



GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

DESARROLLO DE UN SISTEMA DE SEGURIDAD POR CONTROL DE VELOCIDAD Y SEPARACIÓN DE UN ROBOT INDUSTRIAL BASADO EN LIDAR

Autor: Ana Pascual Rubio

Director: Jaime Boal Martín-Larrauri

José Antonio Rodríguez Mondéjar

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título
Desarrollo de un sistema de seguridad por control de velocidad y separación de un robot
industrial basado en LIDAR

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2020/21 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.: Ana Pascual Rubio Fecha: 05/ 07/ 2021

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO



Firmado digitalmente por BOAL
MARTIN LARRAURI JAIME -
05304600H
Fecha: 2021.07.12 13:23:13
+02'00'

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

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Firmado por RODRIGUEZ MONDEJAR JOSE ANTONIO - 29057313X el día
12/07/2021 con un certificado emitido por AC FNMT Usuarios

Fdo.: José Antonio Rodríguez Mondéjar Fecha://



GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

DESARROLLO DE UN SISTEMA DE SEGURIDAD POR CONTROL DE VELOCIDAD Y SEPARACIÓN DE UN ROBOT INDUSTRIAL BASADO EN LIDAR

Autor: Ana Pascual Rubio

Director: Jaime Boal Martín-Larrauri

Director: José Antonio Rodríguez Mondéjar

Madrid

DESARROLLO DE UN SISTEMA DE SEGURIDAD POR CONTROL DE VELOCIDAD Y SEPARACIÓN DE UN ROBOT INDUSTRIAL BASADO EN LIDAR

Autor: Pascual Rubio, Ana.

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

Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

El presente proyecto desarrolla un sistema de seguridad implementado en un entorno de trabajo industrial para permitir a un brazo robótico compartir su área de trabajo con humanos. Para ello se equipa dicho entorno con una cámara LIDAR, cuyos datos obtenidos son procesador para realizar un seguimiento del robot y del humano, calculando la distancia que hay entre ellos y adaptando en base a esta la velocidad del robot.

Palabras clave: LIDAR, nube de puntos, monitorización de distancia y velocidad

1. Introducción

Se desea garantizar la seguridad en un entorno de trabajo industrial en el cual puedan coincidir la intervención humana con la de un robot automático. En concreto, se trabaja en el laboratorio de automatización de la Universidad Pontificia de Comillas- ICAI- con un brazo robótico de la marca ABB, modelo IRB 120 [1]. Para ello se equipa la instalación con un sensor LIDAR (acrónimo del inglés *Laser Imaging Detection and Ranging*) de la marca Intel RealSense, modelo L515 [2]. Se coloca en estático enfocando el entorno de trabajo en cuestión, de manera que, aunque el robot se mueva, continúe en el campo de visión del sensor, a no ser que salga del entorno compartido con el factor humano.

Con un programa desarrollado en MATLAB, se consigue así monitorizar la entrada de elementos externos- como puede ser la intervención humana (a la cual se hará referencia como obstáculo a lo largo todo del documento)- y su distancia al robot. En función de dicha distancia se adaptará la velocidad del robot: menor velocidad a menor distancia, llegando a producirse la detención del robot en caso de ser necesario.

2. Definición del proyecto

El sistema desarrollado consta de 3 componentes principales, comunicados unilateralmente según explica la Figura 1.



Figura 1. Diagrama de bloques general

El sensor LIDAR, que viene además equipado con una cámara RGB, está constantemente tomando imágenes del entorno para ser tratadas por un programa de

monitorización desarrollado en MATLAB. Este se encarga de determinar la presencia del robot o de un obstáculo y calcular la distancia entre ellos en caso de coincidir. A partir de ella establece a qué velocidad debe ir el robot para que sea seguro el trabajo colaborativo. Se manda entonces al programa encargado de controlar el robot dicho valor de velocidad para que la adapte.

3. Descripción del sistema

El entorno de trabajo en el que se desarrolla el proyecto es el mostrado en la Figura 2. El sensor se encuentra situado de manera fija tal que perciba la zona colaborativa, es decir, donde puedan coincidir robot y humano.

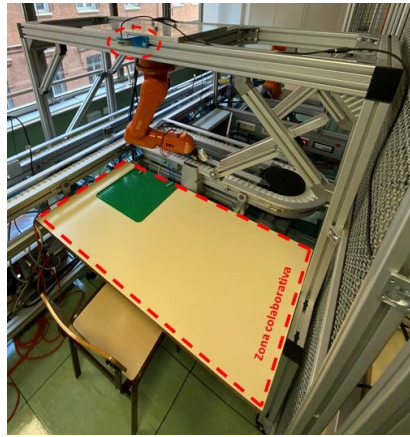


Figura 2. Entorno de trabajo

El LIDAR proporciona una imagen RGB y una nube de puntos del entorno, tratándose esta última de una representación tridimensional, tal como se muestra en la Figura 3 para ayudar a entender el concepto.

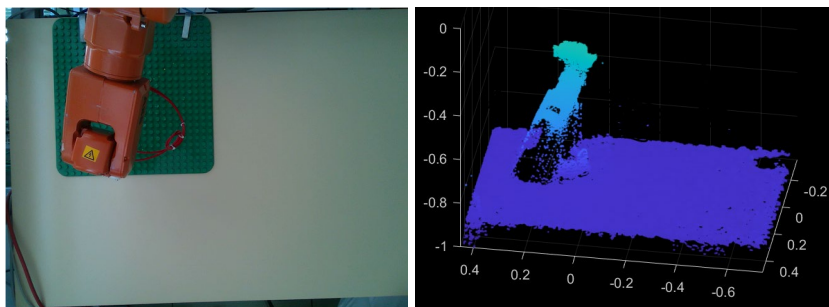


Figura 3. Imagen RGB y nube de puntos asociada

Para que el programa de monitorización sea capaz de identificar los distintos factores que hay en el entorno en cada momento, necesita haber registrado previamente el denominado fondo, esto es, el entorno de trabajo compartido libre de elementos externos, ni robot ni obstáculos (ver Figura 4).

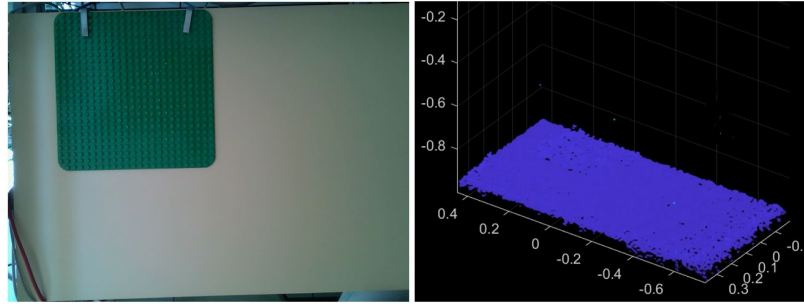


Figura 4. Fondo

Con esto el programa ya puede empezar a procesar las imágenes que le vayan llegando del entorno. Para ello, se establece un bucle por el cual en cada nueva iteración se recoge una nueva imagen, siguiendo la lógica que expone la Figura 5 para concluir el nivel de riesgo del entorno y con ello la velocidad idónea para el robot.

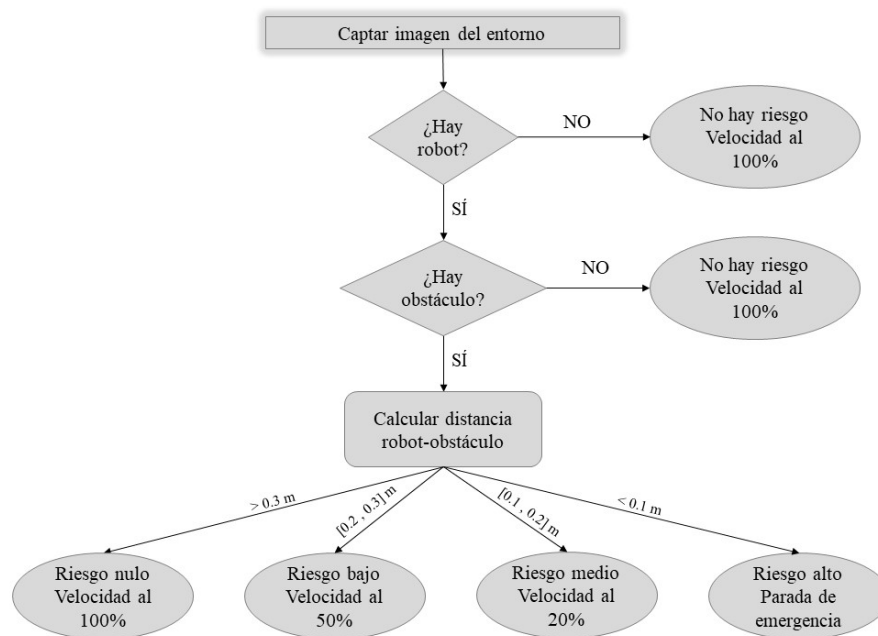


Figura 5. Estructura iterativa del programa de monitorización

Existe un paso previo nada más capturar una imagen del entorno: detectar y seleccionar aquellos puntos de la nueva imagen que han cambiado de sitio con respecto del fondo.

Se procede entonces en primer lugar a buscar la presencia del brazo robótico, lo cual se hace filtrando los puntos para seleccionar aquellos que tengan un color anaranjado, ya que es el color característico del robot. Si no hay suficientes puntos naranjas se considera que no hay robot, en cuyo caso no hay peligro para el humano y el robot puede operar al 100% de su velocidad, ya que se encontrará en una zona de trabajo de uso suyo exclusivo.

En caso de haber robot, se busca la presencia de un obstáculo/humano. Para ello, el conjunto de puntos inicialmente seleccionados como variaciones del fondo son procesador por el algoritmo DBSCAN [3], un algoritmo de agrupación o *clustering* basado en densidad capaz de dividir la nube de puntos en los distintos objetos que la forman (robot, mesa, persona...). El clúster más grande (que no se corresponda con el robot) es seleccionado como posible obstáculo. Si su dimensión no supera un mínimo

establecido no es considerado finalmente un obstáculo, concluyendo de nuevo que no hay riesgo y permitiendo operar al 100% de velocidad.

Sin embargo, si su dimensión es suficientemente grande como para considerarse un obstáculo, con esta nube de puntos asociada al clúster y con la nube de puntos asociada con anterioridad al robot, se calcula la distancia entre ellas, que se corresponde con la distancia en metros entre el robot y la persona. En base a esta distancia se establece el nivel de riesgo de la situación y la velocidad a la que debe ir el robot, según especifica la Tabla 1.

<i>NIVEL DE RIESGO</i>	<i>DISTANCIA [cm]</i>	<i>VELOCIDAD</i>
NULO	Superior a 30	100%
BAJO	Entre 30 y 20	50%
MEDIO	Entre 20 y 10	20%
ALTO	inferior a 10	0% (Parada)

Tabla 1. Niveles de riesgo, distancias y velocidades

Esta velocidad es enviada a un programa encargado de controlar el robot, que lo mueve repetidamente entre los cuatro mismos puntos para simular que se encuentra realizando una tarea.

4. Resultados

A continuación, la Figura 6 representa un ejemplo de cómo el programa de monitorización detecta con éxito y exactitud al robot y a un obstáculo, calculando entre ellos una distancia de 0,16 m en este caso.

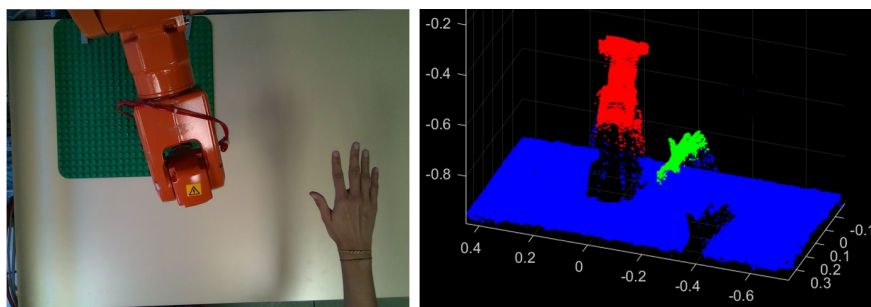


Figura 6. Ejemplo de la segmentación fondo-robot-obstáculo

El problema es que para ello se ha necesitado alrededor de 1 segundo, lo cual es relativamente elevado para un sistema de seguridad, pues se requieren tiempos de respuesta dentro del orden de mini o incluso nano segundos. La Figura 7 recoge en un gráfico los datos relacionados con el tiempo de computación empleado en cada iteración al poner en marcha el sistema.

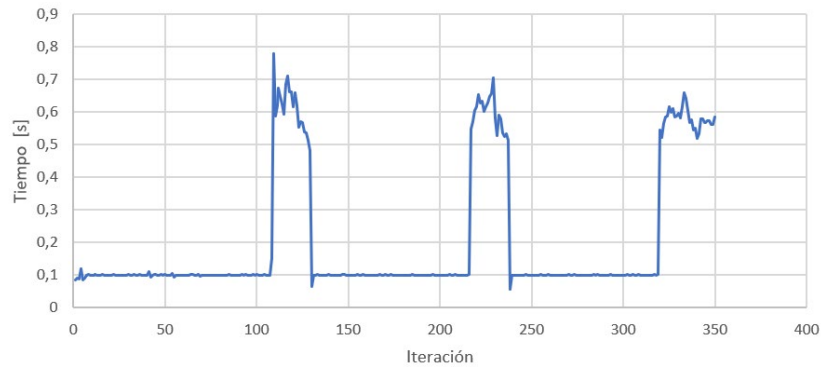


Figura 7. Tiempos de computación por iteración

Se pueden distinguir dos casuísticas: tiempos alrededor de los 0,1 s cuando el área de trabajo se encuentra vacía y tiempos de entre 0,5 y 0,8 s si aparecen robot y/o obstáculo, correspondiéndose los tiempos más altos a la aparición de ambos. Esto se debe a que, como se aprecia en la Figura 5, la iteración finaliza antes o después en función de la casuística.

5. Conclusiones

Se ha implementado el sistema desarrollado en el laboratorio de la universidad poniendo en marcha el robot, ejecutando los programas e introduciendo en varias ocasiones la mano para comprobar la reacción del robot.

Aunque se consigue adaptar la velocidad y, lo más importante, evitar la colisión, no lo hace cumpliendo con las distancias críticas establecidas para cada nivel de riesgo (Tabla 1). Esto se debe a que los elevados tiempos de computación comentados implican una respuesta muy lenta en el robot.

Sin embargo, se ha comprobado que la tarea de segmentación del entorno en fondo, robot y obstáculo la lleva a cabo con gran éxito. Por ello, para conseguir que este sistema pueda ser considerado fiable a la hora de procurar la seguridad, se recomienda un futuro desarrollo en un lenguaje de programación más rápido (Python, Java, C++...), pero manteniendo la estructura del código.

6. Referencias

- [1] «ABB,» [En línea]. Available: <https://new.abb.com/products/robotics/es/robots-industriales/irb-120>. [Último acceso: 29 06 2021].
- [2] «Intel RealSense,» [En línea]. Available: <https://www.intelrealsense.com/lidar/>. [Último acceso: 22 06 2021].
- [3] «Clustering DBSCAN,» *Stat Developer* [En línea] Available: <https://www.statdeveloper.com/clustering-dbscan/>. [Último acceso: 02 07 2021]

DEVELOPMENT OF A SAFETY SYSTEM FOR SPEED AND SEPARATION CONTROL OF AN INDUSTRIAL ROBOT BASED ON LIDAR

Author: Pascual Rubio, Ana.

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

Collaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

This project develops a safety system implemented in an industrial work environment in order to allow a robotic arm to share its work area with humans. To achieve this purpose, the environment is equipped with a LIDAR camera, whose data is processed to track the robot and the human, calculating the distance between them and adapting the robot's speed accordingly.

Keywords: LIDAR, point cloud, speed and separation monitoring

1. Introduction

The aim is to guarantee safety in an industrial work environment in which human intervention can coincide with an automatic robot. Specifically, we are working in the automation laboratory of the Universidad Pontificia de Comillas- ICAI- with an ABB robotic arm, model IRB 120 [1]. For this purpose, the facility is equipped with a LIDAR sensor (Laser Imaging Detection and Ranging) of the Intel REALSENSE brand, model L515 [2]. It is placed static focusing on the work environment in question, so that, even if the robot moves, it remains in the field of view of the sensor, unless it leaves the environment's area shared with the human factor.

With a program developed in MATLAB, it is thus possible to monitor the entry of external elements - such as human intervention (which will be referred to as an obstacle throughout the document) - and their distance from the robot. Depending on this distance, the robot's speed will be adapted: the shorter the distance, the lower the speed and, if necessary, the robot will stop.

2. Project's definition

The developed system consists of 3 main components, communicated unilaterally as shown in Figure 1.

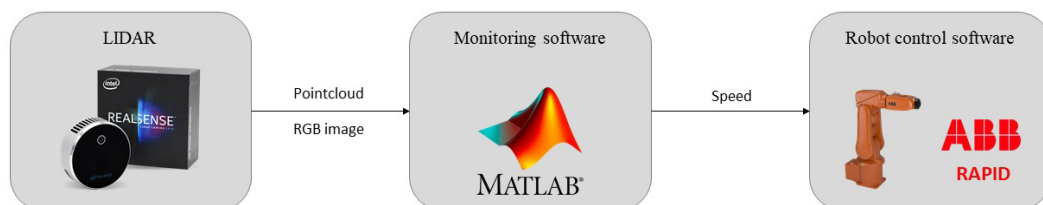


Figure 1. General block diagram

The LIDAR sensor, which is also equipped with an RGB camera, is constantly taking images of the environment to be processed by a monitoring program developed in MATLAB. This is responsible for determining the presence of the robot or an obstacle

and calculating the distance between them if they coincide. Based on this, it establishes the speed at which the robot must go to ensure safety in a collaborative work. This speed value is then sent to the program in charge of controlling the robot so that it can adapt it.

3. Description of the model/system/tool

The working environment in which the project is developed is shown in Figure 2. The sensor is located in a fixed position so that it perceives the collaborative zone, i.e., where robot and human can coincide.



Figure 2. Working environment

The LIDAR provides an RGB image and a point cloud of the environment, which is a three-dimensional representation, as shown in Figure 3 to help understand the concept.

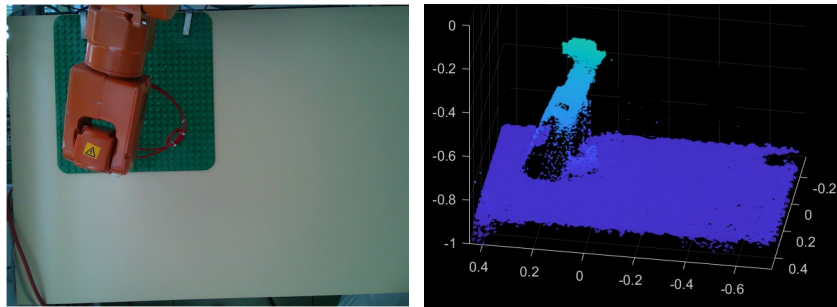


Figure 3. RGB image and point cloud associated

In order for the monitoring program to be able to identify the different factors in the environment at any given moment, it needs to have previously recorded the so-called background, i.e., the shared working environment free of external elements, robots or obstacles (see Figure 4).

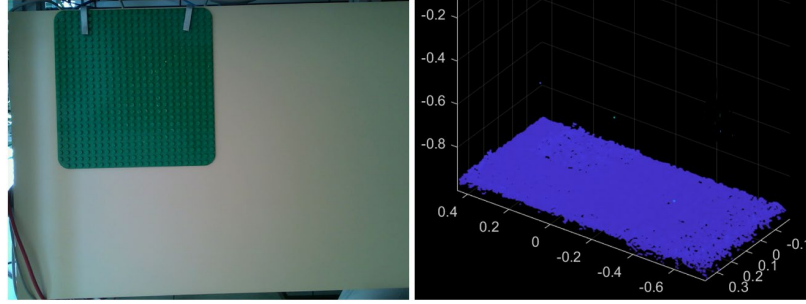


Figure 4. Background

With this, the program can now start processing the images coming from the environment. To do this, a loop is established whereby in each new iteration a new image is collected, following the logic shown in Figure 5 to conclude the level of risk of the environment and thus the ideal speed for the robot.

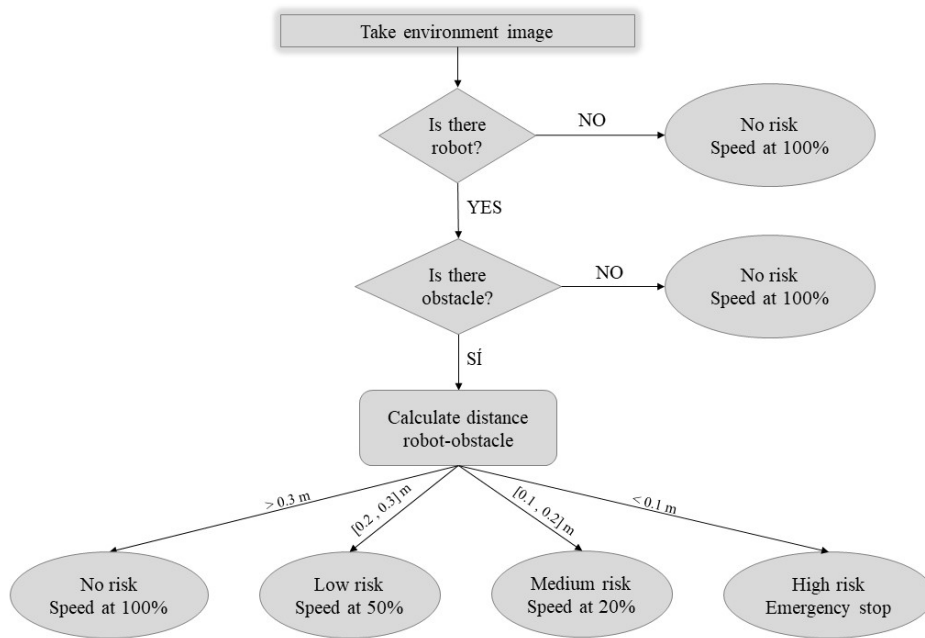


Figure 5. Iterative structure of the monitoring program

There is a preliminary step just after capturing an image of the environment: detecting and selecting those points of the new image that have changed location with respect to the background.

The first step now is to search for the presence of the robotic arm, which will be done by filtering the points to select those with an orange color, since this is the characteristic color of the robot. If there are not enough orange points, it is considered that there is no robot, in which case, there is no danger for the human and the robot can operate at 100% of its speed, since it will be in a work area for its exclusive use.

If there is a robot, the presence of an obstacle/human is searched for. For this, the set of points initially selected as variations of the background are processed by the DBSCAN [3] algorithm, a density-based clustering algorithm capable of dividing the point cloud into the different objects that form it (robot, table, person...). The largest cluster (and not corresponded to the robot) is selected as a possible obstacle. If its dimension does not

exceed a set minimum it is not finally considered as an obstacle, concluding again that there is no risk and allowing to operate at 100% speed.

However, if its dimension is large enough to be considered an obstacle, with this point cloud associated to the cluster and with the point cloud previously associated to the robot, the distance between them is calculated, which corresponds to the distance in meters between the robot and the person. Based on this distance, the risk level of the situation and the speed at which the robot should go are established, as specified in Table 1.

<i>RISK LEVEL</i>	<i>DISTANCE [cm]</i>	<i>SPEED</i>
NON	Greater than 30	100%
LOW	Between 30 and 20	50%
MEDIUM	Between 20 and 10	20%
HIGH	Lower than 10	0% (Stop)

Table 1. Risk levels, distances and speeds

This speed is sent to a program in charge of controlling the robot, which moves it repeatedly between the same four points to simulate that it is performing a task.

4. Results

Figure 6 below shows an example of how the monitoring program successfully and accurately detects the robot and an obstacle, calculating a distance of 0.16 m between them in this case.

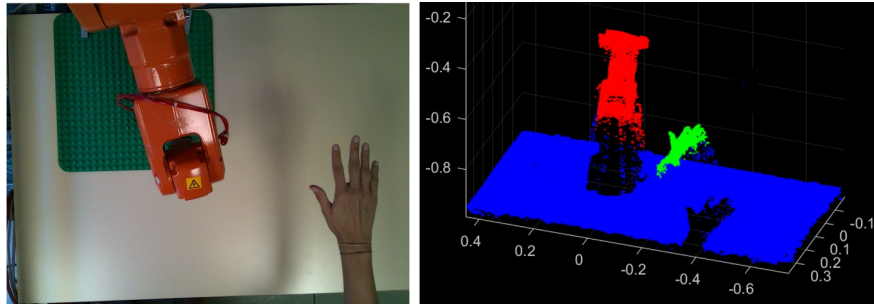


Figure 6. Example of background-robot-obstacle segmentation

The problem is that this took about 1 second, which is relatively high for a security system since response times are required in the order of mini or even nano seconds. Figure 7 shows in a graph the data related to the computation time used in each iteration when starting up the system.

