlo.

En la Figura 53 está el mismo objeto (persona) que los tres primeros vídeos detectan, pero el cuarto no. El cuadrado verde representa el brazo robótico y el cuadrado rojo los objetos en movimiento. Los vídeos 1, 2, 3 y 4 son (a), (b), (c) y (d) respectivamente.

Esto puede ser debido a que, debido a las condiciones lumínicas, la sustracción de fondo no detecta los objetos en movimiento. Puesto que no tiene que calcular contornos ni distancias, tarda menos en procesar el *frame*. La sustracción de fondo trabaja sobre la imagen de fondo, por lo que el que los objetos puedan estar borrosos no tendría por qué afectar a su correcto funcionamiento.

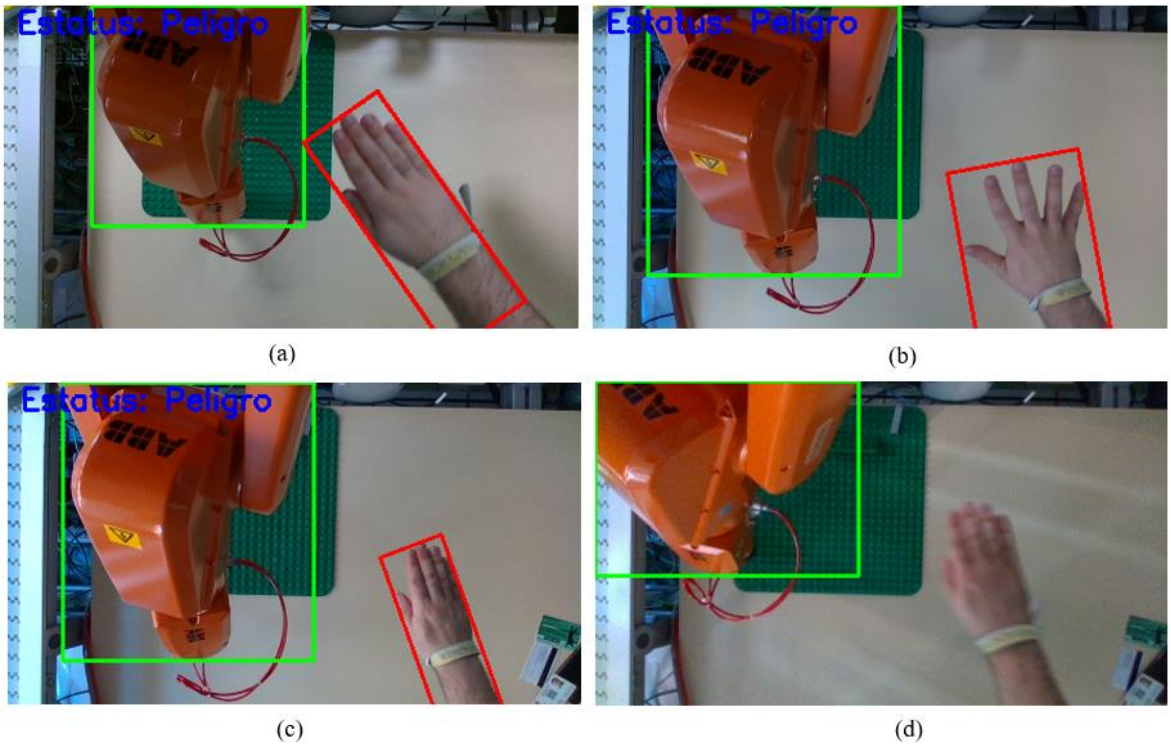

Figura 53. Detección de persona con distintas iluminaciones

Capítulo 5. CONCLUSIONES Y TRABAJOS FUTUROS

En este trabajo fin de grado se ha mejorado el sistema de seguridad para detener un brazo robótico en caso de que una persona u objeto entre en su área de trabajo.

Por los resultados obtenidos se pueden extraer las siguientes conclusiones.