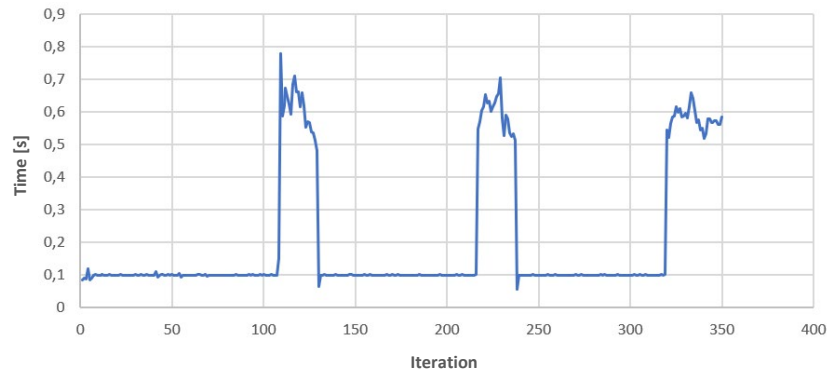


Figure 7. Computing time per iteration

Two cases can be distinguished: times around 0.1 s when the work area is empty and times between 0.5 and 0.8 s whenever robot and/or obstacle appear, corresponding the higher times to the appearance of both. This is due to the fact that, as shown in Figure 5, the iteration ends earlier or later depending on the case.

5. Conclusions

The system developed has been implemented in the university laboratory by starting the robot, running the programs and inserting the hand several times to check the robot's reaction.

Although it manages to adapt the speed and, most importantly, avoid collision, it does not do so while complying with the critical distances established for each risk level (Table 1). This is due to the high computational times mentioned above, which imply a very slow response in the robot.

However, it has been proven that the task of segmenting the environment into background, robot and obstacle is carried out with great success. Therefore, in order to ensure that this system can be considered reliable in terms of safety, future development in a faster programming language (Python, Java, C++...) is recommended, while maintaining the structure of the code.

6. References

- [1] «ABB,» [Online]. Available: <https://new.abb.com/products/robotics/es/robots-industriales/irb-120>. [Last access: 29 06 2021].
- [2] «Intel RealSense,» [Online]. Available: <https://www.intelrealsense.com/lidar/>. [Last access: 22 06 2021].
- [3] «Clustering DBSCAN,» *Stat Developer* [Online] Available: <https://www.statdeveloper.com/clustering-dbscan/>. [Last access: 02 07 2021]

Índice de la memoria

Capítulo 1. Introducción	1
Capítulo 2. Descripción de las Tecnologías.....	3
2.1 LIDAR.....	3
2.2 Brazo robótico	5
Capítulo 3. Estado de la Cuestión.....	9
3.1 Colaboración Humano-Robot (HRC).....	9
3.1.1 Parada segura monitorizada (SMS)	10
3.1.2 Guiado manual (HG).....	10
3.1.3 Control de velocidad y separación (SSM)	11
3.1.4 Limitación de fuerza y potencia (PFL).....	12
3.2 Sensores para percibir el entorno	12
3.2.1 Sensores de fuerza	13
3.2.2 Sensores de ultrasonido.....	13
3.2.3 Sensores fotoeléctricos	14
3.2.4 Sensores de imagen	15
3.2.5 Sensores láser.....	15
3.3 Procesamiento de datos	15
3.3.1 YOLO (You Only Look Once).....	16
3.3.2 PointNet [22].....	18
3.3.3 Algoritmo K-Means	19
3.3.4 Algoritmo DBSCAN.....	20
3.4 Aplicaciones en la industria	22
Capítulo 4. Definición del Trabajo	25
4.1 Justificación.....	25
4.2 Objetivos	26
4.2.1 Reconocimiento del entorno	26
4.2.2 Control de velocidad	27
4.2.3 Parada de emergencia.....	27
4.3 Metodología.....	27

4.4	Planificación.....	29
4.5	Recursos utilizados.....	31
Capítulo 5.	Sistema Desarrollado	33
5.1	Programa de monitorización del entorno	33
5.1.1	Inicialización	34
5.1.2	Capturar imagen del entorno	39
5.1.3	Detección del robot	40
5.1.4	Detección de un obstáculo.....	50
5.1.5	Cálculo de distancia	54
5.1.6	Enviar dato	57
5.2	Programa para el control del robot	57
5.2.1	Declaración de variables/constantes.....	57
5.2.2	Proceso principal (Main)	58
5.2.3	Interrupción (recibir_mensajes).....	59
5.3	Soporte para el LIDAR.....	60
5.3.1	Primer diseño	61
5.3.2	Segundo diseño	63
Capítulo 6.	Análisis de Resultados.....	65
6.1	Comportamiento del robot.....	65
6.2	Tiempos de computación.....	67
6.2.1	Lenguaje Python.....	70
6.2.2	Reducir los datos de entrada.....	70
6.3	Pérdida de información	73
6.4	Obstáculos adicionales	74
6.4.1	Obstáculos pequeños	75
6.4.2	Obstáculos grandes	76
Capítulo 7.	Conclusiones y Trabajos Futuros.....	77
7.1	Distinto lenguaje de programación.....	78
7.2	Adaptación de trayectoria.....	78
Capítulo 8.	Bibliografía.....	81
ANEXO I.	Objetivos de Desarrollo Sostenible	87

Índice de ilustraciones

Ilustración 2-1. Esquema básico del LIDAR [3]	3
Ilustración 2-2. Ejemplo nube de puntos de una imagen.....	4
Ilustración 2-3. Ejes robot ABB IRB 120 [5].....	5
Ilustración 2-4. Alcance robot IRB 120 [6].....	6
Ilustración 2-5. Disposición del robot en el laboratorio	7
Ilustración 3-1. Robot KUKA LBR iiwa 14 R820 en fábrica de montaje Volkswagen [12]	11
Ilustración 3-2. Relación Resistencia-fuerza en escala logarítmica [15].....	13
Ilustración 3-3. Esquema funcionamiento sensor IR [17]	15
Ilustración 3-4. Ejemplo básico del funcionamiento de YOLO [19]	17
Ilustración 3-5. Sencillo ejemplo del funcionamiento de K-Means [25]	19
Ilustración 3-6. Estructura iterativa DBSCAN	21
Ilustración 3-7. Ejemplo DBSCAN	22
Ilustración 3-8. Funcionamiento del sistema FreeMove®	24
Ilustración 4-1. Estudios realizados en métodos de HRC [35].....	25
Ilustración 4-2. Cronograma de objetivos	30
Ilustración 4-3. Aspecto del sensor LIDAR utilizado [36].....	31
Ilustración 4-4. Robot y entorno de trabajo utilizados	31
Ilustración 5-1. Diagrama de bloques general	33
Ilustración 5-2. Estructura iterativa del programa	34
Ilustración 5-3. Estructuras nube de puntos	35
Ilustración 5-4. Fondo empleado en el proyecto	37
Ilustración 5-5. Nube de puntos original del LIDAR	38
Ilustración 5-6. Puntos a eliminar para seleccionar la mesa.....	39
Ilustración 5-7. Situación del entorno de trabajo.....	40
Ilustración 5-8. Flujo de operaciones para detectar el robot.....	41
Ilustración 5-9. Resultado diferencias imagen-fondo.....	42
Ilustración 5-10. Resultado de seleccionar puntos anaranjados	44

Ilustración 5-11. DBSCAN sobre los puntos naranjas	45
Ilustración 5-12. Asociación del mayor clúster con el robot.....	46
Ilustración 5-13. Píxeles asociados a los puntos naranjas del robot detectados	47
Ilustración 5-14. Selección de puntos dentro del recuadro.....	48
Ilustración 5-15. Diferencias imagen-fondo (dentro del rectángulo)	49
Ilustración 5-16. DBSCAN sobre los puntos en el marco del robot.....	49
Ilustración 5-17. Robot detectado en la imagen del entorno	50
Ilustración 5-18. Flujo de operaciones para detectar un obstáculo	51
Ilustración 5-19. Nueva imagen sin el robot	52
Ilustración 5-20. Diferencias de la imagen sin el robot con el fondo.....	52
Ilustración 5-21. DBSCAN para detectar obstáculo.....	53
Ilustración 5-22. Distinción del obstáculo en la imagen del entorno	54
Ilustración 5-23. Diferenciación fondo-robot-obstáculo	55
Ilustración 5-24. Distancia robot-obstáculo 30 cm.....	56
Ilustración 5-25. Distancia robot-obstáculo 20 cm.....	56
Ilustración 5-26. Distancia robot-obstáculo 10 cm.....	56
Ilustración 5-27. Estructura que encierra al robot	61
Ilustración 5-28. Primer diseño del soporte.....	62
Ilustración 5-29. Enfoque del primer diseño de soporte.....	63
Ilustración 5-30. Segundo diseño del soporte.....	64
Ilustración 5-31. Disposición final del soporte.....	64
Ilustración 6-1. Fotogramas significativos de la implementación. Perspectiva de un observador externo	66
Ilustración 6-2. Fotogramas representativos de la implementación. Perspectiva del LIDAR	67
Ilustración 6-3. Gráfico de tiempos de computación para una resolución de 640x480.....	68
Ilustración 6-4. Fondo con menor resolución.....	71
Ilustración 6-5. Imagen del entorno con menor resolución.....	71
Ilustración 6-6. Gráfico de tiempos de computación para una resolución de 320x240.....	73
Ilustración 6-7. Robot cercano al sensor	74

Ilustración 6-8. Clasificación de un escenario con obstáculo pequeño	75
Ilustración 6-9. Segmentación fondo-robot-obstáculo con elementos adicionales pequeños	75
Ilustración 6-10. Segmentación fondo-robot-obstáculo con elementos adicionales grandes	76
Ilustración 0-1. Objetivos de Desarrollo Sostenible [39]	87

Índice de tablas

Tabla 2-1. Limitaciones de alcance y velocidad [6]	6
Tabla 4-1. Cronograma de objetivos	30
Tabla 5-1. Códigos RGB de distintas tonalidades naranjas [37]	43
Tabla 5-2. Niveles de riesgo	55
Tabla 6-1. Tiempos de computación DBSCAN para una resolución de 640x480 píxeles..	69
Tabla 6-2. Tiempos de computación DBSCAN para una resolución de 320x240 píxeles..	72
Tabla 6-3. Comparación de tiempos para distintas resoluciones de imagen	72

Capítulo 1. INTRODUCCIÓN

Actualmente vivimos en pleno desarrollo de la cuarta revolución industrial- conocida como Industria 4.0-, que combina los planos físico y digital. Esta revolución apuesta por una mayor automatización y conectividad en los procesos apoyándose en diversas tecnologías y herramientas, como pueden ser la realidad y visión artificial, robótica colaborativa, programas seguridad, internet de las cosas (*internet of things*, IoT) ...

En cuanto a la automatización se trata, surge la necesidad de supervisar las máquinas cuando no trabajan en un entorno aislado, sino que existe la posibilidad de interactuar con otras o incluso con el factor humano. Así pues, se han abierto numerosas líneas de investigación enfocadas al reconocimiento de objetos y mapeo de un entorno, encontrando en los sensores LIDAR (un emisor láser de distancia) un gran potencial aún por explotar.

Sin embargo, hasta el momento los proyectos en los que se ha trabajado con sensores LIDAR se han centrado principalmente en entornos ajenos al sector industrial. Por ejemplo, una de las aplicaciones más populares últimamente de este tipo de sensores es la navegación autónoma de vehículos. En ella, a partir de la información recogida por el LIDAR el vehículo es capaz de regular su velocidad y dirección con el fin de evitar una colisión o seguir una trayectoria determinada. Dichos vehículos son comúnmente destinados al transporte de personas [1]-es la aplicación más conocida de este concepto-, aunque también se está probando en robots que circulan por las calles [2], robots destinados por ejemplo al reparto de paquetes o al reconocimiento facial.

Y aunque aún siguen desarrollándose proyectos para mejorar esta aplicación, ha resultado tener un éxito considerable gracias a la precisión de esta tecnología y a su fiabilidad con independencia de las condiciones ambientales del entorno de trabajo.

Así pues, se considera que, para mantener la seguridad en un entorno de trabajo industrial, en el cual un brazo robótico se mueve realizando determinadas tareas, la implementación de un software basado en las imágenes obtenidas con una cámara LIDAR podrá dar muy buenos resultados.

Para demostrarlo, se ha equipado un robot industrial ABB IRB120 de la minifábrica disponible en uno de los laboratorios de la Universidad Pontificia de Comillas -ICAI- con un sensor de este tipo para detectar si hay algún obstáculo en su área de trabajo. Gracias a un programa desarrollado en MATLAB, el robot será capaz de adaptar su velocidad en función de la distancia a la que se encuentre de dicho obstáculo y, en caso de ser necesario, se detendrá para evitar una colisión.

Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

2.1 LIDAR

A lo largo de todo el documento se va a mencionar en numerosas ocasiones el LIDAR, bien como sustantivo a secas o bien precedido por sensor o cámara. El acrónimo proviene de su definición en inglés *Laser Imaging Detection and Ranging*, aunque también se emplea el acrónimo LiDAR procedente de *Light Detection and Ranging*. Se trata de un sensor láser destinado a la representación tridimensional del entorno que percibe. En la Ilustración 2-1 se muestra un esquema básico de sus componentes principales:

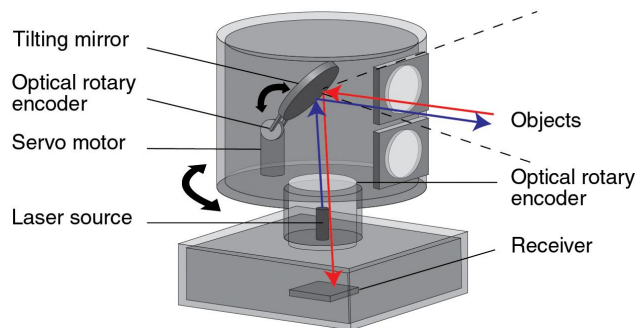


Ilustración 2-1. Esquema básico del LIDAR [3]

Cuenta con un emisor de haces de rayos láser que, antes de salir al exterior, rebotan contra un espejo giratorio, de manera que, aún con el LIDAR fijo, el rayo no siempre se dirige al mismo punto, consiguiendo así un amplio barrido del entorno. Cuenta con dos movimientos rotatorios: uno con eje vertical y otro horizontal. Una vez en el exterior, al encontrarse con un objeto el rayo rebota volviendo así al LIDAR donde será captado por una lente receptora.

Además, existe un procesador interno que se encarga de calcular las distancias basándose en la técnica del tiempo de vuelo, que establece la distancia en un medio de la velocidad del

rayo en el medio multiplicada por el tiempo que este tarda entre que es emitido y recibido por el sensor. El procesador registra junto con este dato la información de los encoders relacionados con las rotaciones vertical y horizontal y con ello es capaz de ubicar en el espacio cada punto detectado. Se consigue así una representación tridimensional del entorno de gran precisión, denominada nube de puntos.

Para ayudar a tener una idea más clara de este concepto, en la Ilustración 2-2 se muestra una imagen a color de un entorno y su nube de puntos asociada.

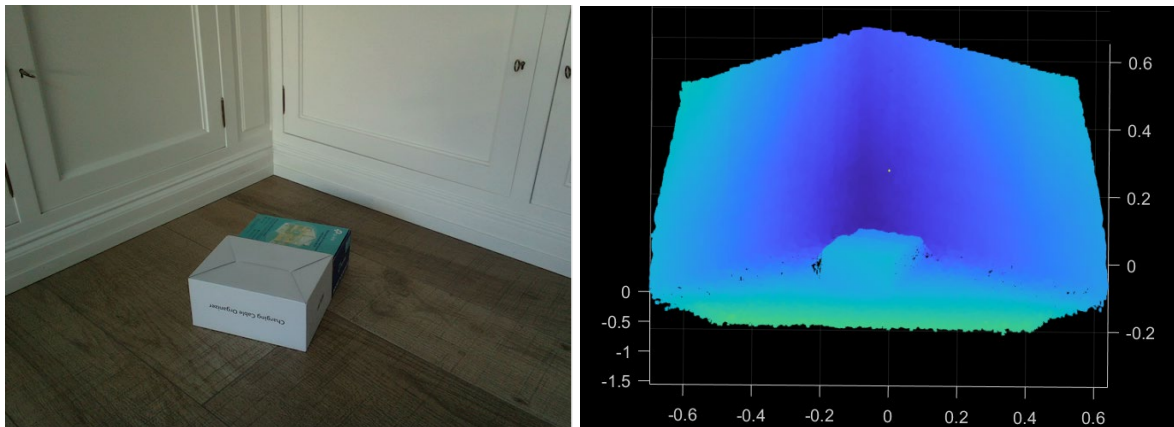


Ilustración 2-2. Ejemplo nube de puntos de una imagen

El LIDAR en cuestión con el que se realiza el trabajo es el Intel® RealSense™ LiDAR Camera L515, que es capaz de capturar, además de la nube de puntos mencionada, una imagen RGB con una resolución máxima de 1920x1080 píxeles y un campo de visión de 70°x43° ($\pm 3^\circ$) a una velocidad máxima de 30 FPS (Fotogramas Por Segundo).

Por otro lado, las características técnicas de la cámara en cuanto a percepción de profundidad son las siguientes:

- Resolución máxima de 1024x768 píxeles.
- Campo de visión de 70°x55° ($\pm 3^\circ$). Ligeramente superior al de la imagen RGB.
- Velocidad máxima de 30 FPS.
- Mínima y máxima distancia detectables: 25 cm y 9 m respectivamente.

- Precisión de entre 5 y 14 mm.
- Cotas en metros (m).

2.2 BRAZO ROBÓTICO

La implementación del proyecto se lleva a cabo en un robot industrial de la empresa ABB, en concreto el modelo IRB 120 [4]. Este es el más pequeño de entre los robots que ofrece la compañía, con tan solo 25 kg de peso y unas dimensiones de 180x180x700 mm, capaz de soportar una carga de 3 kg. Cuenta con 6 ejes (ver Ilustración 2-3) que le proporciona un alcance de 580 mm (ver Ilustración 2-4) y cada uno de ellos con unas limitaciones de velocidad y alcance según especifica la Tabla 2-1.

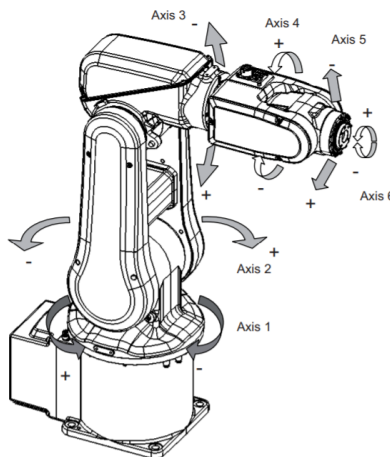


Ilustración 2-3. Ejes robot ABB IRB 120 [5]

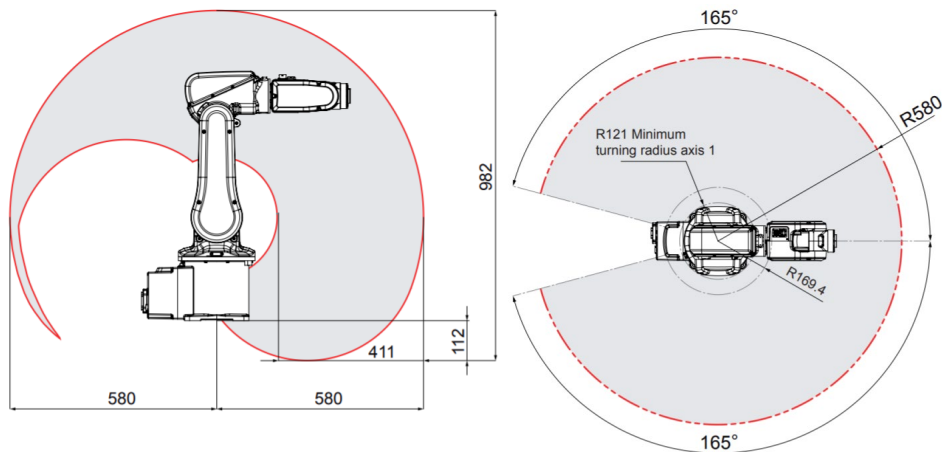


Ilustración 2-4. Alcance robot IRB 120 [6]

EJE	TIPO DE MOVIMIENTO	RANGO DE TRABAJO	VELOCIDAD MÁXIMA
1	Movimiento de rotación	+165° a -165°	250 °/s
2	Movimiento del brazo	+110° a -110°	250 °/s
3	Movimiento del brazo	+70° a -110°	250 °/s
4	Movimiento de la muñeca	+160° a -160°	320 °/s
5	Movimiento de doblado	+120° A -120°	320 °/s
6	Movimiento de giro	+400° a -400° (por defecto) +242 a -242 (revoluciones como máximo)	420 °/s

Tabla 2-1. Limitaciones de alcance y velocidad [6]

El robot se encuentra en el laboratorio de automatización de la Universidad Pontificia de Comillas- ICAI- montado en unas estructuras a modo de jaulas como medida de seguridad, tal como se puede ver en la Ilustración 2-5. De esta manera, el robot puede estar trabajando en modo automático siempre y cuando la puerta de la jaula esté cerrada, pasando a modo manual en cuanto esta sea abierta.

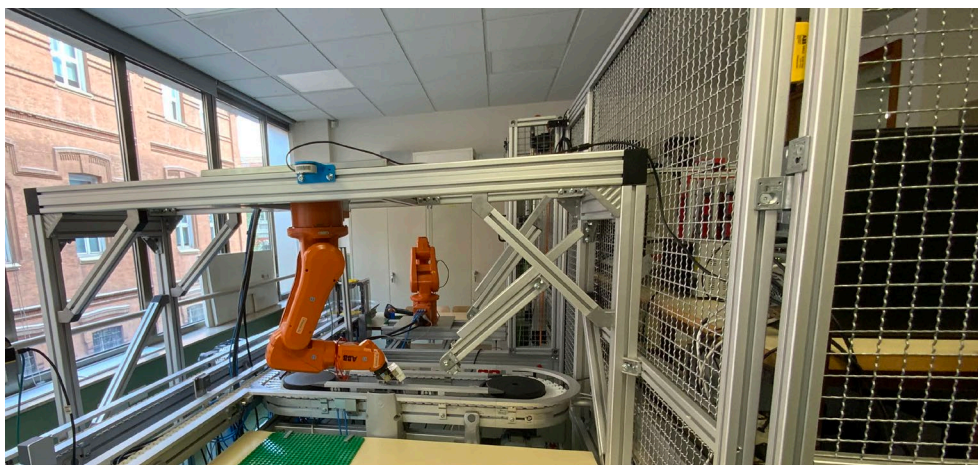


Ilustración 2-5. Disposición del robot en el laboratorio

Así, aunque el cambio de unidades para velocidad de $^{\circ}/s$ a m/s depende del radio de giro, se estima que el robot alcanza unos $6 m/s$ cuando trabaja en automático, esto es $20 km/h$ aproximadamente. Sin embargo, al pasar a modo manual la velocidad se reduce a menos de la mitad para asegurar que una colisión con el robot no resulte peligrosa.

Capítulo 3. ESTADO DE LA CUESTIÓN

Cada vez son más las tareas que la tecnología es capaz de realizar de manera independiente, automatizada y sin necesidad de supervisión humana en entornos compartidos, bien con otras máquinas o bien con trabajadores. Por ello, es preciso que los equipos encargados de llevar a cabo estas tareas cuenten con una serie de sensores que les permitan tener una noción de lo que ocurre a su alrededor y así poder adaptar su actividad.

Surge así una tecnología orientada a garantizar la seguridad en un entorno compartido por robots y humanos, conocida como HRC (siglas del inglés *Human-Robot Collaboration*). Su aparición se remonta al año 2004 con el primer robot colaborativo- conocidos ahora con el acrónimo *cobots* del inglés *collaborative robots*- lanzado por la empresa alemana KUKA, modelo LBR3. Sin embargo, no se generaliza el uso de este tipo de robots hasta 2008, cuando el fabricante danés Universal Robots comercializa el UR5, el primer *cobot* que garantiza un funcionamiento seguro en colaboración con humanos sin necesidad de jaulas o vallas [7].

En esta sección, se van a tratar los 3 aspectos básicos para lograr esta colaboración entre robots y humanos: distintas técnicas para la prevención de riesgos, posibles sensores integrados en las máquinas para percibir el entorno y los algoritmos empleados para tratar la información obtenida con los sensores. Además, se tratarán algunas soluciones implementadas recientemente en el mundo industrial con el fin de garantizar la seguridad cuando se trata de combinar acción humana con actividades automatizadas llevadas a cabo por máquinas.

3.1 COLABORACIÓN HUMANO-ROBOT (HRC)

A grandes rasgos, para hacer posible la actividad de estos dos factores en un mismo entorno de trabajo y garantizar la seguridad se pueden diferenciar dos enfoques principales: una

opción cooperativa y otra colaborativa. En la primera, ambos factores desarrollan su trabajo en un mismo entorno, pero nunca al mismo tiempo; mientras que en la segunda pueden coexistir, ya sea realizando tareas independientes o coordinadas.

A su vez, se distinguen 4 tipos de HRC [8]: parada segura monitorizada, guiado manual, control de velocidad y separación y, por último, limitación de fuerza y potencia. Estos se encuentran especificados en las normas estándar ISO EN 10218-1 [9] y 10218-2 [10], donde se recogen una serie de requisitos de diseño y medidas de protección para cada uno, pero es en la norma ISO/TS 15066 [11] donde se amplían y se exponen detalladamente los requisitos de cada método (máxima fuerza, velocidad, potencia, etc.).

3.1.1 PARADA SEGURA MONITORIZADA (SMS)

Las siglas provienen de su denominación en inglés *Safety-rated Monitored Stop*. Es el método más sencillo e impide que el robot y el operario trabajen simultáneamente en un mismo entorno, ya que el robot se detendrá en cuanto el operario entre en el entorno de trabajo o bien se le impedirá a este entrar mientras el robot esté realizando su tarea.

Una manera de lograr este funcionamiento es con una barrera física, colocando una jaula o vallas alrededor de la maquinaria bloqueando así el acceso mientras esta se encuentre en movimiento. La otra alternativa es colocar una serie de sensores a lo largo del perímetro del área colaborativa para así detectar la entrada del trabajador y detener el robot.

3.1.2 GUIADO MANUAL (HG)

Las siglas provienen de su denominación en inglés *Hand Guiding*. En cuanto una persona entra en el entorno de trabajo el robot se detiene, pero se permite la actividad simultánea de ambos factores siempre y cuando el robot esté guiado por el trabajador y no funcione de manera autónoma. Para una mayor fiabilidad, se reduce la velocidad de operación y se incorporan sensores de fuerza externa para controlar la actividad del robot.

Los robots destinados a este tipo de actividades tienen una forma bastante ergonómica con perfiles redondeados- como se aprecia en la Ilustración 3-1- y su manejo es bastante intuitivo

ya que están restringidos a realizar determinados movimientos. El diseño de estos movimientos se realiza de manera que tanto el operario como el robot lleven a cabo la tarea de la manera más fluida posible y evitando colisiones [12]. La simulación es una herramienta que se está abriendo camino para diseñar este tipo de maquinaria, pues permite estudiar distintos escenarios para un puesto de trabajo y su tarea.



Ilustración 3-1. Robot KUKA LBR iiwa 14 R820 en fábrica de montaje Volkswagen [12]

3.1.3 CONTROL DE VELOCIDAD Y SEPARACIÓN (SSM)

Las siglas provienen de su denominación en inglés *Speed and Separation Monitoring*. El área de trabajo colaborativa se divide en varias zonas que estarán bajo la vigilancia de escáneres o sensores ópticos. Así, el robot puede operar a máxima velocidad/potencia salvo cuando se detecte la aparición de un agente externo en alguna de estas zonas, en cuyo caso el robot reducirá su velocidad y la adaptará en función de su distancia con dicho agente externo.

El objetivo de esta técnica es mantener en todo momento el robot a una distancia segura de cualquier persona, de manera que sea compatible con la trayectoria del robot y suficiente para que este realice una parada sin colisionar [13].

De esta manera se reduce considerablemente el tiempo que la máquina se encuentra parada, aunque el robot podrá llegar a detenerse cuando no sea capaz de mantener una distancia mínima, la cual viene dada por la norma ISO/TS 15066 [11] y sigue la ecuación (1):

$$D_{\min} = k_H(t_1 + t_2) + k_R t_1 + B + \delta \quad (1)$$

Donde las variables tienen los siguientes significados:

k_H : velocidad del humano

k_R : velocidad del robot

B : distancia euclidiana necesaria para detener el robot

3.1.4 LIMITACIÓN DE FUERZA Y POTENCIA (PFL)

Las siglas provienen de su denominación en inglés *Power and Force Limiting*. Mientras que las técnicas de HRC recién comentadas pueden aplicarse, en términos generales, a todos los *cobots* del mercado, esta última requiere de robots específicos que limiten su fuerza/potencia en caso de impacto para así disminuir el riesgo.

El robot puede medir su posición y velocidad gracias a encoders, así como las fuerzas y los pares se miden con sensores de par situados en sus articulaciones o únicamente midiendo la corriente por los motores [14]. Con esta información el robot es capaz de identificar un impacto con un obstáculo y reaccionar instantáneamente frenando o ejerciendo una fuerza en sentido contrario para así disminuir la energía del impacto.

3.2 *SENSORES PARA PERCIBIR EL ENTORNO*

Para que el robot sepa cuándo debe adaptar su actividad necesita tener controlado su entorno y detectar si este sufre un cambio imprevisto. Por ello, los sensores juegan un papel muy importante en esta colaboración humano-robot y se le incorporan o bien al robot o bien al entorno de trabajo.

Existen una gran variedad y en función del método HRC serán convenientes unos u otros. En este apartado se van a comentar algunos de los sensores con más utilidad y potencial en este ámbito de la industria.

3.2.1 SENSORES DE FUERZA

También referidos a ellos como sensores de presión, destacan los resistivos, que disminuyen su resistencia logarítmicamente a medida que aumenta la fuerza ejercida sobre ellos, como se muestra en la Ilustración 3-2.

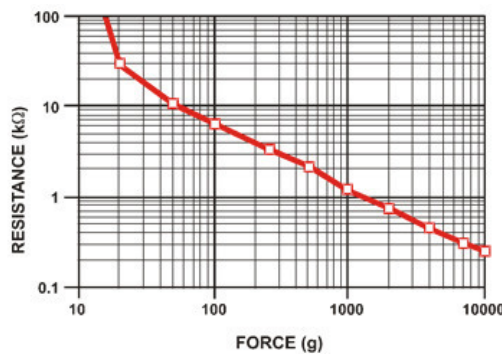


Ilustración 3-2. Relación Resistencia-fuerza en escala logarítmica [15]

Entre sus aplicaciones relacionadas con HRC, pueden emplearse para detectar la irrupción en el entorno de trabajo y activar consecuentemente el freno de la máquina (SMS) o para recubrir el robot, o partes críticas, y así detectar colisiones (PFL).

3.2.2 SENSORES DE ULTRASONIDO

Miden la distancia a un objeto emitiendo una onda ultrasónica que rebota contra este devolviendo un eco que es recibido el cabezal. Dicha distancia se halla en base al tiempo transcurrido entre la emisión y la recepción de la onda-conocido como tiempo de vuelo-, siguiendo la ecuación (2):

$$d = \frac{1}{2} \cdot v \cdot t_f \quad (2)$$

Donde v es la velocidad de la señal en el medio en el cual se esté transmitiendo y t_f es el tiempo de vuelo.

La limitación de este tipo de sensores es que tienen una alta dependencia con el medio, pues la velocidad de la señal dependerá de la temperatura o la humedad y también se verá afectada por ruido, interferencias, corrientes de convección... dificultando todo ello su correcta estimación. Además, la determinación del tiempo de vuelo también supone un ligero reto ya que la señal eco puede malinterpretarse si es atenuada por el medio o porque depende de la forma y orientación del objeto reflectante, dando lugar a una subestimación o sobreestimación de la distancia [16].

Así pues, aunque la medición de distancias en base a sensores ultrasónicos resulta más sencilla y menos costosa, su baja resolución y alta dependencia con el entorno descartan su uso para aplicaciones que requieran gran precisión como es garantizar la seguridad en un entorno compartido por humanos y máquinas.

3.2.3 SENSORES FOTOELÉCTRICOS

Por otro lado, se encuentran también para medir distancias los sensores fotoeléctricos, destacando los de luz infrarroja (IR), más utilizados en este campo ya que presentan una mayor precisión, ofrecen una respuesta más rápida y no tienen tanta dependencia con las condiciones ambientales (temperatura, presión...). Si bien, la detección de los objetos viene condicionada por su material o color, pues es necesario que reflejen la luz.

Este tipo de sensores cuentan con un LED infrarrojo y un detector de luz (PSD, siglas del inglés *Position Sensing Device*). El LED emite un haz de luz IR que se reflejará en un objeto volviendo al sensor y alcanzando el detector para formar un “punto óptico”. En función de la distancia a la que se encuentre el objeto, este punto óptico tendrá una determinada posición en el receptor (véase Ilustración 3-3). El sensor cuenta además con un circuito de procesamiento de señales que determina en base a dicha posición la distancia al objeto reflectante (principio de triangulación), la cual se transmite con una señal analógica.

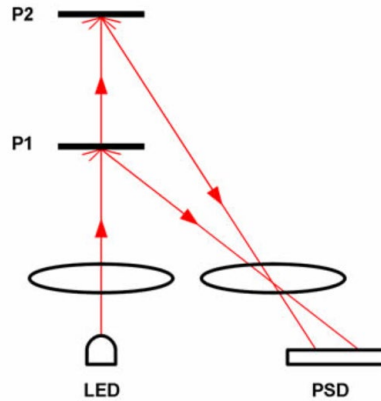


Ilustración 3-3. Esquema funcionamiento sensor IR [17]

3.2.4 SENSORES DE IMAGEN

A esta familia de sensores pertenecen las conocidas cámaras, destacando las RGB-D (*Red Green Blue and Depth*) por ser unas de las más utilizadas actualmente en aplicaciones de robótica. Esto se debe a que, además de capturar una imagen a color, registra datos de profundidad gracias a sensores infrarrojos incorporados.

3.2.5 SENSORES LÁSER

Los principios para la medición de distancias de este tipo de sensores son similares a los comentados anteriormente para ultrasonidos o infrarrojos, pues los hay que se basan en el tiempo de vuelo y otros que utilizan el principio de triangulación. La diferencia fundamental es que el emisor emitirá rayos láser, que no presenta dependencias tan fuertes con el entorno ofreciendo así una mayor precisión.

3.3 PROCESAMIENTO DE DATOS

Es necesario interpretar la información que recogen los sensores para poder reconocer los distintos objetos que conforman el entorno, así como identificarlos, clasificarlos en distintos grupos o hacer un seguimiento de algunos que susciten interés. Y es que esta interpretación de imágenes, ya sean de color o tridimensionales con nubes de puntos, aunque para el ser

humano es algo tan simple y rápido, es hoy en día un gran reto con ya numerosas soluciones, pero que aún siguen en desarrollo.

Aunque el comienzo del desarrollo de algoritmos destinados a esta función se remonta varias décadas atrás, en 2012 se introdujo la red neuronal convolucional (CNN, del inglés *Convolutional Neural Network*) AlexNet, que marcó esta área de trabajo y se convirtió en una herramienta muy potente para la clasificación de imágenes, sentando un precedente en el campo del *Deep Learning* (DL). Su arquitectura está formada por 8 capas: 5 de convolución y 3 completamente conectadas, conteniendo en la última de ellas una función de activación *softmax* con la que se consigue un vector que contiene la distribución de probabilidad sobre mil clases. [18]

A partir de entonces se empezó a desarrollar el gran potencial que tienen las CNN en la resolución de los desafíos relacionados con la visión artificial y a continuación se van a tratar algunas de las que más destacan o han destacado a lo largo de los años, así como algunos algoritmos más sencillos en los que estas redes se ayudan.

3.3.1 YOLO (YOU ONLY LOOK ONCE)

Actualmente, dentro del ámbito de la detección de objetos destaca el algoritmo YOLO (del inglés *You Only Look Once*), por su buen balance entre rapidez y precisión alcanzando con su segunda versión- YOLOv2- niveles de 76.8 mAP (Mean Average Precision) a 67 FPS y 78.6 mAP a 40 FPS.

YOLO [19] se modela como un problema de regresión que resuelve en tiempo real la segmentación y clasificación de una imagen, es decir, proporciona múltiples cajas o *bounding boxes* a la vez que la probabilidad de clase para cada una de ellas. Para ello, la imagen de entrada se divide en una cuadrícula de dimensión $S \times S$ a determinar por el usuario y si el centro de un objeto cae en una de las celdas, esta será la encargada de detectar al objeto. Cada *bounding box* resultante llevará asociada la siguiente información: coordenadas (x,y) del centro de la caja, el ancho y alto de la caja y el grado de confianza. En

la Ilustración 3-4 se expone una representación muy básica pero muy visual de proceso seguido por el algoritmo.

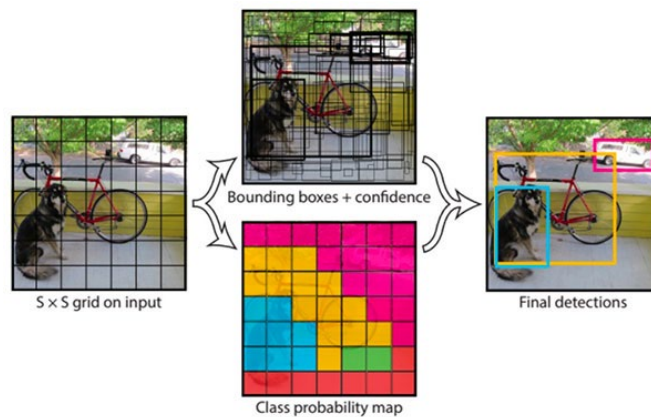


Ilustración 3-4. Ejemplo básico del funcionamiento de YOLO [19]

En su primera versión, YOLOv1 se estructura en una única red convolucional con 24 capas convolucionales seguidas de 2 capas completamente conectadas, mientras que YOLOv2 utiliza otra red dentro de la primera reduciendo el número de capas convolucionales a 19. En esta segunda versión del algoritmo se mejora sobre todo la precisión incluyendo un proceso de normalización para la imagen de entrada, 5 capas *max pooling* (que compactan la cantidad de información con un criterio de máximos), aumentando la resolución de la cuadrícula... [20]

Sin embargo, entre las limitaciones de YOLO destaca la dificultad para detectar objetos pequeños si se encuentran en grupo o para identificar objetos posicionados de manera inusual. Por ello, el algoritmo se encuentra en constante desarrollo, y la versión YOLOv3 consigue una mayor precisión en cuanto a pequeños objetos. Entre los cambios que esta incorpora, destaca la sustitución de la capa *softmax* por clasificadores logísticos independientes, que permite un enfoque multiclase ya que un mismo objeto puede pertenecer a más de una clase (por ejemplo, animal y perro). [21]

Si bien este método de segmentación y clasificación fue inicialmente enfocado al estudio de imágenes 2D, su aplicación puede extrapolarse a las tres dimensiones que ofrecen las nubes de puntos captadas por un sensor LIDAR. Para ello existen básicamente dos alternativas:

1. Transformar la imagen 3D a 2D y utilizar YOLO. Esta transformación se basa en la proyección de los puntos en una vista de pájaro, en una frontal o en *voxels*, con el inconveniente de que en este proceso se pierde información de la nube de puntos.
2. Crear un nuevo algoritmo basado en YOLO que trabaje con la nube de puntos. Soluciona el problema de la pérdida de información, pero en consecuencia es un proceso más lento. En 2017 Charles et al. marcaron un antes y un después en esta técnica al proponer una red de DL denominada PointNet [22] que trabajaba por primera vez directamente con las nubes de puntos. A partir de ese momento surgen nuevos algoritmos con el mismo propósito, como por ejemplo PointRCNN desarrollado por Shaoshuai et al. en [23] o SO-Net por Jiaxin et al. en [24].

3.3.2 POINTNET [22]

Este algoritmo está destinado a la segmentación y clasificación de imágenes 3D. Se trata de una CNN compuesta por dos redes: una de clasificación y otra de segmentación, aunque ambas comparten gran parte de su estructura. A partir de N datos de entrada en forma de vectores (x,y,z) , el objetivo de clasificación es para cada objeto detectado proporcionar k puntuaciones, una para cada una de las k clases candidatas. Y en cuanto a la segmentación semántica, el algoritmo proporciona un vector $N \times m$ de manera que para cada punto de entrada se indica la puntuación asociada a cada una de las m subcategorías semánticas.

La estructura de la red se divide en 3 grandes módulos: uno para ordenar, canonizar y entrenar los datos de entrada; otro para asociar características locales y globales a los datos; y por último otro para alinear los datos mediante una matriz de transformación obtenida en una mini red aparte, para ser posteriormente procesados por la capa *softmax*.

3.3.3 ALGORITMO K-MEANS

Su objetivo es dividir un conjunto de datos de entrada en diferentes clústeres (ver Ilustración 3-5), por lo cual tiene una aplicación directa en la segmentación. Para ello, partiendo de una asignación aleatoria de clústeres, realiza de manera iterativa las siguientes operaciones:

1. Calcular el centro de cada clúster teniendo en cuenta los datos que han sido asociados a cada uno.
2. Volver a asociar los datos a un clúster, de manera que cada dato pertenezca al clúster con cuyo centro mantiene la menor distancia.
3. Calcular el error de la clasificación sumando los cuadrados de las distancias de cada punto al centro de su clúster asociado.

Las iteraciones finalizarán, dando con el resultado de la segmentación, cuando el valor de este error se comience a estabilizar.

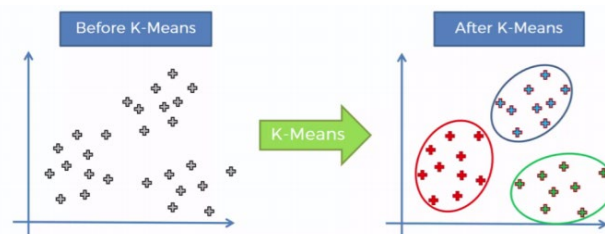


Ilustración 3-5. Sencillo ejemplo del funcionamiento de K-Means [25]

Xuanyu et al. se aprovechó de este algoritmo para ampliar la utilidad de YOLO con nubes de puntos. En [26] desarrolla un método de clasificación y segmentación en el cual registra un entorno con cámaras RGB, a cuyas imágenes aplica YOLO, y con un LIDAR. A partir de los resultados obtenidos con YOLO cada punto de la nube se asigna a un *bounding box* en función de su ubicación, así partiendo de esta primera clasificación se aplica K-Means para mejorar el resultado.

3.3.4 ALGORITMO DBSCAN

Acrónimo procedente del inglés *Density Based Spatial Clustering of Applications with Noise*, es un algoritmo de agrupamiento (*clustering*) basado en densidad y utilizado para identificar clústeres de formas arbitrarias en un conjunto de datos con ruido y valores atípicos (*outliers*).

Un clúster se trata de una región densa de puntos que se encuentra separada de los demás clústeres por regiones de menor densidad. Para su distinción, DBSCAN funciona a partir de dos parámetros definidos por el usuario:

- Épsilon (eps): especifica la máxima distancia a la que pueden encontrarse dos puntos para ser considerados vecinos.
- Puntos mínimos (minPts): indica el número mínimo de puntos vecinos necesarios para ser considerados como región densa o clúster.

Los puntos pueden clasificarse en tres tipos distintos:

- Punto de núcleo: es considerado tal si dentro de un radio igual a épsilon contiene un número de vecinos mayor o igual al mínimo especificado (minPts). Siempre pertenece a un clúster.
- Punto de borde: dentro de un radio igual a épsilon contiene menos vecinos de los mínimos especificados, pero él mismo es un vecino de un punto de núcleo. Por lo tanto, forma parte de un clúster, en concreto de su frontera.
- Punto de ruido: resto de puntos, es decir, contiene menos vecinos del mínimo especificado y no es vecino de un punto de núcleo. No pertenece a ningún clúster.

El procedimiento seguido por el algoritmo se muestra en la Ilustración 3-6 y es explicado a continuación.

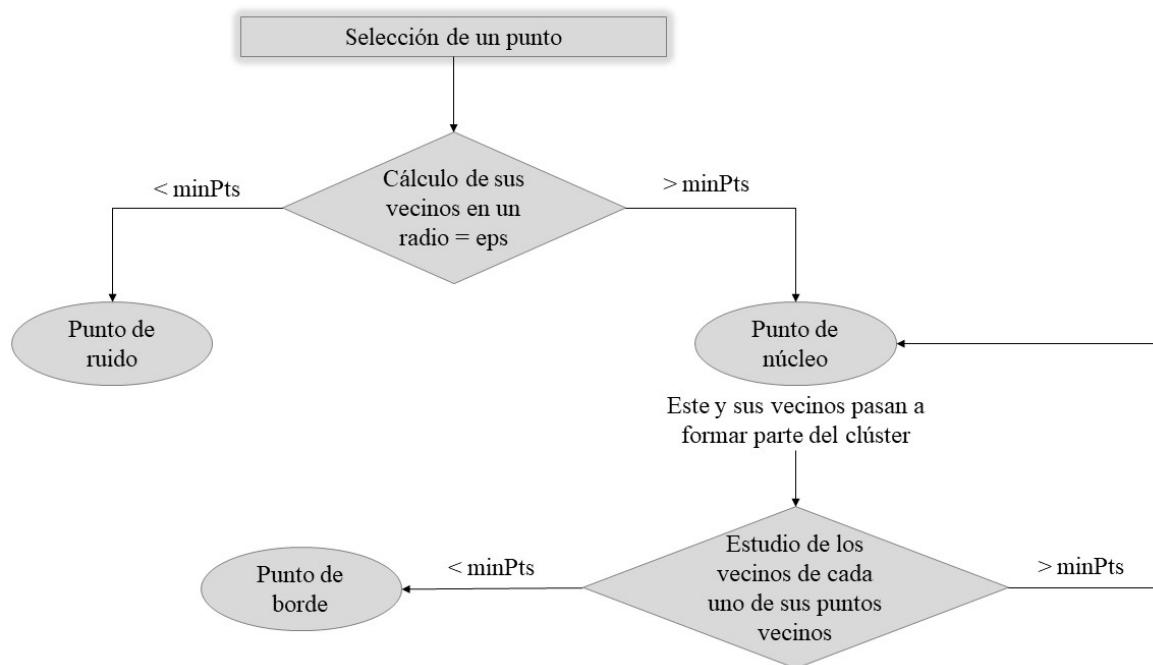


Ilustración 3-6. Estructura iterativa DBSCAN

1. Se selecciona aleatoriamente un punto (en el caso de no tratarse de la primera iteración, será un punto que no haya sido seleccionado en iteraciones anteriores).
2. Se cuenta su número de vecinos en un radio igual a épsilon para poder clasificarlo:
 - a. Punto de ruido si hay menos vecinos que el mínimo establecido (minPts), finalizando así la iteración. Un punto de este tipo podrá pasar a formar parte de un clúster si en iteraciones posteriores se clasifica como punto de borde.
 - b. Punto de núcleo si hay suficientes vecinos, continuando con el siguiente paso de la iteración.
3. Tanto el punto de núcleo como sus vecinos pasan a formar parte del clúster. A su vez, se estudia cuántos vecinos tiene cada uno de estos puntos vecinos del núcleo:
 - a. Si tiene menos de minPts es considerado punto de borde y se pasará al siguiente vecino en caso de haberlo, en caso contrario se llega al final de la iteración.
 - b. Si tiene más de minPts es considerado punto de núcleo, volviendo a comenzar el tercer paso de la iteración.

La Ilustración 3-7 muestra un ejemplo de la segmentación de un conjunto de datos realizada por DBSCAN, en la cual se identifican tres clústeres y ningún ruido.

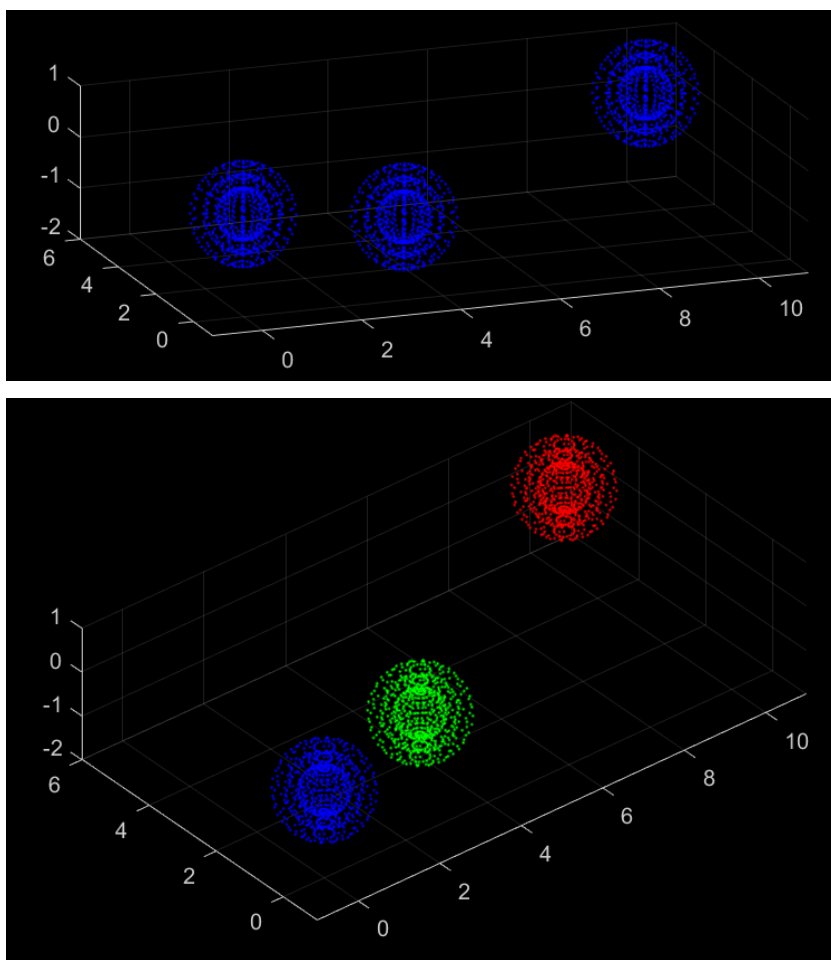


Ilustración 3-7. Ejemplo DBSCAN

3.4 APLICACIONES EN LA INDUSTRIA

Estos tres factores cruciales del HRC se combinan para dar lugar a sistemas de seguridad y, sin duda, hay una combinación protagonista en el mundo de la industria: Control de Velocidad y Separación como método y cámaras RGB-D como instrumento para percibir el entorno.