- **Python frente a Matlab.** Se ha conseguido probar que el sistema de seguridad se ejecuta más rápido en Python que en Matlab, obteniendo los mismos resultados de alarma en caso de que un objeto entre en el área de trabajo. El sistema de seguridad enviará las señales de alerta más rápido al brazo robótico en Python, haciendo el sistema más eficaz y seguro.
- **Económico.** Como se ha probado que, además de ser más rápido ejecutando el programa, también detecta los objetos cuando entran en el área de trabajo, cumple los requisitos de seguridad y por tanto es posible su implementación en las industrias, en lugar de Matlab. La inversión en Python es más económica puesto que el programa Python sobre el que se ha hecho la programación no requiere licencia, su aprendizaje es rápido y está muy extendido.
- **Iluminación.** Se ha estudiado la influencia de la iluminación en la respuesta con Python, detectándose que cuando hay luz natural indirecta sin iluminación artificial no detecta bien los objetos de su entorno. Sin embargo, dado que, en las industrias, los brazos robóticos están instalados dentro de naves industriales con luz artificial y potencia suficiente, no es un problema la iluminación ya que es casi siempre la misma.
- **Límite de *frames* por segundo (FPS).** Se pueden desarrollar e implementar métodos más complicados hasta cierto límite, 60 fps (16 ms), ya que esa es la velocidad máxima a la que pasa los *frames* la cámara al sistema de seguridad. No tiene sentido bajar de los 16 ms si a cambio se pierde precisión de los procesos ya que ese es el límite.

Vistos y analizados los resultados se puede concluir que la velocidad de procesamiento de imágenes usando el lenguaje Python son mayores que las de Matlab. Esta mayor velocidad hace que personas u objetos que se acerquen al robot sean detectados con más antelación. Esta mejora permite que el sistema de seguridad por control de velocidad y separación del robot industrial (IRB 120) instalado en el laboratorio de automatización de Comillas ICAI funcione de manera más segura.

No obstante, como se indicó en el trabajo, se ha de seguir avanzando en la respuesta de los robots a movimientos en entornos colaborativos. Por limitaciones de tiempo no ha sido posible realizarlas en este proyecto. En concreto, para futuros trabajos se propone lo siguiente.

- **Lenguaje de programación C++.** Para aumentar aún más la velocidad de procesamiento, como se ha comprobado que Python es más rápido que Matlab, también lo puede ser C++ con respecto a Python. Tal y como se avanzó en el Capítulo 2 de estado de la técnica, este lenguaje de programación es más complejo, pero es el más rápido de los tres.
- **Cambio de procesos.** Se pueden implementar procesos o métodos más complejos. En vez de realizar el método de segmentación por color para reconocer el brazo robótico, utilizar un algoritmo de aprendizaje CNN (redes neuronales convolucionales) para realizar la segmentación del brazo de forma más eficiente. También se puede utilizar otro método en vez del SSIM para las sombras debido a su alto tiempo de procesamiento.
- **Incluir la dimensión profundidad.** El sistema actual mide las distancias en un solo plano, por lo que no diferencia si el objeto está a una determinada altura. Esto ocasiona errores en el cálculo de distancias. Como la cámara de INTEL D435 proporciona una estimación de la profundidad se puede diseñar otro sistema de seguridad que sí la considere.

Capítulo 6. BIBLIOGRAFÍA

[1] IGAPE. Estado del Arte de Automatización y Robótica. Oportunidades Industria 4.0 en Galicia. http://www.igape.es/es/ser-mas-competitivo/galiciaindustria4-0/estudos-e-informes/item/1529-oportunidades-industria-4-0-en-galicia. "Última visita 30/08/2020".