En [27] se expone el funcionamiento de un sistema resultante de esta combinación, donde el espacio de trabajo se divide en dos: el área donde trabaja el robot, en la cual se activa una estrategia para evitar colisiones basándose en los datos de profundidad de la cámara; y una zona fuera del alcance del sensor de profundidad, pero monitorizada por una cámara RGB para detectar la presencia de una persona procesando la imagen con YOLO. El alcance de esta imagen puede llegar hasta los 10 m y el uso del sensor de profundidad puede limitarse a una zona mucho más reducida donde realmente sea crítica la colaboración de las dos partes. Así, hasta que con la imagen RGB no se detecte la presencia humana no se activa el sistema basado en los datos de profundidad.

Aunque este tipo de aplicaciones son las más comunes y fáciles de implementar dado que ya están bastante establecidas, se sigue estando muy lejos de dotar a los robots con una percepción similar a la de los humanos. Esto es porque se pierden muchos datos percibiendo únicamente color y profundidad, como por ejemplo la velocidad o la intención de movimiento de la persona. Hasta ahora la solución más común viene dada por la combinación de sensores [28]. A. S. Modur propone en [29] un sistema de sensores formado por seis radares de 60 GHz y seis cámaras para seguir con precisión la ubicación y velocidad de los trabajadores en los 360° alrededor del robot. Si bien, estas combinaciones no se realizan solo para aumentar la información recogida, sino también para coordinar varios métodos de prevención de riesgos. Por ejemplo, el SSM con el PFL, de manera que se monitorice la distancia y cuando se alcance un mínimo se active el sistema limitador de fuerza [30].

Una vez implantadas estas técnicas, se busca la manera de percibir toda esta información sin recurrir a la incorporación de tantos sensores. Surge así el auge por los sensores 3D y el desarrollo de sistemas de seguridad basados en ellos [31] [32] [33]. La empresa *Veo Robotics* ha desarrollado por ejemplo un sistema de seguridad, denominado *FreeMove®*, que se encarga de mantener la seguridad en un entorno de trabajo. El sistema está formado por una serie de sensores 3D de su propio desarrollo, basados en el tiempo de vuelo, que se colocan en la periferia del área de trabajo para conseguir una representación exacta. Cuenta además

con un ordenador propio industrial encargado de procesar los datos de los sensores, localizando en todo momento al robot y su posible posición en un futuro cercano, las piezas con las que se esté trabajando y un posible humano. Todo ello tal y como se muestra en la Ilustración 3-8 y calculando la distancia del humano a un posible peligro. [34]

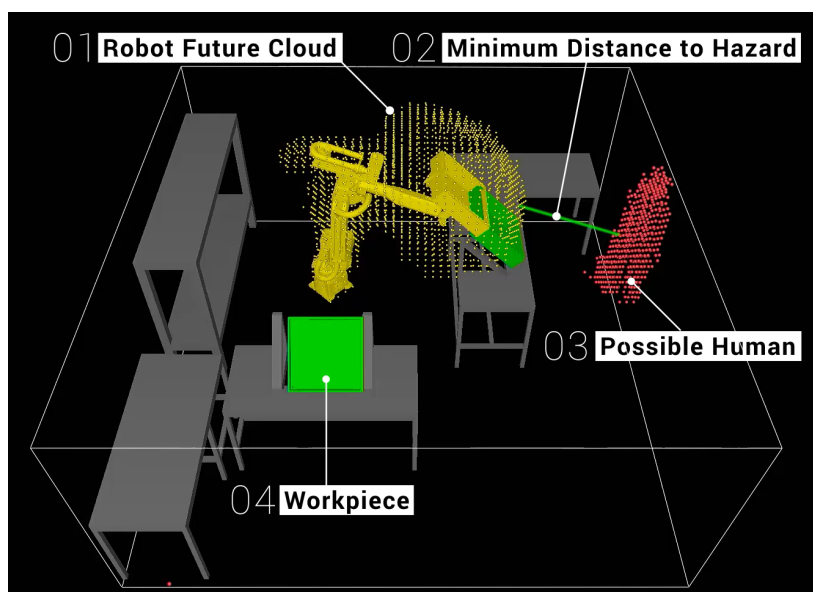


Ilustración 3-8. Funcionamiento del sistema FreeMove®

Capítulo 4. DEFINICIÓN DEL TRABAJO

4.1 JUSTIFICACIÓN

Actualmente no se está aprovechando al cien por cien las capacidades de producción. Hay procesos que necesitan de la fuerza y velocidad de una máquina a la par que la destreza y versatilidad de los humanos, pero para poder unir sus esfuerzos es necesario rebajar el rendimiento de ambos para prevenir accidentes.

Como destaca la sección 3.1, desde hace unas décadas se vienen desarrollando los métodos HRC para facilitar la coordinación del trabajo entre humanos y robots. La Ilustración 4-1 muestra los esfuerzos destinados al desarrollo de cada uno de estos métodos a lo largo del tiempo.

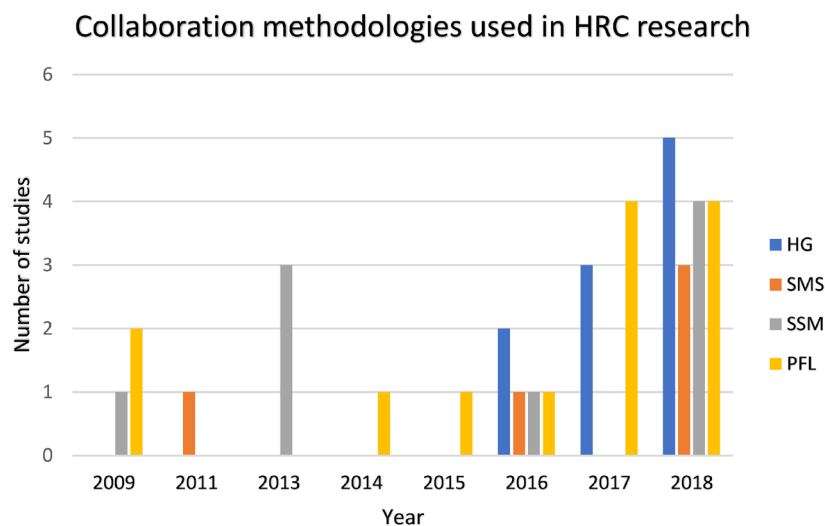


Ilustración 4-1. Estudios realizados en métodos de HRC [35]

Con este proyecto se pretende contribuir al estudio de la implementación de un sistema basado en el Control de Velocidad y Separación (SSM), pues se considera que hace que la colaboración entre humanos y robots sea lo más parecida posible a cómo trabajarían dos humanos a la vez: sincronizados y sin estorbarse.

Además, es un método que busca mantener al robot operando el mayor tiempo posible, parándose solo en caso de ser estrictamente necesario y siempre a la máxima velocidad que le permita la situación del entorno. Por ello, desde el punto de vista empresarial, es el que ofrece mayores ventajas, pues asegura que el robot trabaje a máxima potencia durante mayores periodos de tiempo aumentando así la eficiencia y el rendimiento de la tarea llevada a cabo.

4.2 OBJETIVOS

La principal finalidad del proyecto es crear un entorno de trabajo seguro en el que se pueda mantener el funcionamiento de una máquina sin necesidad de pararla cada vez que se desee acceder a dicho entorno y, por supuesto, sin tener ningún accidente a la hora de hacerlo. Para conseguirlo se deben cubrir los aspectos descritos a continuación.

4.2.1 RECONOCIMIENTO DEL ENTORNO

Como base del programa desarrollado, se deben identificar los distintos factores que participan en el entorno, clasificados en:

- Elementos pertenecientes al entorno habitual de trabajo, es decir, los que forman parte de este en todo momento, como por ejemplo el inmobiliario. Al conjunto de estos elementos se le va a denominar fondo y será crucial para identificar al resto de los factores.
- Robot. Estará realizando de manera autónoma una determinada tarea para la cual, en ocasiones, necesitará entrar en el entorno de trabajo. Una vez esto ocurra, el programa debe tenerlo identificado en todo momento.

- Elementos externos. Cualquier elemento que entre en escena ajeno al fondo y que no sea el robot. Es lo que se denominará a lo largo del documento como obstáculo y que comúnmente se tratará de una persona (o parte de ella), si bien también puede ser algún objeto apoyado en la mesa o que se acerque al robot.

4.2.2 CONTROL DE VELOCIDAD

En caso de que coexistan en el entorno de trabajo el robot y un elemento externo, el programa debe monitorizar en todo momento la distancia que mantienen estos dos factores. A partir de ella se adaptará la velocidad del robot, de manera que a medida que vaya disminuyendo esta distancia, disminuirá también la velocidad, para así reducir el riesgo de que ocurra un accidente.

Si dicho elemento externo se tratase del factor humano, con esta medida se le facilita a la persona en cuestión realizar su tarea esquivando fácilmente al robot, pues su velocidad reducida le otorga tiempo suficiente para reaccionar ante sus movimientos.

4.2.3 PARADA DE EMERGENCIA

En algunas ocasiones el control de velocidad puede no ser suficiente para evitar un accidente. Es por ello por lo que el programa debe contar con una parada de emergencia, gracias a la cual si la distancia entre robot y obstáculo alcanza un mínimo establecido se deberá frenar la máquina inmediatamente.

4.3 METODOLOGÍA

Gran parte del proyecto se trata del procesamiento de los datos obtenidos con la cámara LIDAR. Esta proporciona una nube de puntos en 3 dimensiones de todo aquello que se encuentre en su campo de visión. Se trabajará a partir de ella y apoyándose en ocasiones en la información RGB que también recoge.

En primer lugar, se trabaja en la identificación de superficies para poder distinguir así los distintos objetos que observe la cámara. La idea base es capturar el entorno habitual de trabajo en estático y establecer las superficies que se hayan distinguido como fondo (guardándolo en una variable), facilitando así la identificación de elementos externos que entren en escena.

Una vez en la minifábrica, lo primero es colocar el LIDAR de manera que enfoque la zona en la que el robot se puede mover, lo que constituirá el entorno de trabajo. Aunque provisionalmente para las primeras pruebas la cámara puede colocarse haciendo uso de un trípode, se acaba recurriendo a la impresión 3D para hacerse con un soporte a medida más fiable y adaptado a las necesidades del proyecto.

Tras colocar la cámara, se recoge la información de dicho fondo, así como una serie de vídeos del brazo robótico en movimiento. Con ello se comienza a trabajar en la siguiente etapa: la segmentación del robot. Además de identificar el brazo robótico por sus superficies, se debe distinguir del resto del entorno en todo momento durante su movimiento.

Hasta aquí el programa es capaz de identificar cuándo aparece en escena un elemento ajeno al fondo y además logra distinguir si se trata del robot o de otro factor (obstáculo), pero estaría limitado a un único elemento, bien robot o bien obstáculo. El proyecto continúa entonces con el desarrollo del programa hacia la capacidad de buscar e identificar simultáneamente estos dos elementos y, en caso de encontrar ambos, calcular la distancia entre ellos.

La siguiente y última fase del proyecto se trata de incorporar el código que se ha ido desarrollando al robot. Es necesario entonces integrar el programa encargado de calcular la distancia con otro encargado del control del robot, tanto de la trayectoria como de la velocidad. El primero envía un dato en función de la distancia a la que se encuentre el robot del obstáculo que es recibido por el segundo y utilizado para regular la velocidad.

4.4 PLANIFICACIÓN

De manera orientativa y aproximada, se propuso el cronograma descrito en la Tabla 4-1 y representado en la Ilustración 4-2, donde el color más oscuro representa la estimación inicial de tiempos acorde con la información de la tabla y el color más claro se corresponde con el reparto real del tiempo llevado a cabo.

<i>FECHA</i>	<i>OBJETIVO</i>	<i>DESCRIPCIÓN</i>
Enero 2021	Familiarizarse con la cámara LIDAR y el entorno de trabajo MATLAB	Hasta esta fecha la idea es indagar y conocer de qué funciones dispone MATLAB para obtener y tratar las imágenes obtenidas con la cámara L515, centrándose en las nubes de puntos
Febrero 2021	Identificar superficies	Con la ayuda de las funciones que MATLAB pone a la disposición del usuario, segmentar la uniforme nube de puntos inicial en las distintas superficies (objetos) que componen la imagen.
Marzo 2021 (01/03-21/03)	Detectar obstáculos	Partiendo de un fondo previamente capturado, identificar elementos que entren en escena y no pertenezcan a dicho fondo.
Marzo/abril 2021 (22/03-14/04)	Segmentar robot	Con el brazo en movimiento, tenerlo identificado en todo momento.

Abril 2021 (15/04-30/04)	Integración con el robot	Desarrollar un código que calcule las distancias entre el brazo robótico y los obstáculos. Integrarlo con otro código que controle la velocidad del brazo en función de dichas distancias.
Mayo 2021 (01/05-15/05)	Pruebas y mejoras	Pruebas en la minifábrica para comprobar el funcionamiento deseado y mejorar o corregir fallos en caso de ser necesario.

Tabla 4-1. Cronograma de objetivos

TAREA	ENERO 2021				FEBRERO 2021				MARZO 2021				ABRIL 2021				MAYO 2021				JUNIO 2021				
	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	S22	S23	S24	S25
Familiarizarse con la cámara LIDAR y el entorno de MATLAB																									
Identificar superficies																									
Detectar obstáculos																									
Segmentar robot																									
Integración con el robot																									
Pruebas y mejoras																									

Ilustración 4-2. Cronograma de objetivos

4.5 RECURSOS UTILIZADOS

A continuación, se enumeran los recursos necesarios para llevar a cabo el proyecto:

- Intel RealSense LiDAR Camera L515 (representado en la Ilustración 4-3). Proporcionado por la universidad. Incluye un trípode y el cable USB tipo A a tipo C necesario para conectar la cámara al ordenador.



Ilustración 4-3. Aspecto del sensor LIDAR utilizado [36]

- Robot industrial ABB IRB120 y su entorno de trabajo (representados en la Ilustración 4-4). Disponibles en la minifábrica de ICAI.

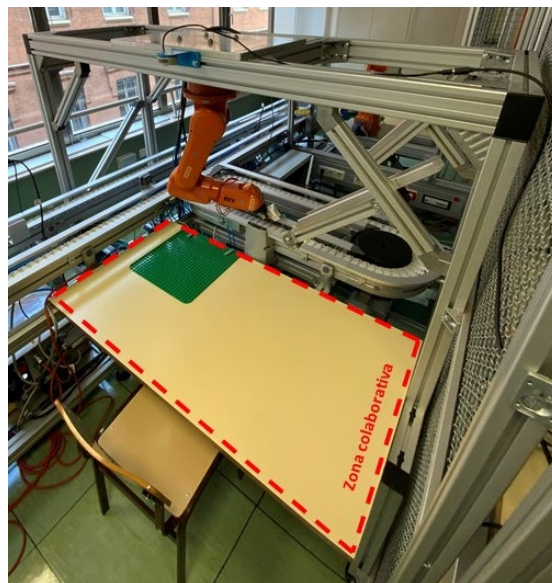


Ilustración 4-4. Robot y entorno de trabajo utilizados

- Ordenador con MATLAB. Para desarrollar el software del proyecto.

Adicionalmente, para acoplar la cámara LIDAR al entorno de trabajo será necesario hacerse con un soporte, por ejemplo, a partir de impresión 3D.

- Impresora 3D. Disponible en el departamento de ingeniería mecánica.

Capítulo 5. SISTEMA DESARROLLADO

El sistema encargado de garantizar un entorno de trabajo seguro consta de tres elementos principales, relacionados entre sí unidireccionalmente como representa la Ilustración 5-1:



Ilustración 5-1. Diagrama de bloques general

5.1 PROGRAMA DE MONITORIZACIÓN DEL ENTORNO

En este apartado se van a explicar los pasos seguidos para el desarrollo del programa encargado de monitorizar el entorno de trabajo. Se trata de, inicializaciones aparte, un bucle que se va repitiendo siempre que el programa esté encendido. Este realiza en cada iteración un análisis del entorno: comprobar si coexisten robot y obstáculo y en caso afirmativo calcular la distancia, estableciendo en consecuencia la velocidad del robot. La Ilustración 5-2 muestra el flujo seguido en cada una de estas iteraciones.

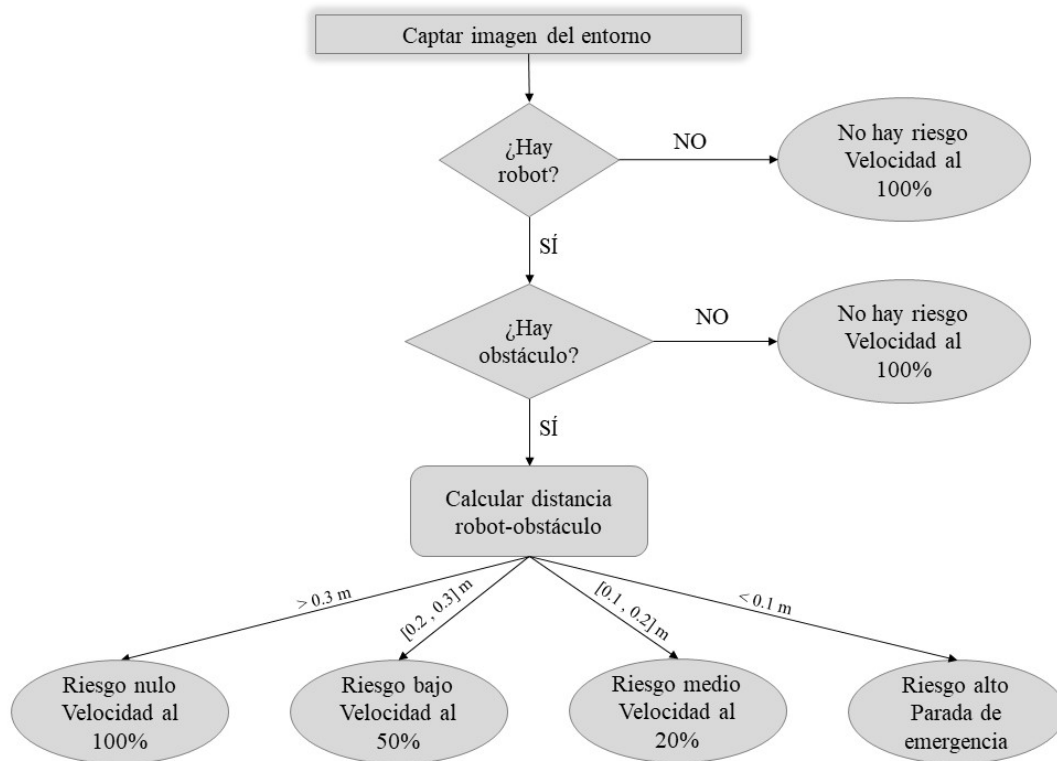


Ilustración 5-2. Estructura iterativa del programa

5.1.1 INICIALIZACIÓN

El programa debe inicializarse para cubrir ciertas medidas necesarias previas a entrar en el bucle destinado a la monitorización del entorno.

5.1.1.1 Conexión TCP/IP

Las siglas provienen del inglés *Transmission Control Protocol/Internet Protocol* y es el paso encargado de establecer la conexión con el robot. Para ello se han de introducir tanto su dirección IP (192.168.56.93) como el puerto por el cual se desea establecer la comunicación (1024).

5.1.1.2 Conexión con el LIDAR

Seguidamente se establece la conexión con la cámara. Es aquí donde se indica qué datos se requieren recoger del LIDAR, así como las características técnicas deseadas de cada uno de ellos:

- Profundidad: nube de puntos con una resolución de 640x480 píxeles a 30 FPS. Esto supone una nube de puntos compuesta por 307200 puntos (el producto), que puede estructurarse de dos maneras:
 - Organizada: un tensor de 480x640x3 de manera que las dos primeras dimensiones hacen referencia a los píxeles de la imagen y la tercera está destinada a almacenar las 3 coordenadas de los puntos.
 - Desorganizada: vector de 307200x3 que indica las coordenadas de cada punto, perdiendo la referencia directa de a qué píxel se corresponde cada uno. Si bien su distribución no es aleatoria, sino que guarda una relación con la nube de puntos organizada, tal como se expone en la Ilustración 5-3: los puntos se colocan ordenadamente por filas.

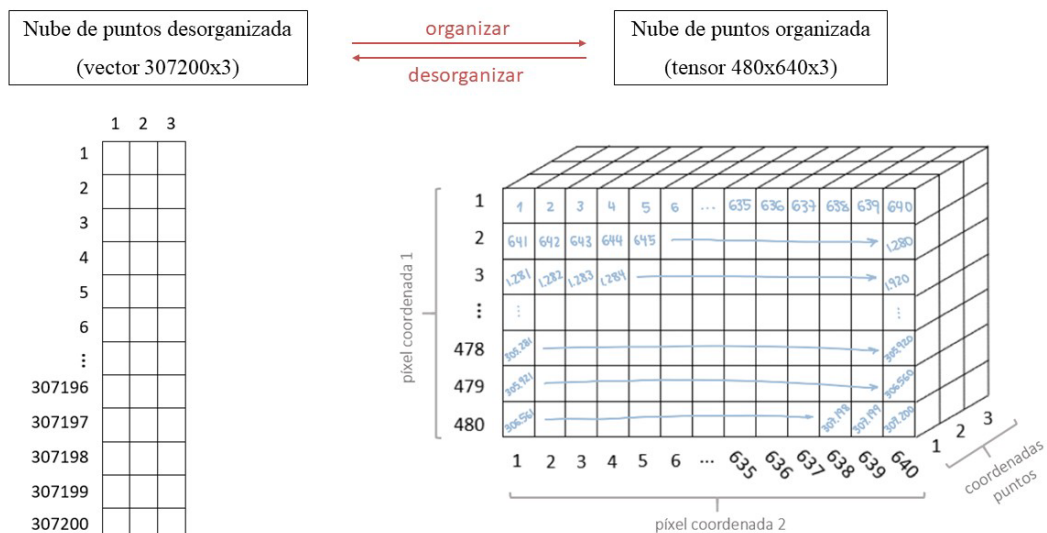


Ilustración 5-3. Estructuras nube de puntos

Aunque la captura del LIDAR se obtiene de manera desorganizada, dada su relación con la organizada será sencillo pasar de una estructura a otra. Con cuál de ellas se trabaja en cada momento dependerá las necesidades de cada proceso.

- Color: imagen RGB con misma resolución y velocidad

Con el fin de obtener un equilibrio entre precisión (mayor a mayor número de puntos) y rapidez (mayor a menor número de puntos), se configura el LIDAR para obtener una imagen, tanto de color como tridimensional, de 640x480 píxeles. Será la misma para ambas imágenes con el objetivo de facilitar el posterior procesamiento de los datos, de manera que a cada punto de la nube de puntos se le corresponda un código de color RGB.

Cabe destacar que, aunque la idea inicial es trabajar con dicha resolución, dado que el procesamiento de los datos consume más tiempo a medida que aumenta la dimensión de la nube de puntos, se prueba a capturar estas imágenes también con menor resolución. Se escoge la mínima permitida por la nube de puntos: 320x240 píxeles, es decir, la mitad en cada coordenada, lo que supone reducir el número de puntos a un cuarto. Aunque la mínima resolución con la que se puede recoger la imagen RGB es de 640x480, se reduce fácilmente para lograr la útil correspondencia entre puntos y píxeles comentada.

La elección de la resolución no influye en el desarrollo del programa que se explica a lo largo de todo este Capítulo 5 así que no se comenta de nuevo hasta alcanzar el análisis de resultados en el Capítulo 6.

La decisión de trabajar con estas dos imágenes se debe, en primer lugar, a la precisión de la nube de puntos a la hora de representar el entorno, la facilidad para calcular distancias gracias a que se conocen las 3 coordenadas de cada punto y los buenos resultados obtenidos en cuanto a *clustering* con este tipo de datos. Por otro lado, la imagen RGB es fundamental para conseguir segmentar el robot en base a su color, y aunque este es el único motivo de su uso, es imprescindible.

5.1.1.3 Captura del fondo

La primera captura de lo que percibe el LIDAR que se hace nada más encender el programa es la destinada a registrar el entorno habitual de trabajo. Este fondo se guardará en una variable (dos realmente, una para la nube de puntos y otra para la imagen RGB: fondo_xyz y fondo_rgb respectivamente) para poder utilizarlo en las iteraciones que se avecinan.

Por ello, es fundamental que antes de encender el programa el entorno se encuentre sin el robot y en el estado en que se va a trabajar. Además, posteriormente no se debe cambiar la distribución o añadir/retirar objetos (excluyendo el robot o los obstáculos), a no ser que se reinicie el programa.

La Ilustración 5-4 muestra el fondo empleado a lo largo del trabajo, tanto la imagen RGB como la nube de puntos, compuesto por una mesa con una región acondicionada para la colocación de piezas lego.

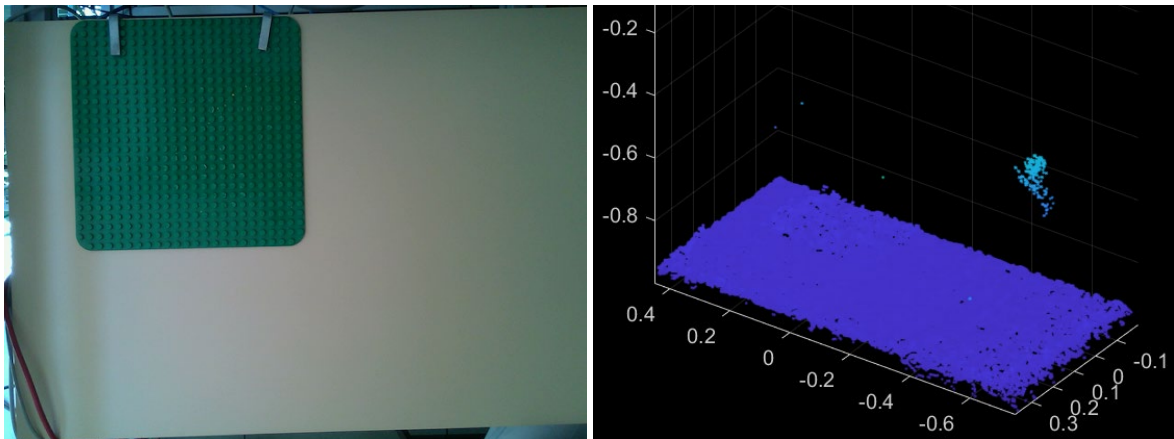


Ilustración 5-4. Fondo empleado en el proyecto

Los puntos que aparecen cerca de la esquina superior derecha de la mesa, aunque en la imagen no aparezca ningún elemento al que puedan corresponderse, se deben a un elemento que queda fuera del alcance de la imagen a color, pero no de la imagen de profundidad. Como se ha visto en el apartado 2.1, la cámara RGB tiene un campo de visión ligeramente inferior al del sensor LIDAR, lo que explica estas diferencias.

Sin embargo, la nube de puntos mostrada en la imagen no es la que proporciona el LIDAR directamente (Ilustración 5-5), sino la resultante de seleccionar el área correspondiente a la mesa. Para ello, se pasa la nube de puntos original por un filtro encargado de eliminar todos aquellos puntos que se encuentren por debajo de la mesa o fuera de ella, es decir, con coordenadas x superiores a 0.45, y superior a 0.4 o inferior a -0.2 o z inferior a -1 (Ilustración 16). Esta medida se toma para todas las nubes de puntos capturadas, no para el fondo en exclusiva.

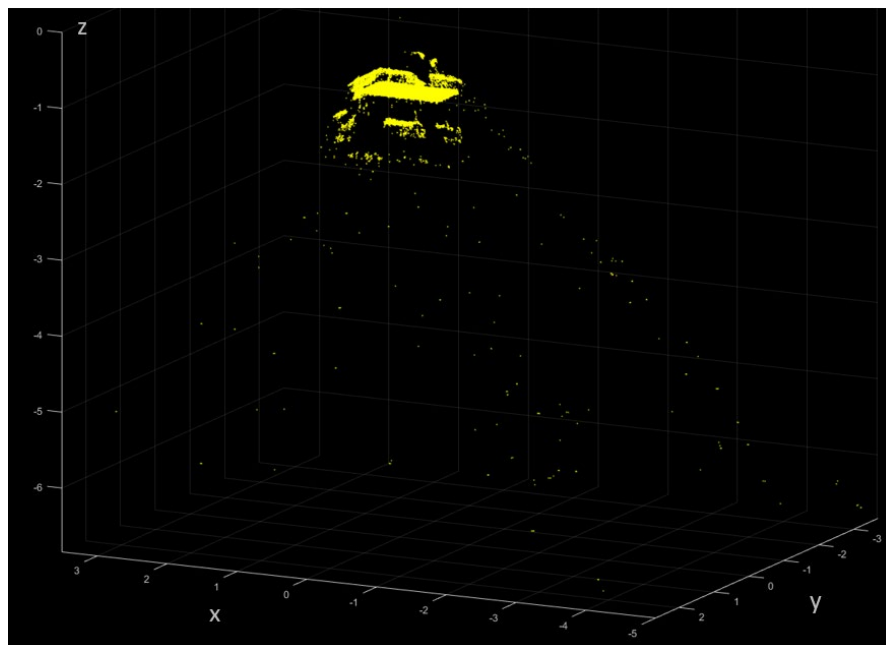


Ilustración 5-5. Nube de puntos original del LIDAR

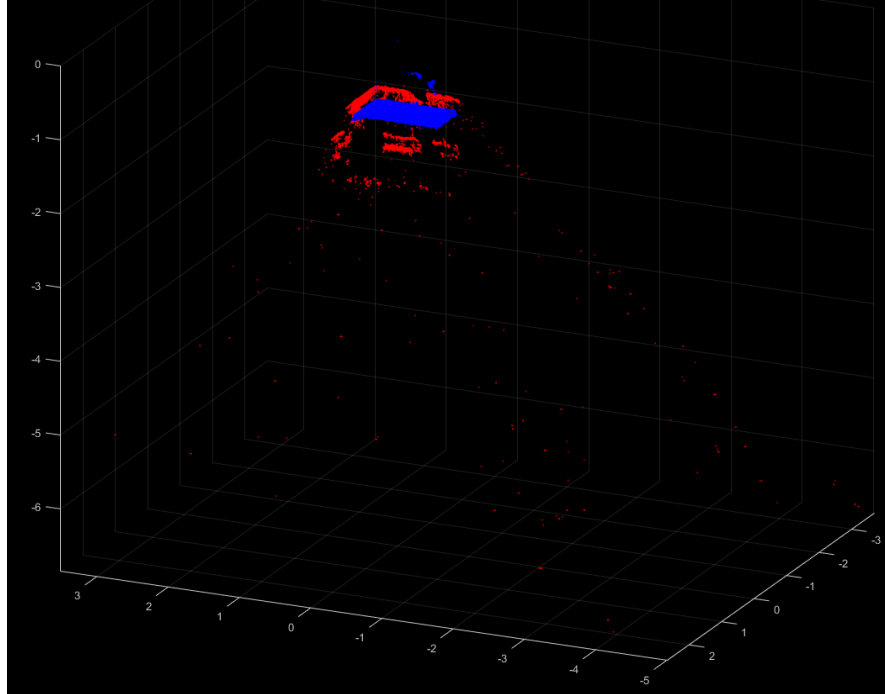


Ilustración 5-6. Puntos a eliminar para seleccionar la mesa

Esta modificación de la nube de puntos se debe a que en las pruebas iniciales del programa se observó en las zonas eliminadas un ruido excesivo, debido bien a su lejanía, a su poca definición o a la tendencia de estas zonas a sufrir ligeros cambios involuntarios. Esto supone que aquí los puntos varían considerablemente su posición en cada captura del LIDAR, perjudicando el posterior procesamiento de los datos. Además, esta medida no supone una pérdida de datos importante, ya que se trata de regiones a las que el robot no alcanza o que no comparte con el factor humano.

En este momento el sistema está listo para comenzar el bucle recién representado en la Ilustración 5-2.

5.1.2 CAPTURAR IMAGEN DEL ENTORNO

En cada iteración del bucle el programa toma una nueva imagen (RGB y nube de puntos) del entorno, la cual se guarda en una variable que se va sobrescribiendo (*img_rgb* e *img_xyz* respectivamente).

En la Ilustración 5-7 se puede ver un ejemplo de cómo se encuentra el entorno de trabajo en uno de los casos más complejos: con el robot y un obstáculo. A partir de esta situación se va a ir ejemplificando el comportamiento del programa en los apartados siguientes.

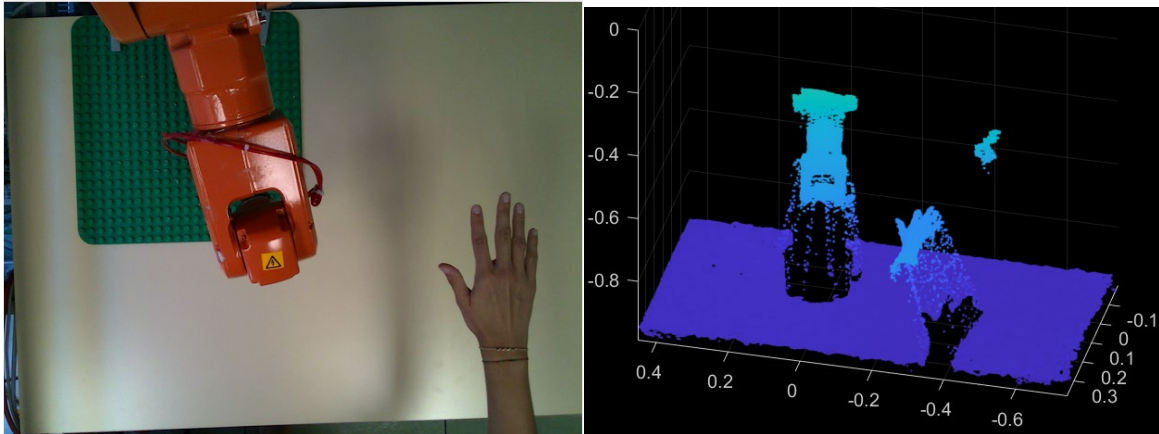


Ilustración 5-7. Situación del entorno de trabajo

5.1.3 DETECCIÓN DEL ROBOT

Lo primero que se va a buscar en esta imagen es la presencia del robot ya que, si no se encuentra en el entorno de trabajo compartido, sino que está operando en una zona donde accede él exclusivamente, no habrá ningún peligro potencial. Se podrá entonces establecer su velocidad máxima finalizando así la iteración.

Con el fin de lograr esta detección se parte de las nubes de puntos de la nueva imagen capturada y la del fondo, aplicando una serie de filtros y siguiendo los pasos mostrados en la Ilustración 5-8, detallados a continuación.

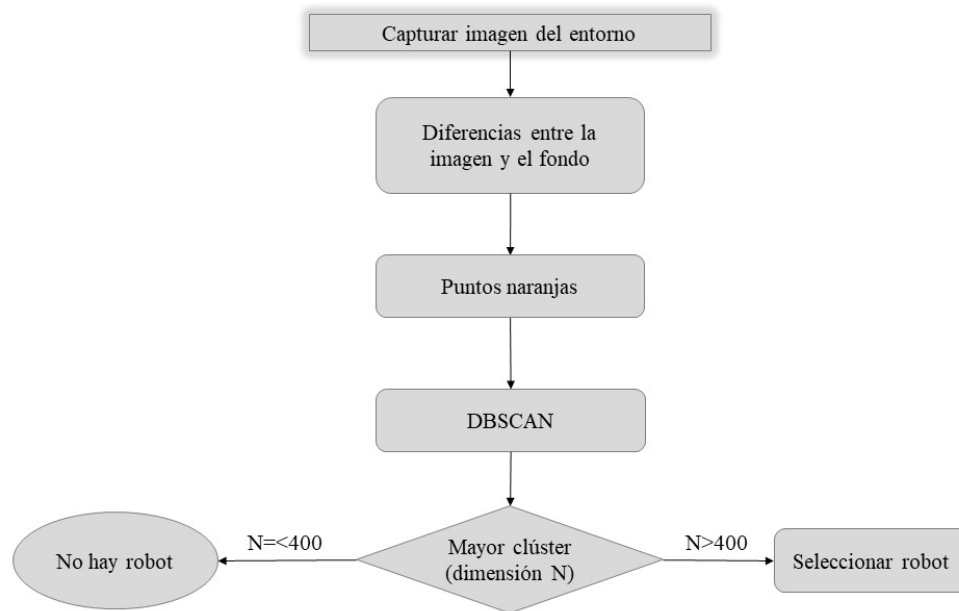


Ilustración 5-8. Flujo de operaciones para detectar el robot

5.1.3.1 Diferencias imagen-fondo

Se compara la nube de puntos de la nueva imagen del entorno con la del fondo, seleccionando aquellos puntos que sean distintos. Para ello se trabaja con las nubes de puntos desorganizadas o en forma de vector, ya que esta estructura permite hacer la resta de ambas. Se estudia entonces el resultado de la resta punto a punto: si el módulo es inferior a un máximo, se considera que el punto no ha cambiado de sitio; de lo contrario el punto será seleccionado.

Este máximo se ha establecido experimentalmente, es decir, capturando el fondo y un par de imágenes con nuevos elementos y probando para distintos valores con cuál se seleccionaban con mayor precisión los puntos que discrepaban del fondo.

En la Ilustración 5-9 se muestra el resultado, donde los puntos azules son la imagen captada originalmente y los amarillos los que han sido detectados como variaciones del fondo.

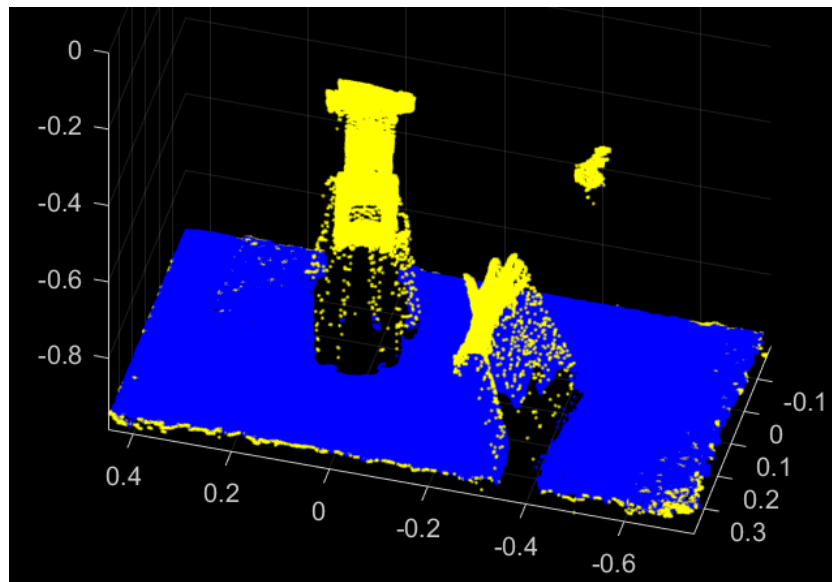


Ilustración 5-9. Resultado diferencias imagen-fondo

Si bien cabe destacar la presencia de ruido en esta selección, pues no resultan únicamente los puntos correspondientes a un nuevo elemento en el entorno, si no que aparecen también puntos que no han cambiado realmente de lugar, pero si lo suficiente en la nube de puntos como para ser escogidos por el filtro. Esto se solucionará más adelante al aplicar DBSCAN.

5.1.3.2 Filtro naranja

Aprovechándose de la apariencia característica del robot, su color naranja, se aplica sobre los puntos seleccionados en el paso anterior un filtro para escoger únicamente aquellos con dicha tonalidad.

Previamente se realiza un estudio, tanto teórico (ver Tabla 5-1) como experimental, acerca de los códigos RGB asociados a distintos colores dentro de la gama naranja.

<i>COLOR</i>	<i>CÓDIGO RGB</i>
 Naranja	(100, 50, 0)

	Naranja intenso	(93, 49, 15)
	Ámbar (ECE/SAE)	(100, 49, 0)
	Naranja pálido	(100, 46, 8)
	Anaranjado	(100, 40, 0)
	Rojo claro anaranjado	(97, 37, 15)
	Llama	(96, 59, 13)
	Amarillo indio	(96, 59, 24)
	Ámbar	(89, 58, 23)

Tabla 5-1. Códigos RGB de distintas tonalidades naranjas [37]

Por otro lado, el estudio experimental consistió en tomar un par de capturas del robot, visualizar sus imágenes de color en MATLAB (comando *imshow*) y con ayuda de la herramienta *Data tips* ir seleccionando puntos pertenecientes al robot para ver entre qué valores se movían los parámetros rojo (R), verde (G) y azul (B). Este estudio se considera más acertado ya que las condiciones lumínicas de la sala influyen en el color captado por la cámara.

Finalmente, se estableció el siguiente rango de colores para asegurar la selección de gran parte del robot:

- R entre 50 y 190
- G entre 10 y 90
- B entre 0 y 70

Se seleccionan entonces los puntos que cumplan estas 3 condiciones de color. Siguiendo el ejemplo visual que se viene representando, la Ilustración 5-10 muestra el resultado de este paso.

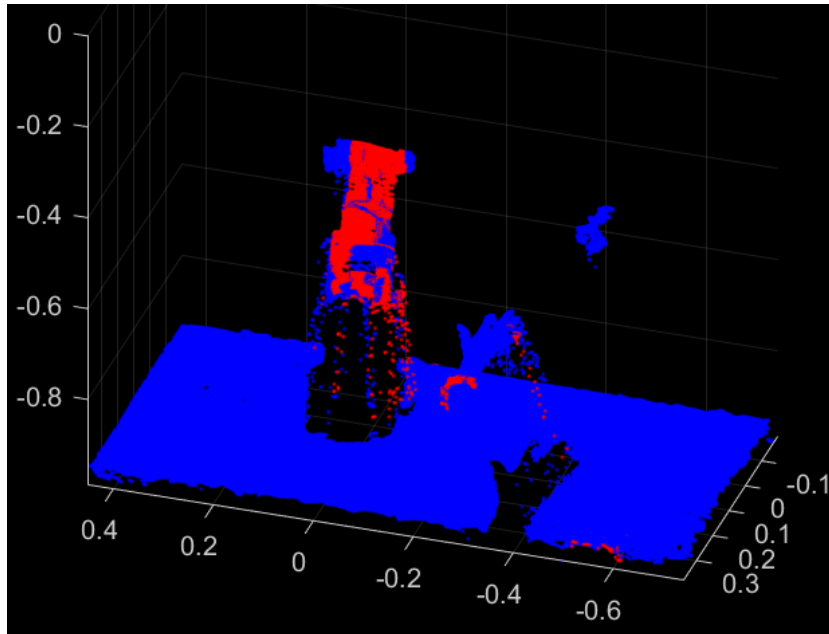


Ilustración 5-10. Resultado de seleccionar puntos anaranjados

5.1.3.3 DBSCAN

En la Ilustración 20 se pueden observar los principales inconvenientes que se presentan: hay puntos ajenos al robot que son seleccionados y hay puntos del robot que no. El primer problema se soluciona pasando el conjunto de datos por el algoritmo DBSCAN; la solución tomada para el segundo no siempre será necesaria y se mencionará más adelante (apartado 5.1.3.5).

Tal y como se explica en la sección 3.3.4, este algoritmo cuenta con dos parámetros de entrada introducidos por el usuario, trabajando en este proyecto con los siguientes valores:

- Épsilon: 0,05 ya que al tratarse de la representación en nube de puntos de elementos macizos (como por ejemplo una mesa, una mano, un robot, una consola...) la proximidad entre los puntos es muy pequeña.

- Puntos mínimos: 50, dado que experimentalmente se ha concluido que las nubes de puntos asociadas a los objetos de interés con los que se va a tratar (como los recientemente mencionados a modo de ejemplo) superan con creces la dimensión de 50 puntos.

Se pasa entonces el conjunto de puntos anaranjados obtenido por DBSCAN para eliminar el ruido y/o segmentarlo en los distintos grupos naranjas que pueda haber en el entorno, resultando un determinado número de clústeres que puede ir variando en cada iteración. Esta variación no supone ningún problema ya que para detectar el posible robot el programa se quedará con el clúster de mayor dimensión. En la Ilustración 5-11 se identifican los distintos clústeres (tres en este caso: rojo, verde y amarillo) y el ruido (blanco) en los que se han segmentado los datos; mientras que en la Ilustración 5-12 se puede apreciar la eficacia de asociar el mayor clúster al robot.

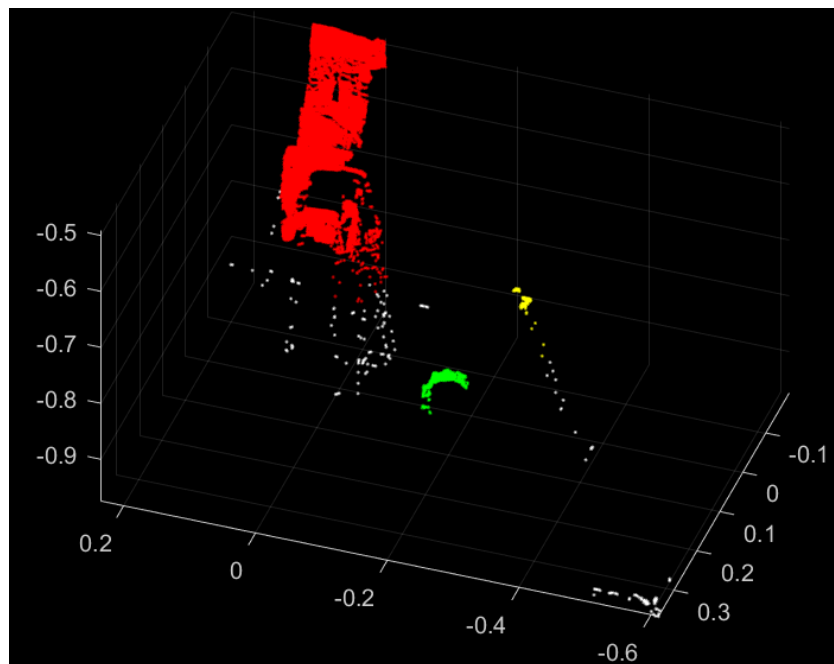


Ilustración 5-11. DBSCAN sobre los puntos naranjas

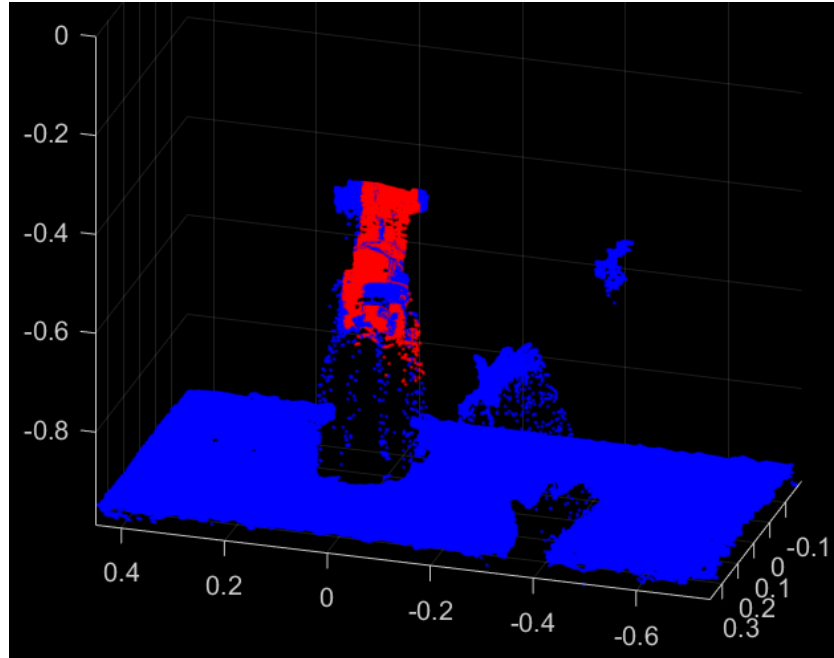


Ilustración 5-12. Asociación del mayor clúster con el robot

Cabe mencionar aquí la recomendación de evitar vestir de naranja si se va a acceder al entorno de trabajo para no dar lugar a confusión. También debería evitarse introducir en escena elementos voluminosos naranjas por la misma razón. Si bien se trata de pura precaución, ya que dadas las dimensiones del robot y la zona enfocada por el LIDAR su clúster debería de seguir siendo el de mayor tamaño.

5.1.3.4 Dimensión

Experimentalmente se ha llegado a la conclusión de que la nube de puntos del brazo robótico está compuesta por lo menos de 400 puntos, independientemente de con qué resolución se trabaje. Así pues, si el clúster de mayor dimensión resulta tener un número menor de puntos se concluye que el robot no está dentro del área de trabajo, estableciendo un nivel de riesgo nulo y finalizando la iteración actual.

En caso contrario, se considera que sí hay robot y se procede a solucionar el segundo problema mencionado en el apartado anterior. Para ello es necesario delimitar el robot con un rectángulo, lo cual se hace estudiando los píxeles correspondientes a cada uno de los

puntos del clúster previamente seleccionado. Dicho rectángulo viene delimitado por los valores mínimo y máximo de las dos coordenadas que definen los píxeles, como se ejemplifica en la Ilustración 5-13.

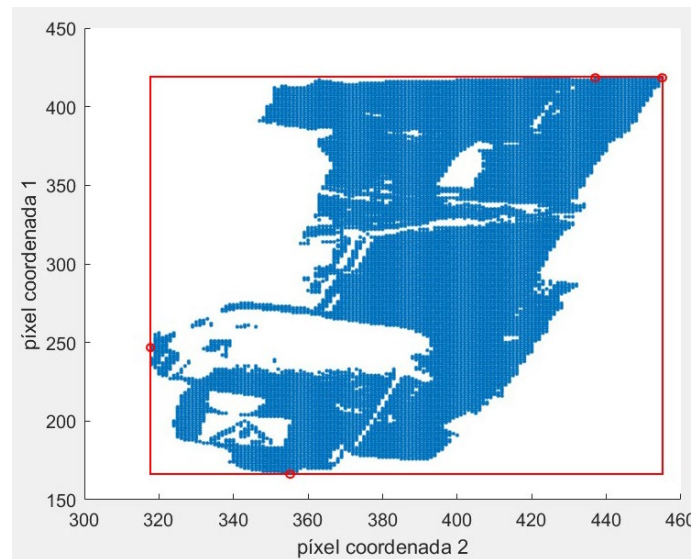


Ilustración 5-13. Píxeles asociados a los puntos naranjas del robot detectados

5.1.3.5 Seleccionar robot

En este paso se termina de seleccionar todo el robot. Resulta necesario organizar la nube de puntos de la imagen del entorno, para poder seleccionar fácilmente aquellos que se encuentren dentro del rectángulo recién calculado. Para asegurarse de que el robot esté comprendido por dicho rectángulo, se amplían sus lados en 20 píxeles, o lo máximo que se permita sin sobrepasar los límites de la imagen (de 1 a 480 para la primera coordenada; de 1 a 640 para la segunda). Se hace lo mismo para la imagen del fondo, tal y como se ejemplifica en la Ilustración 5-14.