[2] Unimate. Joseph Engelberger. https://www.robotics.org/joseph-engelberger/unimate.cfm. "Última visita:21/01/2020"

[3] Norman J. The first industrial robot. https://www.historyofinformation.com/detail.php?id=3616. "Última visita 30/08/2020"

[4] ISO Standard 8373:1994 Robots and robotic devices - Vocabulary

[5] Estimación robots en 2020. IFR. https://ifr.org/news/ifr-forecast-1.7-million-new-robots-to-transform-the-worlds-factories-by-20/. "Última visita: 21/01/2020"

[6] Número de trabajadores y robots. Crónica. https://cronicaglobal.elespanol.com/graficnews/espana-160-robots-10-000-trabajadores_140204_102.html. "Última visita:21/01/2020"

[7] Concepción Góngora Luque. Aplicación de la visión artificial a la seguridad de un robot. Universidad Pontificia de Comillas ICAI, Madrid, July 2019.

[8] IRB 120. ABB. ABB's 6 axis robot – for flexible and compact production. https://new.abb.com/products/robotics/industrial-robots/irb-120. "Última visita:21/01/2020"

[9] IRB 120T. ABB. ABB Robotics Introduces the New IRB 120T. https://www.robotics.org/content-detail.cfm/Industrial-Robotics-News/ABB-Robotics-Introduces-the-New-IRB-120T/content_id/3494. "Última visita:21/01/2020"

[10] Cade cobots. Cobots: los nuevos robots con inteligencia artificial https://cadecobots.com/cobots-robots-con-inteligencia-artificial/. "Última visita: 21/01/2020"

[11] Robótica Blogspot http://robotiica.blogspot.com/2007/10/historia-de-la-robtica.html "Última visita 07/09/2020"

[12] Tabuenca D. TFG Implantación de robots colaborativos en Línea de producción. Escuela de Ingenierías Industriales. Universidad de Valladolid Enero 2017. http://uvadoc.uva.es/bitstream/handle/10324/23076/TFG-584.pdf?sequence=1&isAllowed "Última visita 07/09/2020

[13] Alpe industrial https://www.alpeautomatizar.com/5-fabricantes-de-robots-colaborativos/ "Última visita 07/09/2020

[14] Directiva 2006/42/CE del parlamento Europeo y del Consejo de 17 de mayo de 2006 relativa a las máquinas por la que se modifica la Directiva 95/16/CE (refundición). DOUE 09.06.2006

[15] ISO 10218-1, ISO 10218-1:2011. Robots and robotic devices. Safety requirements for industrial robots. Part 1: Robots. 2011, International Organization for Standardization.

[16] ISO 10218-2, ISO 10218-2:2011. Robots and robotic devices. Safety requirements for industrial robots. Part 2: Robot systems and integration. 2011, International Organization for Standardization.

[17] ISO/TS 15066, ISO/TS 15066 Robots and robotic devices - Collaborative robots. 2016, International Organization for Standardization.

[18] TecnoPLC. http://www.tecnoplc.com/seguridad-de-cobots/. "Última visita 07/09/2020"

[19] Económica Cantabria. Crónica. https://www.cantabriaeconomica.com/a-fondo/los-pequenos-robots-conquistan-las-fabricas-cantabras/. "Última visita:02/09/2020"

[20] TM Series. Omron. http://www.ia.omron.com/products/family/3739/. "Última visita:21/01/2020".

[21] HC 10 Series. Yaskawa https://www.yaskawa.eu.com/en/products/robotics/motoman-robots/seriesdetail/serie/hc10/. "Última visita:21/01/2020"

[22] Robot Vision. Aleš Ude. https://books.google.es/books?hl=en&lr=&id=xAmhDwAAQBAJ&oi=fnd&pg=P

A429&dq=industrial+robot+collaborative+artificial+vision&ots=uqAvEiH__6&sig=14D_lsJbsmm0yDz54RO3coF6Ino#v=onepage&q=industrial%20robot%20collaborative%20artificial%20vision&f=false. "Última visita:21/01/2020"

[23] Prevención Integral. https://www.prevencionintegral.com/canal-orp/papers/orp-2019/normativa-robots-colaborativos-cobots-su-influencia-en-prevencion-riesgos-laborales. "Última visita 07/09/2020"

[24] Zhmud V.A., Kondratiev N.O., Kuznetsov K.A., Trubin V.G., Dimitrov L.V. Application of ultrasonic sensor for measuring distances in robotics. IOP Conf. Series;(1015)032189 Journal of Physics. Int. Conf. Information Technologies in Business and Industry. 2018.

[25] Sensores ultrasónicos. Pepperl+Fuchs. https://www.pepperl-fuchs.com/global/en/classid_182.htm. "Última visita: 31/01/2020"

[26] Gao, HB, Cheng, B, Wang, JQ, et al. Object classification using CNN-based fusion of vision and LIDAR in autonomous vehicle environment. IEEE T Ind Inform 2018; 14(9): 4224–4231.

[27] Sensor LIDAR. Velodynelidar https://velodynelidar.com/. "Última visita: 22/01/2020"

[28] Camera D435. Intel Realsense. https://www.intelrealsense.com/depth-camera-d435/. "Última visita: 22/01/2020"

[29] Kamal Sehairi, Fatima Chouireb, Jean Meunier, "Comparative study of motion detection methods for vídeo surveillance systems," J. Electron. Imaging 26(2), 023025 (2017)

[30] Von Helmholtz's. "Treatise on physiological optics". Translated from the 3rd German Edition. Vol.I James. Southall, JPC Dover Publications. 1925

[31] Gonzalo Silva Blanco, Francisco Avecilla Rangel, Alejandro Rivas Araiza, Manuel Toledano Ayala, Jesús Pedraza Ortega, and Manuel Ramos Arreguín. Desarrollo de un Sistema de Detección de Movimiento basado en Flujo Óptico en Raspberry Pi. *La Mecatrónica en México*, vol.4(No. 2), May2015.

[32] Aires K.R.T., A.M. Santana y A.D. Medeiros (2008). Optical Flow Using Color Information. ACM New York, NY, USA.

[33] Patent: US7502515. March 2009

[34] Sanahuja G., Valera A., Sánchez A.J., Ricolfe-Viala C., Vallés M., Marín L. Control Embebido de Robots Móviles con Recursos limitados basados en Flujo Óptico. Revista Iberoamericana de automática e informática industrial 8 (2011) pp 250-257.

[35] Tresanchez M. Tesis doctoral "Aplicación de sensores de flujo óptico para el desarrollo de nuevos sistemas de medida de bajo coste". Informática e Ingeniería Industrial. Universitat de Lleida. Octubre 2011.

[36] R. C. Nelson and J. Aloimonos. Obstacle avoidance using flow field divergence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(10):1102–1106, Oct1989.

[37] J. Zheng et al., "Extracting roadway background image: Mode-based approach," Transportation Research Record: Journal of the Transportation Research Board (1944), 82-88 (2006).

[38] S. Vahora, C. Narendea and P. Nilesh, "A Robuts Method for Moving Object Detection Using Modified Statistical Mean Method," International Journal of Advanced Information Technology (IJAIT), vol. 2, no. 1, p. 65, 2012.

[39] Solehah Mohd Radzi S., Nizam Yaakob S. Extraction of moving objects using *frame* differencing, Ghost and Shadow removal. 2014. 5th International Conference on Intelligent systems, modelling and simulation. 2166-0662/14 IEEE

[40] D. P. Bertsekas, A. Nedich and V. S. Borkar, "Improved Temporal Difference Methods with Linear Function Approximation," in Learning and Approximate Dynamic Programming, Wiley-IEEE Press, 2004, pp. 231-235.

[41] Santoyo J.E. Tesis doctoral "Sustracción de fondo basado en movimiento". Centro de investigación en Matemáticas A.C. Guanajuato. Julio 2008

[42] Collins, R., Lipton, A. & Kanade, T. (1999). A system for Vídeo surveillance and monitoring. American Nuclear Soc. 8th Int. Topical Meeting on Robotics and Remote Systems.