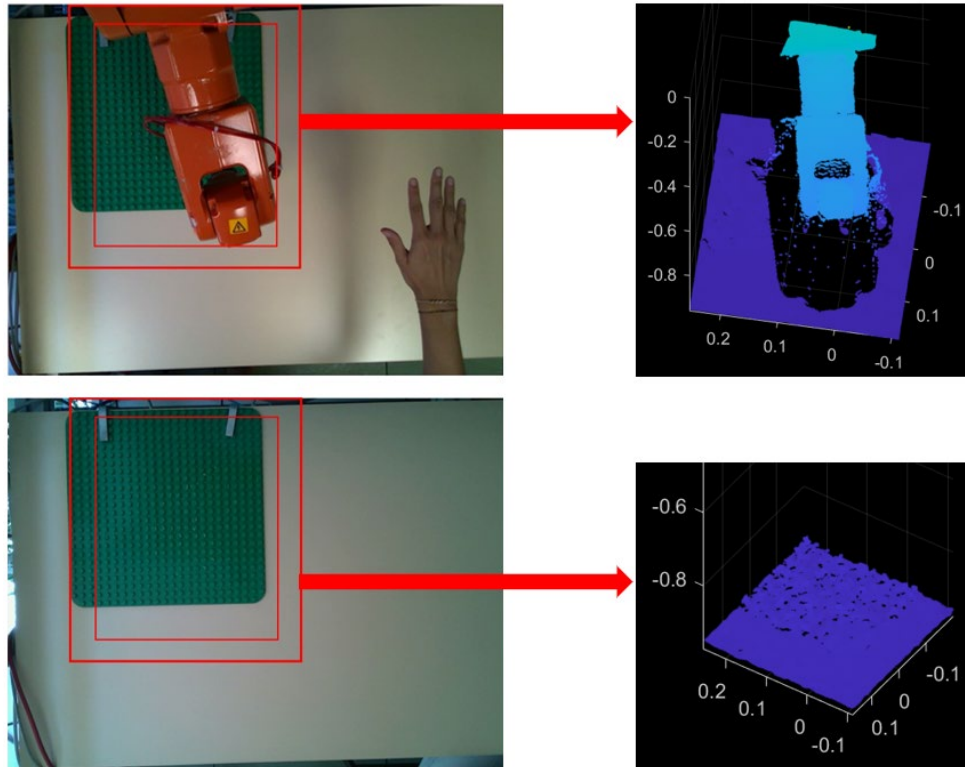


Ilustración 5-14. Selección de puntos dentro del recuadro

Seguidamente se desorganizan estos conjuntos de puntos para recuperar la estructura de vector. Se filtra de nuevo la nube correspondiente al robot para detectar los puntos que difieren del fondo (ver Ilustración 5-15) y a continuación estos se pasan por el algoritmo DBSCAN (ver Ilustración 5-16) para eliminar un posible ruido. Seleccionando el clúster de mayor dimensión se obtiene finalmente la nube de puntos asociada al robot. En la Ilustración 5-17 se representan en azul la nube de puntos asociada a la imagen del entorno, en rojo la del robot.

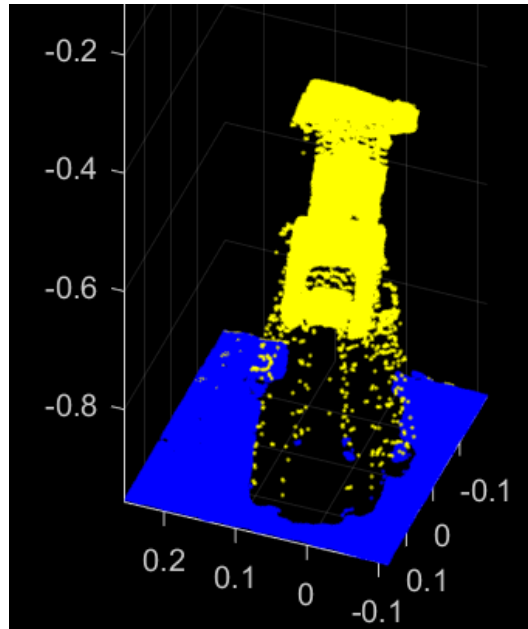


Ilustración 5-15. Diferencias imagen-fondo (dentro del rectángulo)

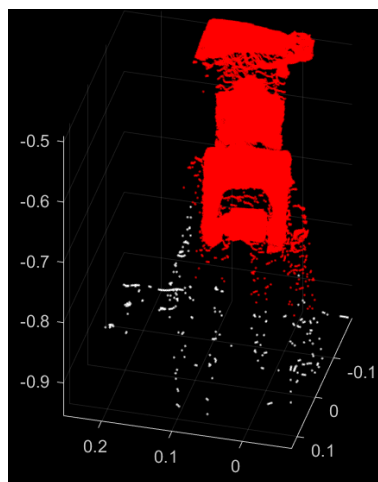


Ilustración 5-16. DBSCAN sobre los puntos en el marco del robot

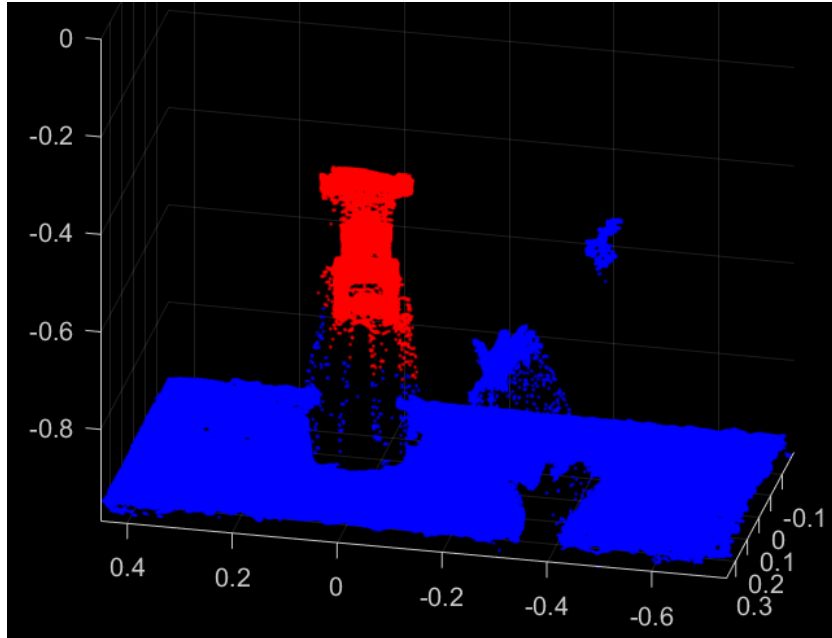


Ilustración 5-17. Robot detectado en la imagen del entorno

5.1.4 DETECCIÓN DE UN OBSTÁCULO

Una vez detectada la presencia del robot en el área de trabajo, interesa saber si además se encuentra una persona o un objeto con el que evitar la colisión. En caso negativo, la situación estará libre de riesgos, permitiendo al robot trabajar a su máxima velocidad y finalizando la iteración; en caso afirmativo, se continuará con el cálculo de la distancia entre estos dos elementos para poder determinar la velocidad idónea del robot.

La Ilustración 5-18 representa el flujo que sigue el programa para encontrar un obstáculo, detallado a continuación.

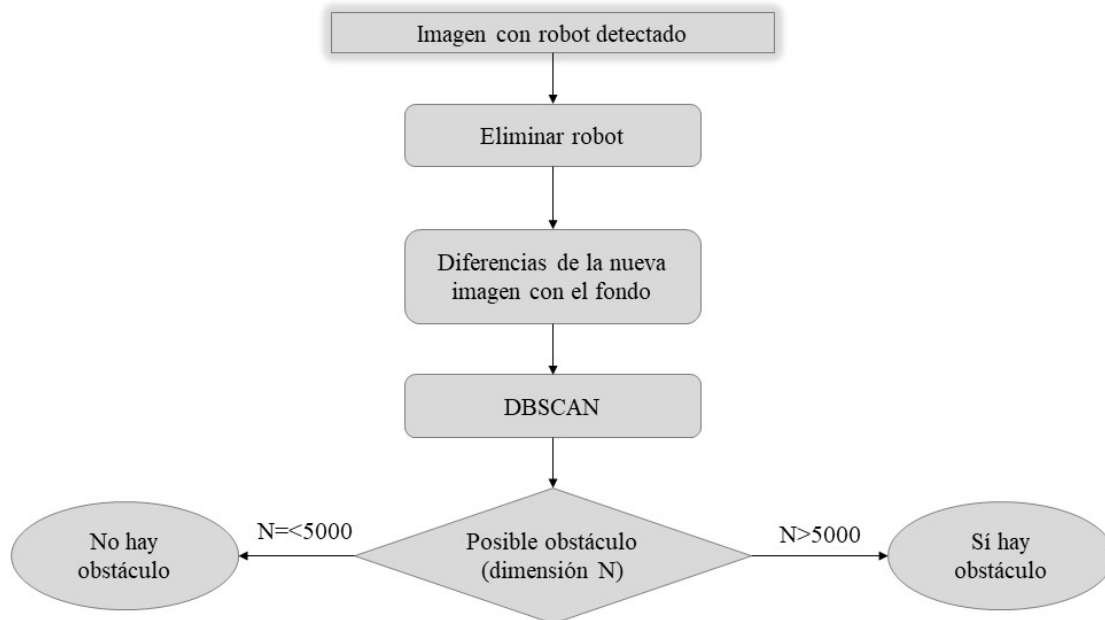


Ilustración 5-18. Flujo de operaciones para detectar un obstáculo

5.1.4.1 Eliminar robot

Con el fin de evitar categorizar al robot también como un obstáculo, debe ser eliminado de la nube de puntos capturada al comienzo de la iteración, resultando una nueva imagen.

Realmente esta operación se trata de una sustitución, pues cada punto perteneciente al robot es reemplazado por el punto del fondo que se encuentre en el mismo píxel, razón por la cual en este paso se vuelve a trabajar con la nube de puntos en forma de tensor (organizada), para conocer rápidamente la relación entre punto y píxel. El resultado de este paso viene representado en la Ilustración 5-19.

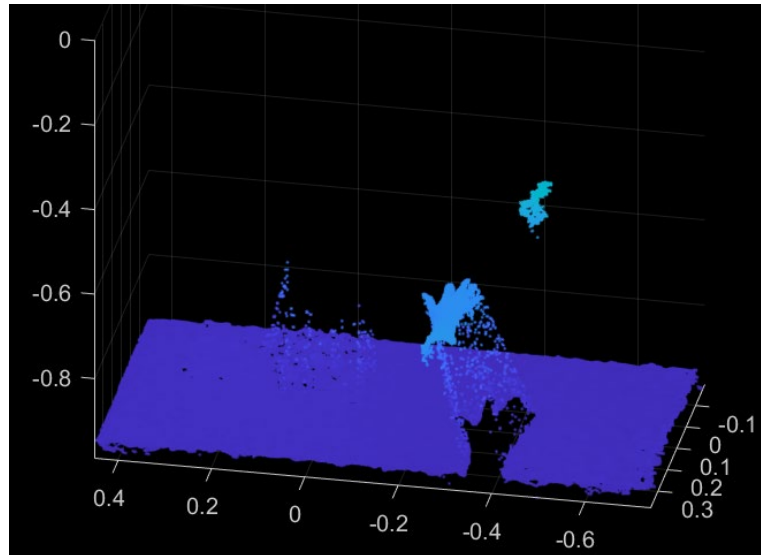


Ilustración 5-19. Nueva imagen sin el robot

5.1.4.2 Diferencias imagen-fondo

Se compara la nube de puntos de la nueva imagen con la del fondo, seleccionando aquellos puntos que sean distintos, tal y como explica el apartado 5.1.3.1. En la Ilustración 5-20 se muestra en azul el fondo y en amarillo dichos puntos.

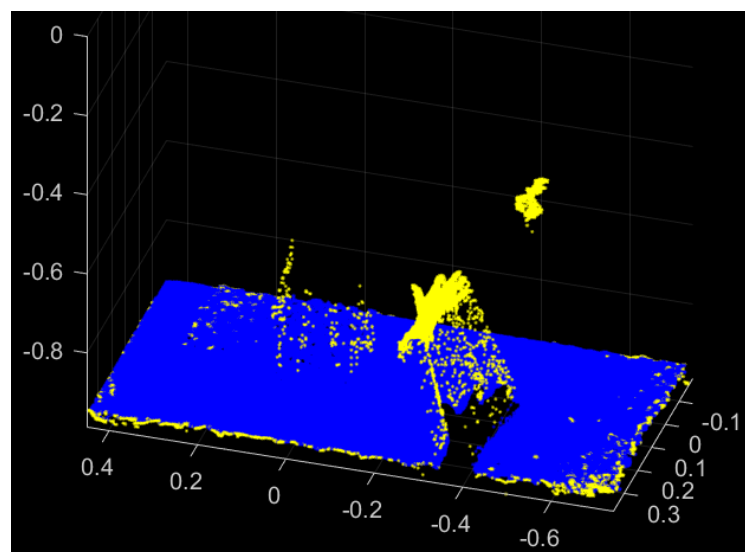


Ilustración 5-20. Diferencias de la imagen sin el robot con el fondo

5.1.4.3 DBSCAN

El conjunto de puntos obtenido se pasa por el algoritmo DBSCAN para segmentarlo y localizar el ruido, resultando un determinado número de clústeres que puede ir variando en cada iteración. De nuevo, esta variación no supone ningún problema ya que para detectar el posible obstáculo el programa se quedará con el clúster de mayor dimensión.

En el ejemplo que se está empleando, se detecta un único clúster (amarillo) y el ruido (rojo) como se muestra en la Ilustración 5-21.

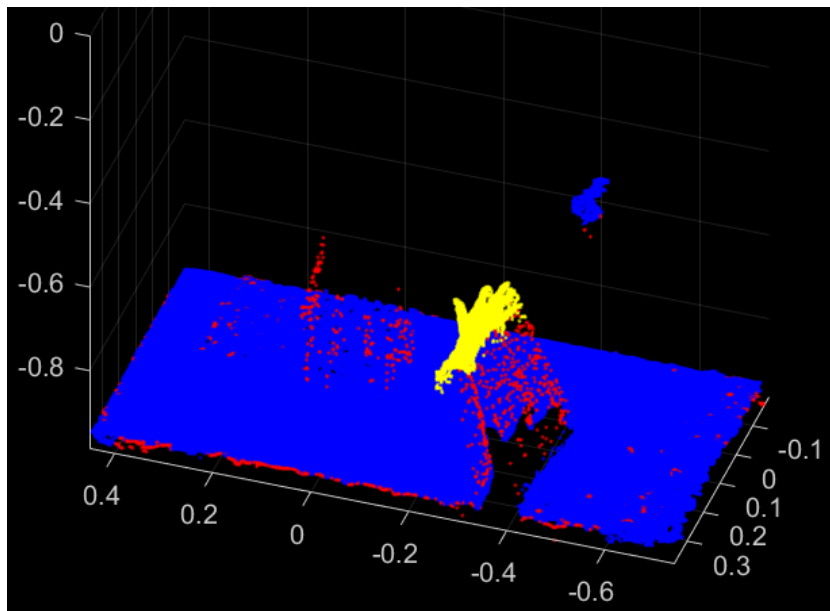


Ilustración 5-21. DBSCAN para detectar obstáculo

5.1.4.4 Dimensión

Experimentalmente se ha llegado a la conclusión de que la nube de puntos de un obstáculo está compuesta por lo menos de 5000 puntos, en caso de trabajar con una resolución de 640x480 píxeles; 1500 para 320x240. Así pues, si el clúster de mayor dimensión resulta tener un número menor de puntos se concluye que el robot se encuentra solo en el área de trabajo, estableciendo un nivel de riesgo nulo y finalizando la iteración actual.

En caso contrario, se considera que sí hay un obstáculo y este es asociado a la nube de puntos del clúster seleccionado anteriormente (como muestra la Ilustración 5-22), guardándolo en una variable para su posterior uso. Se prosigue entonces con la iteración.

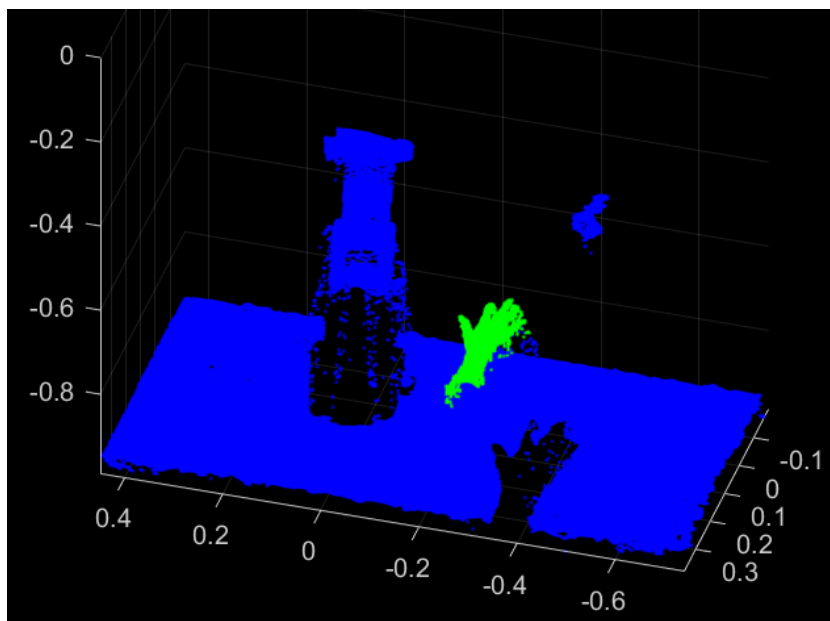


Ilustración 5-22. Distinción del obstáculo en la imagen del entorno

5.1.5 CÁLCULO DE DISTANCIA

Si se ha alcanzado este nivel del programa significa que tanto el robot como un obstáculo se encuentran en el área de trabajo, con lo cual es necesario adaptar la velocidad del robot en función de su distancia con dicho obstáculo.

Gracias a la precisión con la que la nube de puntos permite segmentar estos dos factores (ver Ilustración 5-23) y conocer las coordenadas tridimensionales de todos los puntos que los conforman, el cálculo de la distancia entre ellos resulta muy sencillo y acertado. Se calcula la distancia entre cada punto del robot con cada punto del obstáculo, tomando como resultado el menor valor. En base a esta se establece la velocidad del robot, expresada como un porcentaje de la máxima que puede alcanzar.

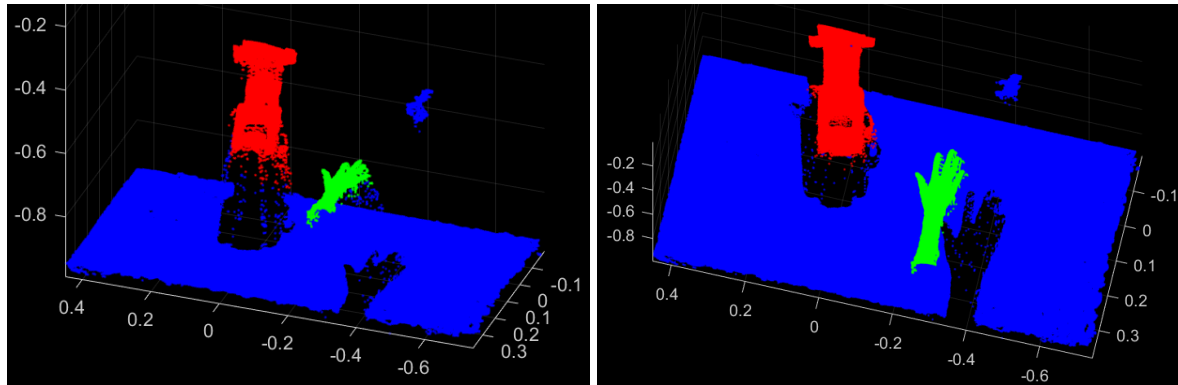


Ilustración 5-23. Diferenciación fondo-robot-obstáculo

Los niveles de riesgo y sus umbrales establecidos se muestran en la Tabla 5-2:

<i>NIVEL DE RIESGO</i>	<i>DISTANCIA [cm]</i>	<i>VELOCIDAD</i>	<i>IMAGEN</i>
NULO	Superior a 30	100%	Obstáculo a 30 cm del robot en Ilustración 5-24
BAJO	Entre 30 y 20	50%	Obstáculo a 20 cm del robot en Ilustración 5-25
MEDIO	Entre 20 y 10	20%	Obstáculo a 10 cm del robot en Ilustración 5-26
ALTO	inferior a 10	0% (Parada)	

Tabla 5-2. Niveles de riesgo

Cabe destacar que esta tabla hace referencia a cuando tanto robot como obstáculo aparecen en el área de trabajo. En caso de estar presente solamente uno de ellos o ninguno, el nivel de riesgo será nulo con una velocidad asociada del 100%.

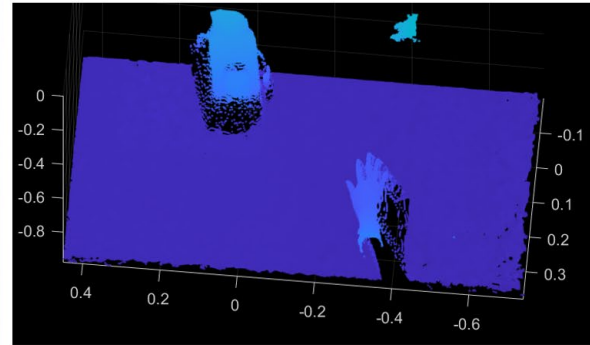
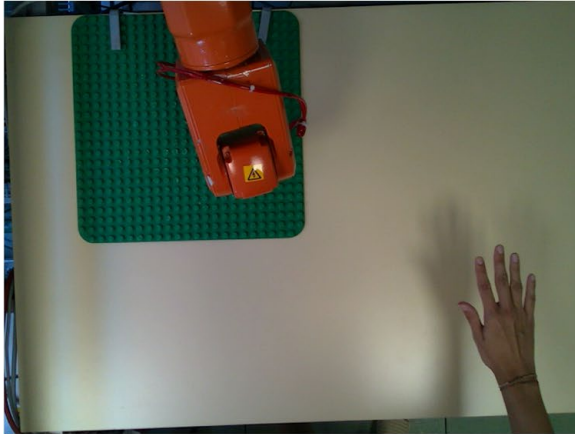


Ilustración 5-24. Distancia robot-obstáculo 30 cm

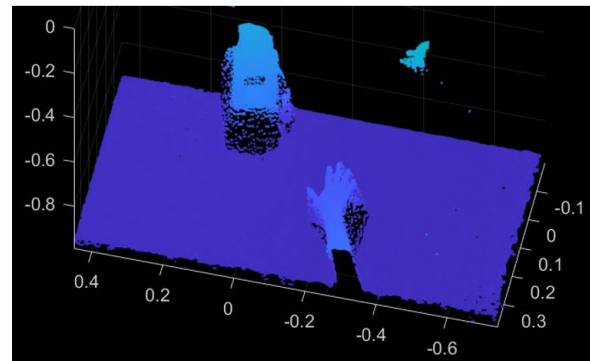
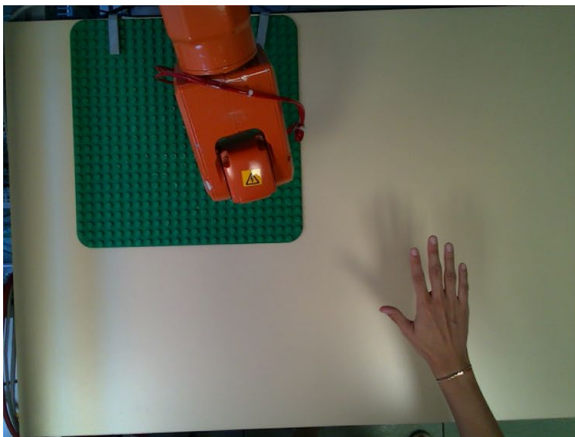


Ilustración 5-25. Distancia robot-obstáculo 20 cm

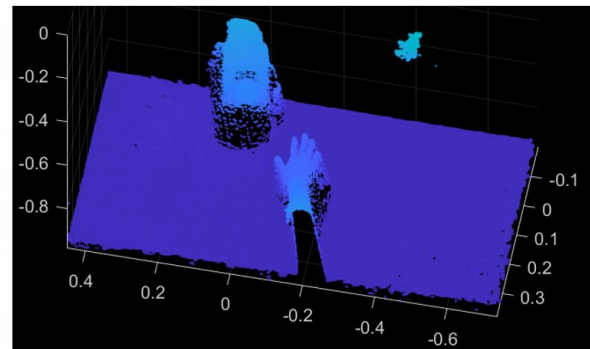
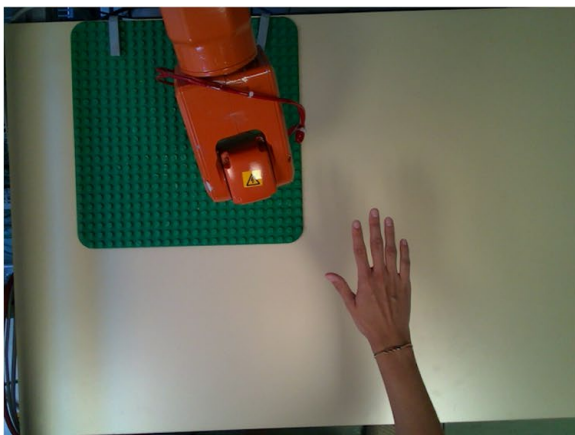


Ilustración 5-26. Distancia robot-obstáculo 10 cm

5.1.6 ENVIAR DATO

El dato que MATLAB envía al robot es directamente el valor del porcentaje de velocidad que se establece en función de la situación en la que se encuentre el área de trabajo.