[43] Loaiza Quintana, Andrés Felipe, Manzano Herrera, David Andrés, Múnera Salazar, Luis Eduardo. Sistema de visión artificial para conteo de objetos en movimiento El

Hombre y la Máquina, núm. 40, septiembre-diciembre, 2012, pp. 87-101. Universidad Autónoma de Occidente. Cali, Colombia

[44] Losada J. TFG. Detección de movimiento perimetral basado en modelo de mezcla de Gaussianas (DMP-GAUSS). Escuela de Ingeniería de Telecomunicación. Universidad de Vigo.2015

[45] Zhen Tang, Zhenjiang Miao, Yanli Wan. Background Substraction Using Running Gaussian Average and Frame Difference. Institute of Information Science, Beijing Jiao Tong University. 2006

[46] Mateu O. Análisis y detección de objetos de primer plano en secuencias de vídeo. PFC. Junio 2009

[47] C. R. Wren, A. Azarbayejani, T. Darrell, and A. P. Pentland. Pfinder: real-time tracking of the human body. IEEE Transactions on Pattern Analysis and Machine Intelligence, 19(7):780–785, July1997.

[48] Martin Bichsel. Segmenting simply connected moving objects in a static scene. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 16:1138–1142,121994.

[49] W.E.L Grimson Chris Stauffer. Adaptive background mixture models for real-time tracking. *The Artificial Intelligence Laboratory Massachusetts Institude of Technology*, 1997.

[50] A. Chan, V. Mahadevan, and N. Vasconcelos. Generalized Stauffer Grimson Background Subtraction for Dynamic Scenes. *Statiscal Visual Computing Lab UC San Diego*, 2008.

[51] A. Elgammal, R. Duraiswami, D. Harwood, and L. S. Davis. Background and foreground modeling using nonparametric kernel density estimation for visual surveillance. *Proceedings of the IEEE*, 90(7):1151–1163, July 2002.

[52] Ahmed Elgammal, Ramani Duraiswami, and Larry Davis. Efficient kernel density estimation using the fast gauss transform with applications to color modeling and tracking. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25:1499–1504, 11 2003.

[53] TIOBE Index for August 2020. https://www.tiobe.com/tiobe-index/ Acceso 31 Agosto 2020

[54] Mi primera guía Matlab. http://matlabinformatica.wikidot.com/. "Última visita 30/08/2020"

[55] MathWorks. https://es.mathworks.com/products/matlab.html. "Última visita 30/08/2020"

[56] Wikipedia Python. https://es.wikipedia.org/wiki/Python. "Última visita 29/08/2020"

[57] Piatetsky, G. Python Leads the 11 Top Data Science, Machine Learning Platforms: Trends and Analysis.2019. Available online: https://www.kdnuggets.com/2019/05/poll-top-data-science-machine-learning-platforms.html (accessed on 1 February 2020).

[58] Raschka S., Patterson J., Nolet C. Machine Learning in Python: Main Developments and Technology Trends in Data Science, Machine Learning, and Artificial Intelligence. MDPI Information 2020, 11, 193. April 2020

[59] Fernández M. Python: ejemplos de sus ventajas para la Inteligencia Artificial https://blog.enzymeadvisinggroup.com/python-ejemplos. Última visita 29/08/2020

[60] Ateeq M., Habib H., Umer A. and Rehman M. U., "C++ or Python? Which One to Begin with: A Learner's Perspective," 2014 International Conference on Teaching and Learning in Computing and Engineering, Kuching, 2014, pp. 64-69

[61] Alzahrani N., Vahid F., Edgcomb A., Nguyen K., Lysecky R. Python Versus C++: An Analysis of Student Struggle on Small Coding Exercises in Introductory Programming Courses. : SIGCSE '18: Proceedings of the 49th ACM Technical Symposium on Computer Science Education. February 2018. Pages 86–91.

[62] Kriegman S., Cappelle C., Corucci F., Bernatskiy A., Cheney N., and Bongard J.C.. 2017. Simulating the evolution of soft and rigid-body robots. In Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '17). Association for Computing Machinery, New York, NY, USA, 1117–1120.

[63] Björgvinsson, T. 2006. Peocounter: people counting software. Gothenburg. Chalmers University of Technology

[64] Hardwaresfera. Solé R. "Matlab se actualize para mejorar el rendimiento de los AMD Ryzen en un 300%. https://hardwaresfera.com/noticias/software/matlab-se-

actualiza-para-mejorar-el-rendimiento-de-los-amd-ryzen-en-un-300/. "Última visita 01/09/2020"

[65] Medusa Robotics. Robot arm irb120 controlled by ROS. https://sites.google.com/site/medusarobotics/abb_irb120. "Última visita 01/09/2020"

[66] LinuxAdictos. Python: los lenguajes también pueden ser de código abierto https://www.linuxadictos.com/python-los-lenguajes-de-codigo-abierto.html. "Última visita 01/09/2020"

[67] Intel Real Sense Technology. https://www.intelrealsense.com/developers/ "Última visita 01/09/2020"

[68] IntelRealSense/Librealsense https://github.com/IntelRealSense/librealsense/blob/master/wrappers/python/examples/read_bag_example.py. "Última visita 01/09/2020"

[69] InterRealSense LibrealsenseTM Python Bindings https://intelrealsense.github.io/librealsense/python_docs/_generated/pyrealsense2.html. "Última visita 01/09/2020"

[70] Mathworks. Computer Vision Toolworks https://es.mathworks.com/products/computer-vision.html "Última visita 18/09/2020"

[71] OpenCV. Thresholding Operations using inRange https://docs.OpenCV.org/3.4/da/d97/tutorial_threshold_inRange.html. "Última visita 01/09/2020"

[72] Zivkovic Z. Improved Adaptive Gaussian Mixture Model for Background Subtraction. . In Proceedings of the ICPR 2004.

[73] Zivkovic Z., Van der Heijden F. Efficient adaptive density estimation per image píxel for the task of background subtraction, Pattern Recognition Letters,Volume 27, Issue 7, 2006, Pages 773-780, ISSN 0167-8655.

[74] Marcomini L.A. Cunha A.L. A comparison between Background Modelling Methods for Vehicle Segmentation in Highway Traffic Videos https://arxiv.org/ftp/arxiv/papers/1810/1810.02835.pdf. "Última visita 01/09/2020"

[75] OpenCV. How to Use Background Subtraction Methods https://docs.OpenCV.org/3.4/d1/dc5/tutorial_background_subtraction.html. "Última visita 01/09/2020"

[76] Wang Z. Image Quality Assessment: From Error Visibility to Structural Similarity. IEE Transactions on image processing Vol. 13 No.4 April 2004. http://www.cns.nyu.edu/pub/eero/wang03-reprint.pdf

[77] Pyimagesearch. Rosebrock A. https://www.pyimagesearch.com/2017/06/19/image-difference-with-OpenCV-and-python/. "Última visita 11/09/2020"

[78] IEEE UCSA Student Branch. http://ucsa.ieeeparaguay.org/2019/05/15/transformaciones-morfologicas-con-vision-artificial/. "Última visita 10/09/2020"

[79] OpenCV. Morphological Transformations. https://OpenCV-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html. "Última visita 01/09/2020"

[80] Mallick S. https://www.learnOpenCV.com/blob-detection-using-OpenCV-python-c/. "Última visita 01/09/2020"

[81] OpenCV. Contour Features https://docs.OpenCV.org/3.4/dd/d49/tutorial_py_contour_features.html. "Última visita: 07/09/2020"

[82] Complete Phyton3 Bootcamp. https://www.youtube.com/watch?v=MkcUgPhOlP8. "Última visita: 07/09/2020"

[83] Pc componentes. https://www.pccomponentes.com/hp-omen-15-ax201ns-intel-core-i7-7700hq-8gb-1tb-gtx-1050-156. "Última visita: 07/09/2020"

[84] Naciones Unidas. "Objetivos y metas de desarrollo sostenible- desarrollo sostenible"https://www.un.org/sustainabledevelopment/es. "Última visita: 30/08/2020"