5.2 *PROGRAMA PARA EL CONTROL DEL ROBOT*

En paralelo con el programa recién detallado se ejecuta otro encargado de controlar el movimiento y la velocidad del robot. Para este segundo programa se emplea RAPID, lenguaje de programación textual de alto nivel desarrollado por la empresa ABB para el control de sus robots.

Como punto de partida se utiliza el código desarrollado por Concepción Góngora en su proyecto de fin de grado [38], donde monitoriza el mismo entorno de trabajo objeto del presente documento con una cámara RGB mientras el robot describe una trayectoria preestablecida. Este se mueve reiteradamente entre los tres mismos puntos y adecúa su velocidad en función del dato que reciba. La estructura de este código con las modificaciones implementadas se detalla en los siguientes apartados.

5.2.1 DECLARACIÓN DE VARIABLES/CONSTANTES

Se declaran las variables y constantes globales necesarias a tratar en las distintas funciones del programa.

Las variables son las siguientes:

- ***server* y *client***: son variables de tipo *socketdev* (proveniente del inglés *socket device*-dispositivo zócalo), encargadas de almacenar la información correspondiente a los dispositivos involucrados en una comunicación. Se corresponden con el robot y el ordenador respectivamente.

- **encendido:** es una variable de tipo lógico (*bool*) que únicamente puede tomar los valores “verdadero” o “falso”. Así, está destinada a indicar si el programa está o no encendido. Se inicializa en “falso”.
- **alarma:** se trata de otra variable tipo *bool*, que indica si el robot está parado (hay alarma) o en movimiento (no hay alarma). También se inicializa en “falso”.
- **intEscucha:** es un tipo de dato (*intnum: interrupt numeric*) usado específicamente para detectar una interrupción.
- **receivedData:** es una cadena de caracteres (*string*), donde se va a almacenar la información recibida de MATLAB.

Por otro lado, se crean las siguientes constantes, todas ellas del tipo *robtarget*, es decir, destinadas a almacenar una posición del robot.

- **p1, p2, p3 y p4:** son las posiciones que irá repitiendo el robot para simular su operación habitual. La primera se encontrará fuera del área de trabajo compartida (donde el robot acude normalmente para recoger las piezas que necesite), es decir, fuera del alcance del LIDAR; mientras que las otras 3 sí están a su alcance.
- **posición_entreno_1 y posición_entreno_2:** necesarias para sacar el robot del alcance del LIDAR a la hora de capturar el fondo. Aunque el objetivo final es alcanzar el segundo punto, se ha considerado oportuno poner uno anterior para restringir la trayectoria del robot y evitar colisiones con el inmobiliario.

5.2.2 PROCESO PRINCIPAL (MAIN)

Es el bucle encargado de mover el robot por los 4 puntos establecidos siempre y cuando el contenido de la variable “encendido” sea “verdadero”.

En caso de ser “falso”, o sea, nada más ejecutar el programa (pues así se inicializa), se establece la conexión con el ordenador, se lleva el robot a la posición de entreno y se activan las interrupciones necesarias. Una vez finalizadas estas preparaciones, se cambia el valor de la variable “encendido” a “verdadero” para así la próxima vez que se entre en el bucle se comience con el movimiento habitual del robot.

5.2.2.1 Conexión robot-ordenador

Se asocia la variable “*server*” con el robot, asignándole su IP y el puerto, tal y como se hace en la conexión en MATLAB. Por otro lado, dado que para iniciar este programa es necesario haber iniciado antes el de MATLAB y este pide acceso, se le concede y se guarda en “*client*”.

5.2.2.2 Posición de entreno

Se lleva el robot a la posición de entreno, pasando por el primer punto y finalizando en el segundo para describir una trayectoria medianamente controlada libre de colisiones. En función de dónde se encontrase, tardará un tiempo distinto, pero sin sobrepasar en ningún caso los 15 segundos. Es por ello que en MATLAB, antes de capturar la imagen que será guardada como fondo, se establece una espera de 15 segundos, para así darle tiempo suficiente al robot para apartarse, asegurando que lo que se capture realmente se trata del fondo libre de elementos externos.

5.2.2.3 Habilitación de interrupciones

En primer lugar, se habilitan las interrupciones y seguidamente se conecta la variable “*intEscucha*” con la función destinada a la interrupción (TRAP) que se ha denominado *recibir_mensajes*.

Es en este momento cuando se cambia la variable “*encendido*” al valor “verdadero”, para no volver a realizar estas configuraciones, a no ser que se apague y se encienda de nuevo el programa.

5.2.3 INTERRUPCIÓN (RECIBIR_MENSAJES)

Es una función que se llama automáticamente cuando MATLAB envía un dato y es la encargada de, como su nombre indica, recibir el mensaje de MATLAB y adaptar su velocidad en base a este. Para ello primero se vacía la variable “*receivedData*” donde a continuación se guarda lo recibido por el ordenador.

Si se recibe un 0 (petición de parada), se para el robot con la función *Stopmove* propia de RAPID y se cambia la variable “alarma” a “verdadero” para indicar que el robot está parado.

Si se recibe otro valor se adaptará la velocidad a dicho valor introduciéndolo como argumento en la función *SpeedRefresh* propia de RAPID también. Si bien, previamente se debe comprobar si el robot estaba parado (alarma=verdadero) y de ser así reanudar su marcha con la función *StartMove*, propia de RAPID.

Inicialmente, el mensaje recibido se pasaba de *string* a número (con ayuda de la función *StrToVal*), con la intención de introducirlo directamente en la función *SpeedRefresh*. Sin embargo, el tipo de dato que devuelve *StrToVal* resulta no ser compatible con el que necesita recibir *SpeedRefresh* por lo que finalmente el cambio de velocidad se realiza a base de condiciones prescindiendo de *string* a número, como se muestra a continuación:

```
IF (receivedData='20') THEN
    SpeedRefresh 20;
ELSEIF (receivedData='50') THEN
    SpeedRefresh 50;
ELSEIF (receivedData='100') THEN
    SpeedRefresh 100;
ENDIF
```

Cabe destacar el empleo de comillas a la hora de comparar la variable “*receivedData*”, necesarias ya que se trata de una cadena de caracteres.

Antes de finalizar, se borra el contenido de la variable “*intEscucha*” y se vuelve a conectar con la interrupción para poder detectarla cuando vuelva a saltar.

5.3 SOPORTE PARA EL LIDAR

El sensor debe colocarse de tal manera que su campo de visión sea capaz de abarcar toda el área de trabajo compartida. Además, debe estar lo suficientemente alejada como para que los objetos no se acerquen a menos de 25 cm, ya que a partir de esa distancia dejaría de detectarlos según se ha visto en sus especificaciones técnicas (apartado 2.1). Si bien no puede

alejarse demasiado, pues más allá de los 9 m también dejaría de detectar, lo cual dadas las dimensiones de la sala en la que se realiza el proyecto no resulta restrictivo.

El LIDAR se coloca aprovechando la estructura que encierra al robot, mostrada en la Ilustración 5-27. Esta contiene unos raíles que ofrecen dos alturas y que permiten mover el soporte de izquierda a derecha. Se emplea el raíl superior por el alcance mínimo comentado de 25 cm.

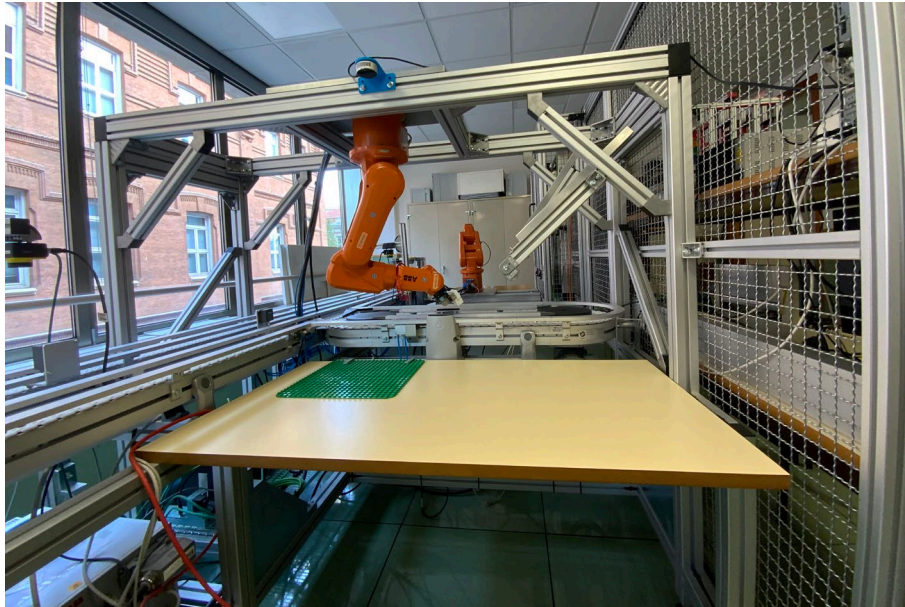


Ilustración 5-27. Estructura que encierra al robot

Se opta por los recursos de impresión 3D que pone la universidad a disposición del alumno para fabricar el soporte, resultando totalmente indiferente el método o el material empleado ya que, dada la ligereza del sensor, la pieza apenas va a soportar esfuerzo alguno.

5.3.1 PRIMER DISEÑO

El soporte tiene que contar con dos partes fundamentales: una destinada a su sujeción al raíl (con 2 pares de tornillos y tuercas) y otra a la sujeción del LIDAR (con 2 tornillos), como expone la Ilustración 5-28.

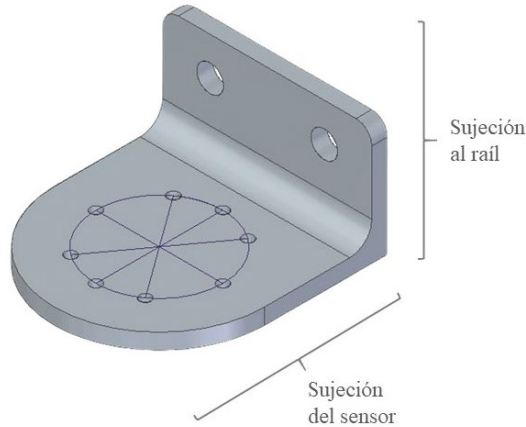


Ilustración 5-28. Primer diseño del soporte

Las restricciones tenidas en cuenta a la hora del diseño de esta pieza han sido las siguientes, para la parte dedicada al sensor:

- Agujeros del tipo 3M separados 40 mm, pues son las especificaciones de los agujeros que presenta el LIDAR para ser atornillado. Sin embargo, se prescinde de hacer los agujeros roscados porque no es estrictamente necesario y complica la impresión.
- Dejar por lo menos 10,5 mm entre el agujero más cercano a la parte dedicada al raíl y esta parte, pues la cámara tiene un diámetro de 61 mm y si no chocaría impidiendo alinear los agujeros del soporte con los del LIDAR.

Para la parte dedicada al raíl únicamente se ha tenido en cuenta el diámetro necesario de los agujeros para poder acoplar los tornillos disponibles en el laboratorio: 8,5 mm.

Se opta por unir ambas partes a 90° pues no se encuentran ventajas significativas al aportar ninguna inclinación, además de que esta podría distorsionar las distancias percibidas. Sí se plantean varias orientaciones de la cámara en el plano paralelo al suelo, de ahí los cuatro pares de agujeros presentes en el diseño.

Una vez impreso, se toma una captura (Ilustración 5-29) para discutir la validez del prototipo, obtenida colocando la cámara en los agujeros perpendiculares al raíl.

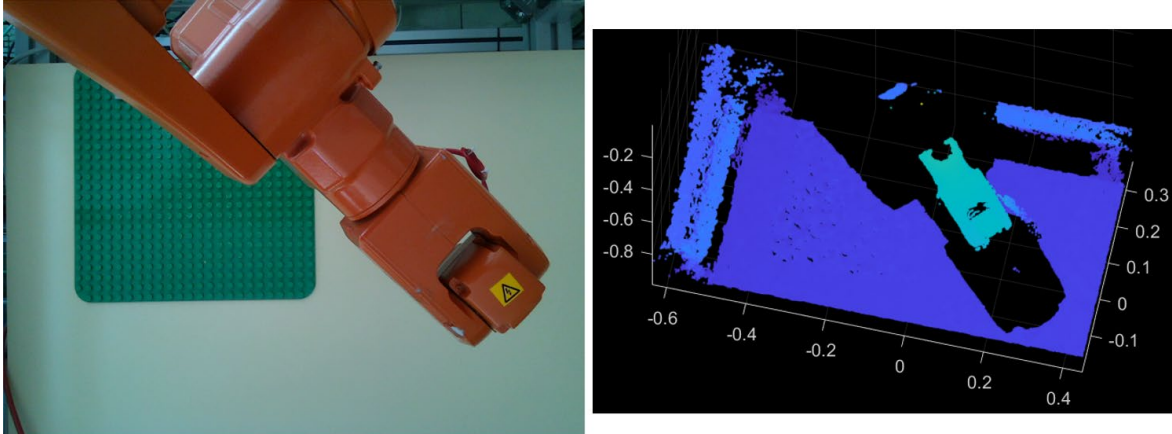


Ilustración 5-29. Enfoque del primer diseño de soporte

Se pueden observar dos inconvenientes en esta posición de la cámara:

- Se pierde parte de la mesa en su extremo más alejado del robot.
- Está demasiado cerca, perdiendo mucha información cuando el robot se encuentre en posiciones elevadas como la que muestra la imagen.

Con el fin de solucionarlos, se diseña un nuevo soporte que aleje y eleve más la cámara del raíl.

5.3.2 SEGUNDO DISEÑO

Este es el definitivo, el que se ha usado para capturar todas las imágenes incluidas en este capítulo y tiene el aspecto que muestra la Ilustración 5-30, donde únicamente se han acotado las medidas de imprescindible cumplimiento para un correcto acople del LIDAR al soporte y del soporte al raíl. El resto de las medidas, como el espesor o los radios de curvatura, no se indican ya que al no tener que cumplir con ningún requisito pueden ser aleatorias.

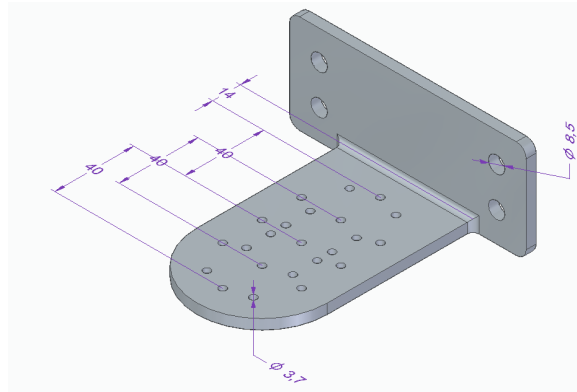


Ilustración 5-30. Segundo diseño del soporte

Cambios realizados en la parte de sujeción al raíl:

- Aumento de la separación entre agujeros, pues en el anterior diseño resultaba complicado enroscar las tuercas ya que había poco margen entre ellas.
- Inserción de un nuevo nivel para elevar la cámara. En caso de no resultar beneficioso se mantiene el nivel ya existente.

Cambios realizados en la parte para la sujeción de la cámara:

- Inserción de dos nuevos niveles para separarla del raíl con el fin de detectar toda la mesa.

Finalmente se coloca la cámara separándola y elevándola lo máximo posible del raíl empleando los agujeros indicados en la Ilustración 5-31 y con dicha orientación del soporte, obteniendo- como se comentaba- las imágenes mostradas a lo largo de todo el documento.

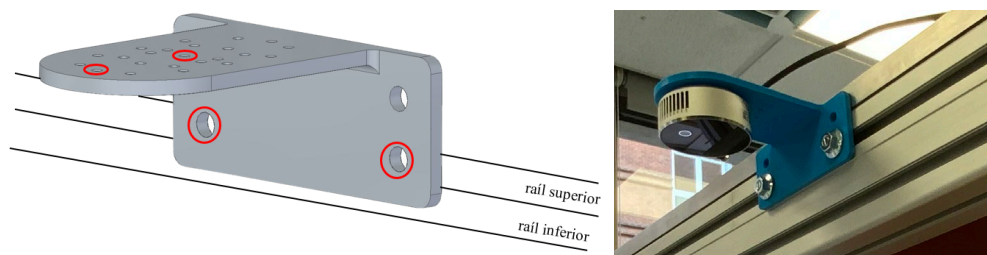


Ilustración 5-31. Disposición final del soporte

Capítulo 6. ANÁLISIS DE RESULTADOS

En esta sección se van a exponer los resultados obtenidos en una implementación real del proyecto poniendo en marcha el robot del laboratorio de la universidad y ejecutando los programas desarrollados. Se comentan además los inconvenientes encontrados tanto para el sensor utilizado como para el sistema de seguridad implementado.

6.1 COMPORTAMIENTO DEL ROBOT

En el siguiente enlace se puede acceder a un vídeo de la puesta en marcha del sistema:

https://upcomillas-my.sharepoint.com/:v:/g/personal/201702047_alu_comillas_edu/EQ2WjopoIgFMqP72AzeVPVQBraETjcVV3u3h1hISGn29zw?e=mipfEw

Además, la Ilustración 6-1 muestra una selección de los fotogramas más representativos de lo ocurrido en dicho vídeo desde la perspectiva de un observador externo, y la Ilustración 6-2 desde la perspectiva del LIDAR. El robot acude a sus dos posiciones de entrenamiento para permitir al programa captar una imagen del fondo. Pasado el tiempo establecido para ello, comienza su movimiento repetitivo entre los cuatro puntos fijados. Al introducir un obstáculo en la trayectoria del robot este reduce su velocidad al acercarse para finalmente detenerse con éxito sin colisionar.



Posición inicial aleatoria



Posición de entreno 1



Posición de entreno 2



Punto 1



Punto 2



Punto 3



Punto 4 y aparición del factor humano



Vuelta al punto 1



Introducción de obstáculo



Reducción de velocidad al 20%



Parada del robot



Reanudación de la marcha al 100%

Ilustración 6-1. Fotogramas significativos de la implementación. Perspectiva de un observador externo

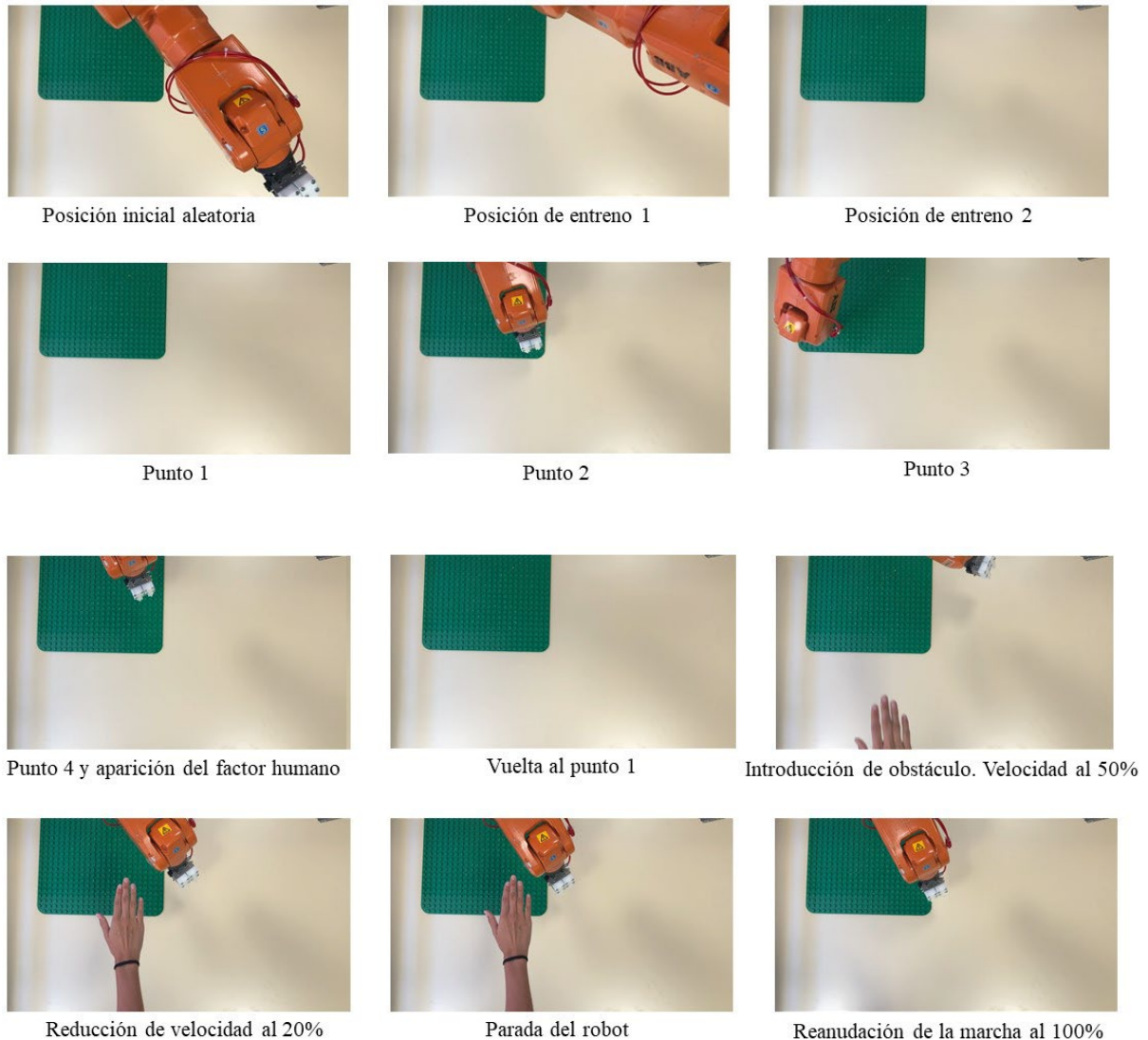


Ilustración 6-2. Fotogramas representativos de la implementación. Perspectiva del LIDAR

6.2 TIEMPOS DE COMPUTACIÓN

En el vídeo se aprecia que las distancias a las que suceden ambas reacciones del robot (frenar y detenerse) son inferiores a las establecidas; así como cuando se retira el obstáculo, puede llegar a transcurrir algún segundo hasta que se reanuda la marcha. Esto se debe a los tiempos de cómputo del programa de monitorización, pues entre que captura la imagen de una

situación y decide qué velocidad enviar, el robot ha seguido avanzando y recibe tarde la información.

El principal problema apreciado en esta implementación del proyecto es entonces el largo tiempo que emplea el robot para reaccionar y modificar su velocidad. La Ilustración 6-3 representan el tiempo empleado en cada iteración del programa de monitorización entre que captura una nueva imagen y decide qué velocidad enviar al robot, para cuando se capturan las imágenes con una alta resolución (640x480 píxeles).

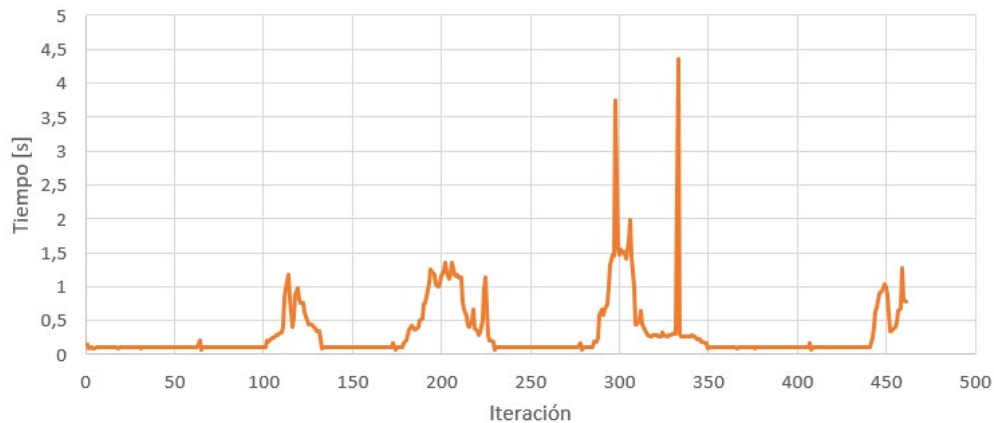


Ilustración 6-3. Gráfico de tiempos de computación para una resolución de 640x480

Se pueden distinguir tres casuísticas: tiempos alrededor de los 0,1 s cuando el área de trabajo se encuentra vacía; tiempos hasta los 1,5 s si el robot se encuentra dentro; y tiempos considerablemente superiores en caso de que coincidan robot y obstáculo. Además, depende de las dimensiones de estos dos, pues el principal causante de este elevado consumo de tiempo es el algoritmo DBSCAN.