[85] Naciones Unidas. "Objetivos y metas de desarrollo sostenible- crecimiento económico". https://www.un.org/sustainabledevelopment/es/economic-growth/. "Última visita: 30/08/2020"

[86] Naciones Unidas. "Objetivos y metas de desarrollo sostenible- infraestructuras" https://www.un.org/sustainabledevelopment/es/infrastructure/. "Última visita: 30/08/2020"

[87] Naciones Unidas. "Objetivos y metas de desarrollo sostenible- producción para el consumo sostenible" https://www.un.org/sustainabledevelopment/es/sustainable-consumption-production/. "Última visita: 30/08/2020"

[88] AI FOR GOOD United Nations Platform Global Summit 2020. https://aiforgood.itu.int/. "Última visita: 30/08/2020"

ANEXO A. OBJETIVOS DE DESARROLLO SOSTENIBLE DEL PROYECTO (ODS)

Los Objetivos de Desarrollo Sostenible, ODS, también conocidos como Objetivos Mundiales, se adoptaron por todos los Estados Miembros en 2015 como una llamada universal para poner fin a la pobreza, proteger el planeta y garantizar la prosperidad para 2030. Estos objetivos se pueden ver en la Figura 54.

Figura 54. Objetivos de Desarrollo Sostenible [83]

Los 17 ODS están integrados, ya que reconocen que las intervenciones en un área afectarán los resultados de otras y que el desarrollo debe equilibrar la sostenibilidad medio ambiental, económica y social. Para conseguir los ODS es necesaria la colaboración de los gobiernos, el sector privado y los ciudadanos por igual para asegurar un planeta mejor. Por tanto, todo el mundo es necesario para alcanzar estos objetivos ambiciosos. Se necesita la creatividad, el conocimiento, la tecnología y los recursos financieros de toda la sociedad para conseguir los ODS en cada contexto.

Más en concreto la tecnología avanzada, tal como la inteligencia artificial y la digitalización, tiene un gran potencial para acelerar el progreso de los ODS y la solución propuesta en el proyecto presentado, al estar relacionado con la inteligencia artificial y la digitalización

aplicada a la industria y sus aspectos de seguridad puede ayudar a cumplir algunas de las metas que se persiguen en varios de los objetivos ODS.

En la Tabla 4 se concretan estos aspectos:

Ámbito de desarrollo sostenible	Objetivo identificado para el desarrollo sostenible	Meta	Rol	Relación con el proyecto
Económico y Social	Objetivo 8: Crecimiento económico y sostenible [85]	Promover puestos de trabajo seguros.	Primario	Con su implantación se lograría reducir los accidentes producidos en las líneas de producción.
		Creación de empleo de mayor calidad en la industria	Primario	Con el sistema desarrollado se incrementa el empleo de mayor calidad.
		Modernización del tejido productivo	Secundario	La solución propuesta es una parte más de la industria 4.0
Económico	Objetivo 9: Promover la industria eficiente y sostenible a 2030 [86].	Aumentar significativamente la contribución de la industria al empleo y al producto interno bruto.	Primario	La implementación del sistema final permitirá hacer más eficiente producir más con menos recursos.
		Aumentar el acceso de las pequeñas industrias y otras empresas, particularmente en los países en desarrollo	Secundario	La solución propuesta se puede aplicar a las PYMES
Social y Ambiental	Objetivo 12: Producción y consumo responsables [87].	Fortalecer la capacidad científica y tecnológica para avanzar a una producción más sostenibles	Secundario	El sistema final producirá más con lo mismo recursos y será más eficiente energéticamente.

Tabla 4. Tabla de los Objetivos de Desarrollo Sostenible del Proyecto

Para finalizar este punto del trabajo y cabe mencionar la plataforma "AI for Good", de las Naciones Unidas cuyo fin es estudiar e impulsar objetivos de desarrollo sostenible a través de la Inteligencia Artificial a través del intercambio y la colaboración entre todos los agentes [88].