El tiempo que este necesita para segmentar los datos de entrada sigue una relación logarítmica con el número de puntos que conformen estos datos. Se recuerda que en cada iteración se realiza por lo menos un paso por DBSCAN y hasta un máximo de 3. En primer lugar y obligatoriamente se pasan los puntos naranjas, en caso de haber suficientes se aplica de nuevo para encontrar al robot y en caso de detectar su presencia se aplica por última vez para buscar un obstáculo.

La Tabla 6-1 presenta los resultados del tiempo empleado por cada uno de estos pasos por el algoritmo para el caso ejemplificado a lo largo del Capítulo 5. Nótese la mejora sufrida al cambiar de ordenador, pues al principio se trabaja con un ordenador personal (procesador Intel Core i7, 7ª generación con 2 núcleos y una frecuencia básica de 2,70 GHz), pero una vez desarrollado todo el código se prueba en los de la universidad (procesador Intel Core i7, 9ª generación con 8 núcleos y 3,00 GHz).

<i>PASO POR DBSCAN</i>	<i>DIMENSIÓN DE LOS DATOS DE ENTRADA</i>	<i>TIEMPO EMPLEADO (i7 7ª gen)</i>	<i>TIEMPO EMPLEADO (i7 9ª gen)</i>
Puntos naranjas (5.1.3.3)	9 500 puntos	0,9 s	0,4 s
Posible robot (5.1.3.5)	17 860 puntos	1,34 s	0,83 s
Posible obstáculo (5.1.4.3)	13 930 puntos	1,2 s	0,38 s

Tabla 6-1. Tiempos de computación DBSCAN para una resolución de 640x480 píxeles

Sin embargo, esta mejora no resulta suficiente pues en temas de seguridad se trata con tiempos de respuesta de milis o incluso nano segundos. Para disminuir estos tiempos se prueban dos alternativas: cambiar a lenguaje Python y reducir los puntos de entrada.

6.2.1 LENGUAJE PYTHON

No se trata de reescribir el programa, sino únicamente la función DBSCAN en este lenguaje. MATLAB permite llamar a funciones fuera de su dominio empleando como sufijo `py`. para el caso de Python.

Si bien, antes de incorporar el uso de la función en MATLAB, se probó con conjuntos de datos artificiales de entre 10mil y 20mil puntos, obteniendo tiempos de computación que también rondaban las unidades de segundos. Es por ello que esta alternativa se descarta rápidamente.

6.2.2 REDUCIR LOS DATOS DE ENTRADA

Sabiendo que al disminuir la cantidad de puntos que pasan por DBSCAN disminuye logarítmicamente el tiempo empleado, se plantea reducir la resolución de la nube de puntos a 320x240 píxeles, como ya se adelantaba en la sección 5.1.1.2. Esto supone, reducir a la mitad por cada dimensión, es decir, nubes de puntos con un cuarto de los puntos previos: 76800 en vez de 307200. Se recuerda además que para poder seguir aprovechándose de la correspondencia con la imagen RGB es necesario también disminuir su dimensión.

Por otro lado, también es necesario adaptar algunos parámetros del código relacionados con las dimensiones: los puntos mínimos necesarios para ser considerado robot u obstáculo y la cantidad de píxeles que se amplía el recuadro que enmarca al robot. Esto se debe a que, evidentemente, si la nube de puntos tiene una dimensión menor, las nubes asociadas a los distintos elementos también se verán reducidas.

Con esta nueva configuración para los datos que recoge el LIDAR, se captura el fondo (ver Ilustración 6-4) y una imagen de una situación análoga a la del ejemplo (ver Ilustración 6-5).

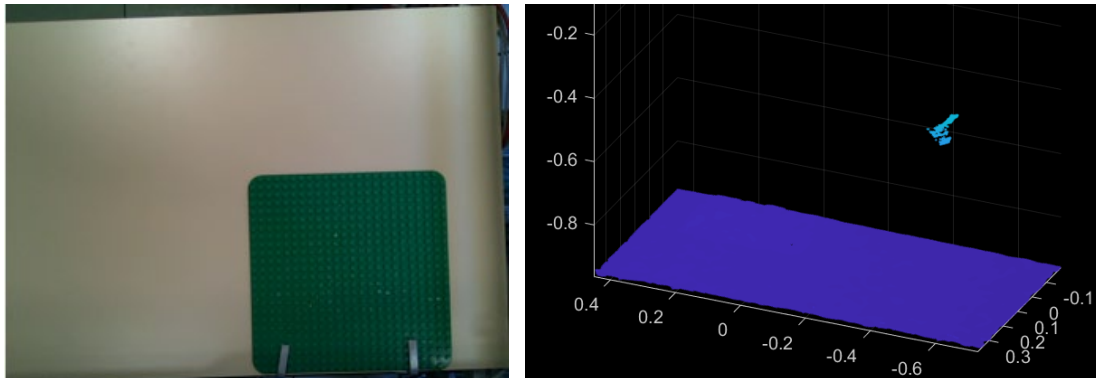


Ilustración 6-4. Fondo con menor resolución

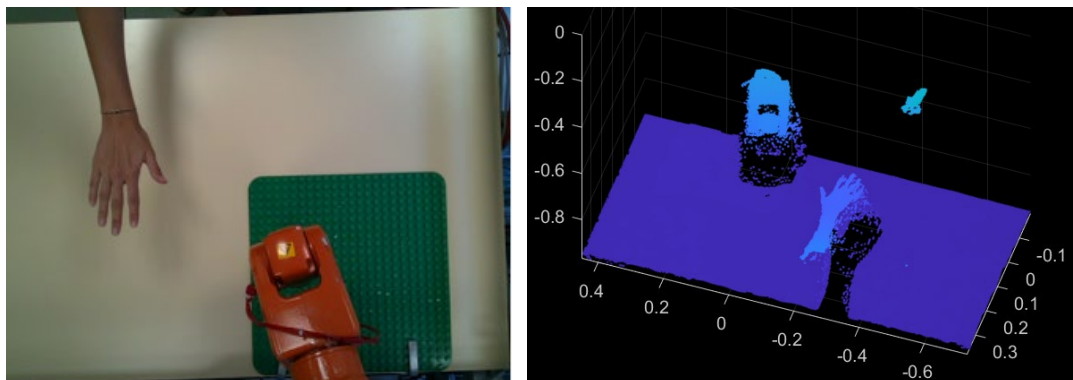


Ilustración 6-5. Imagen del entorno con menor resolución

A simple vista disminuir la resolución no parece suponer una gran pérdida de datos. Para ver si compensa tomar esta medida, la Tabla 6-2 recoge un estudio de los tiempos empleados ahora en cada paso por el algoritmo.

<i>PASO POR DBSCAN</i>	<i>DIMENSIÓN DE LOS DATOS DE ENTRADA</i>	<i>TIEMPO EMPLEADO (i7 7ª gen)</i>	<i>TIEMPO EMPLEADO (i7 9ª gen)</i>
Puntos naranjas	1 640 puntos	0,09 s	0,015 s

Posible robot	3 780 puntos	0,18 s	0,016 s
Posible obstáculo	2 890 puntos	0,18 s	0,028 s

Tabla 6-2. Tiempos de computación DBSCAN para una resolución de 320x240 píxeles

Dado que se produce una mejora evidente, tal y como contrasta la Tabla 6-3, se decide implementar el código modificado para esta resolución y recoger de nuevo los tiempos de computación tras simular la introducción de un obstáculo. El tiempo empleado ahora en cada iteración se muestra en la Ilustración 6-6. Gráfico de tiempos de computación para una resolución de 320x240.

PASO POR DBSCAN	TIEMPO EMPLEADO CON RESOLUCIÓN 640X480	TIEMPO EMPLEADO CON RESOLUCIÓN 320X240
Puntos naranjas	0,4 s	0,015 s
Posible robot	0,83 s	0,016 s
Posible obstáculo	0,38 s	0,028 s

Tabla 6-3. Comparación de tiempos para distintas resoluciones de imagen

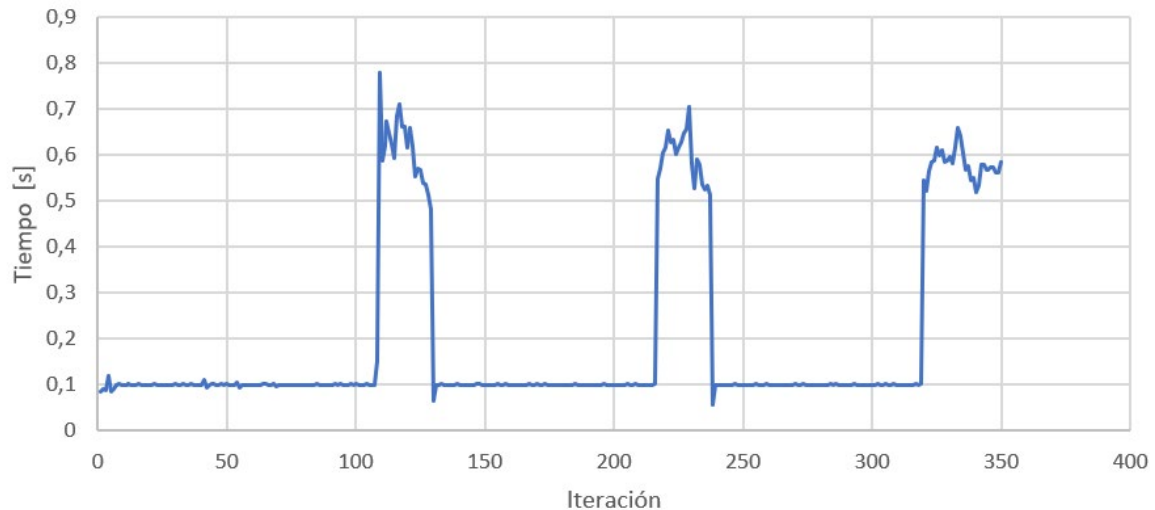


Ilustración 6-6. Gráfico de tiempos de computación para una resolución de 320x240

Estudiando de nuevo las tres casuísticas vemos como la primera de todas- ningún elemento en escena- mantiene unos tiempos de computación alrededor de los 0,1 s. Las dos restantes, robot y robot con obstáculo, apenas se diferencian en esta implementación y se ven reducidos sus tiempos en hasta un 80% con respecto a la anterior. Antes se producía un pico en 4,5 s mientras que ahora apenas alcanza los 0,8 s; sin embargo, aunque sí es cierto que la mejora es admirable, se considera que para ser un sistema destinado a la seguridad aún tiene margen de mejora.

6.3 PÉRDIDA DE INFORMACIÓN

Aunque el diseño segundo del soporte pretende resolver el problema del alcance del sensor, la elevación aportada no supone una solución definitiva. Para ello habría que elevarlo aún más, pero los raíles son fijos.

En las pruebas realizadas, los cuatro puntos por los que se mueve el robot se encuentran muy cercanos a la mesa, o sea, más allá de 25 cm de la cámara; por lo tanto, el robot es percibido al completo en todo momento. Es solo cuando se realizan pruebas aleatorias que se detecta

este problema: si el robot se acerca al sensor deja de ser percibido, como se ejemplifica en la Ilustración 6-5.

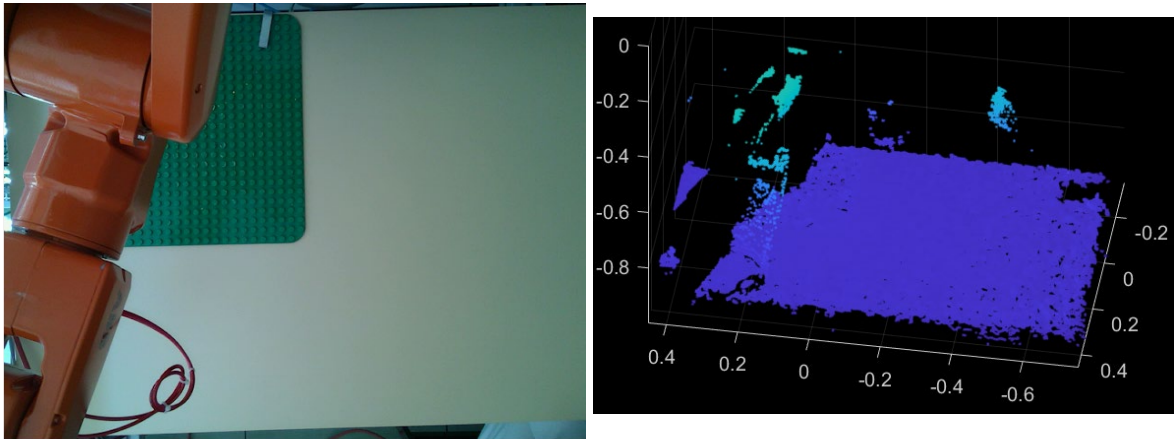


Ilustración 6-7. Robot cercano al sensor

Para resolverlo, habría que rediseñar el soporte para elevarlo bastante más allá del raíl y este sería percibido en todo momento por la cámara. No supondría realmente un problema, ya que al ser un elemento fijo pertenecería al fondo y no sería detectado como obstáculo.

De todas maneras, como esta posición del robot no es habitual en sus operaciones se decide restarle importancia y continuar con el soporte diseñado hasta ahora. Si bien esto es aplicable para el entorno de trabajo actual en el que se realizan las pruebas, en caso de trasladar el sistema a otro lugar habría que reconsiderar la importancia de percibir el robot en determinadas posiciones.

6.4 OBSTÁCULOS ADICIONALES

Otro de los aspectos del funcionamiento del sistema que merecen ser comentados es qué sucede cuando además del factor humano, aparecen en escena otros objetos que podrían tratarse de accesorios que lleve la persona para trabajar, como un estuche o alguna herramienta. Se distinguen entonces dos casos: objetos pequeños y grandes.

6.4.1 OBSTÁCULOS PEQUEÑOS

Los objetos pequeños apoyados en la mesa no son clasificados como obstáculos, como sucede por ejemplo en la Ilustración 6-8 (donde lo morado es fondo y lo rojo robot). Esto se debe a que su nube de puntos tiene una dimensión inferior a la mínima establecida para ser considerado como tal.

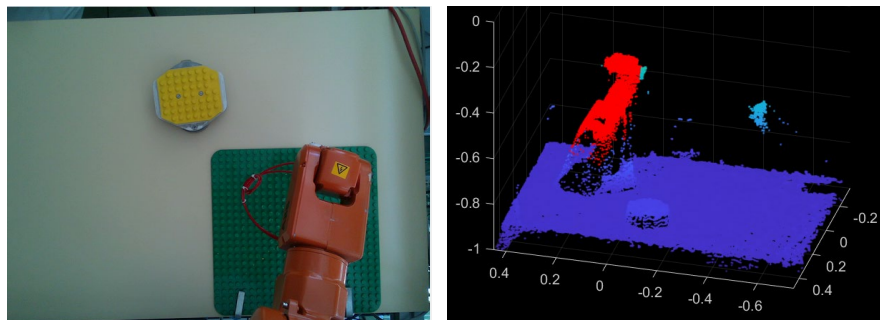


Ilustración 6-8. Clasificación de un escenario con obstáculo pequeño

Podría solucionarse fácilmente rebajando esta cifra, pero dado que estos objetos no son considerados críticos a la hora de garantizar la seguridad se decide que pueden ser ignorados. De todas maneras, en cuanto estos obstáculos adquieran movimiento sí serán detectados, pues para ello alguien o algo los tendrá que coger y mover, aumentando así sus dimensiones.

Objetos de este tipo no suponen ninguna dificultad en caso de coincidir con el factor humano, pues son ignorados como obstáculo y la clasificación del entorno se realiza correctamente, como muestra la Ilustración 6-9:

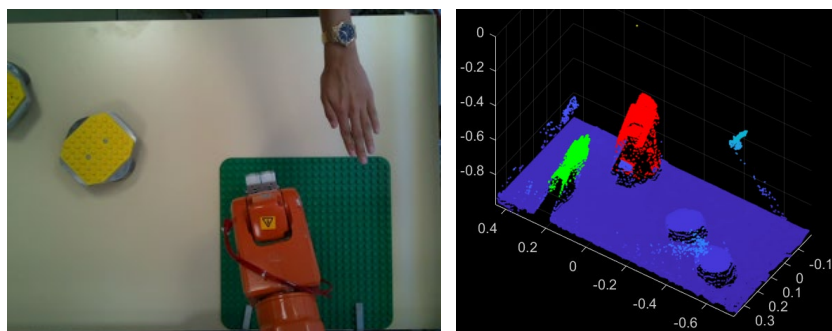


Ilustración 6-9. Segmentación fondo-robot-obstáculo con elementos adicionales pequeños

6.4.2 OBSTÁCULOS GRANDES

En el caso de introducir grandes elementos en el área de trabajo percibida por el LIDAR el sistema pierde su eficacia (ver Ilustración 6-10). Se entiende por elementos grandes aquellos cuyas dimensiones superen las dimensiones de la persona que entra a trabajar en la zona compartida. Dado que el programa asocia el obstáculo con el clúster de mayor dimensión, se deja de monitorizar la distancia entre el robot y la persona.

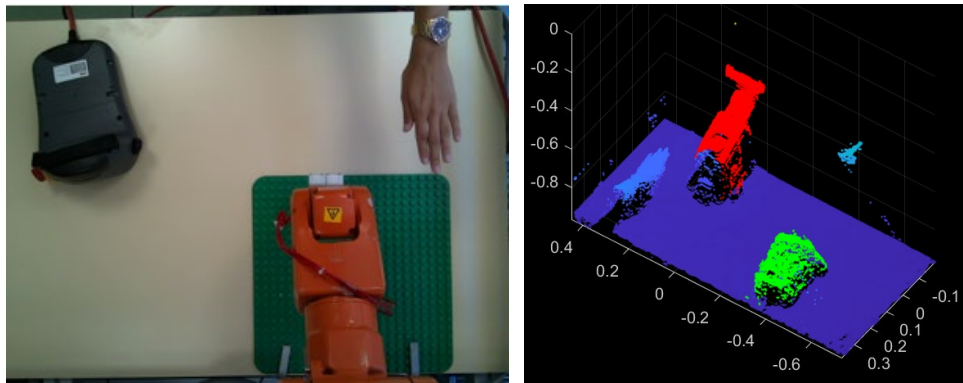


Ilustración 6-10. Segmentación fondo-robot-obstáculo con elementos adicionales grandes

Este caso sí supone un problema a la hora de garantizar la seguridad. Aunque podría evitarse introducir este tipo de objetos en el entorno de trabajo, a priori se identifica una posible solución: no ser considerado obstáculo si no tiene movimiento. De esta manera, en cuanto aparezcan elementos externos en escena se clasifica un posible obstáculo, pero sería en la siguiente iteración cuando se confirme su condición si este cambia de posición.

Capítulo 7. CONCLUSIONES Y TRABAJOS FUTUROS

Este proyecto convierte un entorno de trabajo compartido en colaborativo. El laboratorio de automatización de la Universidad Pontificia de Comillas- ICAI- parte de un funcionamiento en el cual para que el robot trabaje a altas velocidades debe encontrarse dentro de unas vallas cerradas para asegurar que no haya nadie en el interior. En cuanto una persona decide entrar, abriendo una de las vallas, el robot reduce su velocidad a menos de la mitad o llega incluso a detenerse.

Con el fin de permitir al robot continuar con su actividad independientemente de la presencia de personas, se instala un sensor LIDAR con una cámara RGB incorporada (modelo Intel RealSense LiDAR Camera L515). Este aporta al robot la información necesaria sobre el estado del entorno que le rodea, permitiéndole así adaptar su actividad. Todo ello lo logra gracias a un sistema de seguridad compuesto por dos programas. El primero, desarrollado enteramente en MATLAB, es la piedra angular de este proyecto y es el encargado de monitorizar el entorno, examinarlo y decidir la velocidad adecuada en base a este. El segundo se encarga de controlar el robot, tanto su posición como su velocidad. Se establece entre ambos una comunicación unilateral mediante la cual el primer programa le indica al segundo la velocidad a la que debe mover el robot.

En definitiva, se consigue que el robot modifique su velocidad en función de la distancia a la que se encuentre de cualquier elemento externo que no esté normalmente en el entorno de trabajo.

Se trabaja principalmente con la nube de puntos que proporciona el sensor LIDAR, que se trata de una representación 3D del entorno. La imagen RGB se utiliza únicamente para aplicar un filtro de color a los puntos con el fin de seleccionar los naranjas, ya que es el color del robot y ayuda a detectarlo en la escena. La ventaja de la nube de puntos es que se

segmenta con gran facilidad en los distintos conjuntos (robot, mesa, persona, etc.) y permite calcular distancias de una manera muy precisa.

Se consigue entonces con elevado éxito diferenciar en la escena al robot y a cualquier elemento externo que aparezca, calculando su distancia con gran exactitud. Sin embargo, tiene un gran inconveniente y es que la buena definición de la nube de puntos conlleva un elevado número de datos que al ser procesados por MATLAB implican un elevado tiempo de computación. Así, aunque el sistema es capaz de examinar correctamente la situación en cada momento y adaptar su velocidad en consecuencia, la falta de una respuesta rápida hace inapropiado su uso para mantener la seguridad si se desea trabajar a máxima potencia.

Es por ello que a continuación se proponen una serie de posibles desarrollos para mejorar el rendimiento del sistema para garantizar la seguridad.

7.1 DISTINTO LENGUAJE DE PROGRAMACIÓN

En la sección 6.2.1 ya se propone introducir un nuevo lenguaje de programación, pero de una manera breve aplicándolo únicamente a las partes del código que suponen un mayor consumo de tiempo.

Aquí se propone reescribir por completo el programa en lenguajes más rápidos como pueden ser Java o C++, descartando a priori Python dados los resultados obtenidos al probar a reescribir la función DBSCAN en este lenguaje (sección 6.2.1).

7.2 ADAPTACIÓN DE TRAYECTORIA

No tanto por seguridad, sino ya por temas de eficiencia y rendimiento en cuanto a actividades industriales, esta segunda propuesta implica que, en vez de pararse ante una colisión inminente, el robot corrija su trayectoria.

La idea es que el robot se mantenga siempre a una determinada distancia de un obstáculo que aparezca, modificando así su trayectoria, pero no su destino. En caso de no poder garantizar la separación, se pasaría a aplicar el método actualmente desarrollado: reducir la velocidad.

Capítulo 8. BIBLIOGRAFÍA

- [1] D. Feng, L. Rosenbaum y K. Dietmayer, «Towards Safe Autonomous Driving: Capture Uncertainty in the Deep Neural Network For Lidar 3D Vehicle Detection,» *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Maui, HI, USA, 4-7 Nov. 2018, pp. 3266-3273, doi: 10.1109/ITSC.2018.8569814.
- [2] K.-T. Song, Y.-H. Chiu, L.-R. Kang, S.-H. Song, C.-A. Yang, P.-C. Lu y S.-Q. Ou, «Navigation Control Design of a Mobile Robot by Integrating Obstacle Avoidance and LiDAR SLAM,» de *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Miyazaki, Japan, 7-10 Oct. 2018, pp. 1833-1838, doi: 10.1109/SMC.2018.00317.
- [3] «RENISHAW,» [En línea]. Available: <https://www.renishaw.es/es/optical-encoders-and-lidar-scanning--39244>. [Último acceso: 01 06 2021].
- [4] «ABB,» [En línea]. Available: <https://new.abb.com/products/robotics/es/robots-industriales/irb-120>. [Último acceso: 29 06 2021].
- [5] «ABB Library,» [En línea]. Available: <https://search.abb.com/library/Download.aspx?DocumentID=3HAC035960-005&LanguageCode=es&DocumentPartId=&Action=Launch>. [Último acceso: 29 06 2021].
- [6] «ABB Library,» [En línea]. Available: https://search.abb.com/library/Download.aspx?DocumentID=ROBO149EN_D&LanguageCode=en&DocumentPartId=2&Action=Launch. [Último acceso: 29 06 2021].

- [7] «Robótica colaborativa: historia y tendencias recientes,» *EU Automation*, 2019. [En línea] Available: <https://www.euautomation.com/cl/automated/magazine/article/robotica-colaborativa-historia-y-tendencias-recientes1> [Último acceso: 05 07 2021]
- [8] P. N. Ales Vysocky, «Human-Robot Collaboration in industry,» *MM Science Journal*, pp. 903-906, 2016.
- [9] «Robots y dispositivos robóticos. Requisitos de seguridad para robots industriales. Parte 1: Robots. (ISO 10218-1:2011)».
- [10] «Robots y dispositivos robóticos. Requisitos de seguridad para robots industriales. Parte 2: Sistemas robot e integración. (ISO 10218-2:2011)».
- [11] «ISO/TS 15066:2016 Robots and robotic devices — Collaborative robots».
- [12] U. Dombrowski, T. Stefanak y J. Perret, «Interactive Simulation of Human-robot Collaboration Using a Force Feedback Device,» *Procedia Manufacturing*, vol. 11, pp. 124-131, 2017.
- [13] J. A. Marvel, «Performance Metrics of Speed and Separation Monitoring in Shared Workspaces,» *IEEE Transactions on Automation Science and Engineering*, vol. 10, nº 2, pp. 405-414, April 2013, doi: 10.1109/TASE.2013.2237904
- [14] K. Kokkalis, G. Michalos, P. Aivaliotis y S. Makris, «An approach for implementing power and force limiting in sensorless industrial robots,» *Procedia CIRP*, vol. 76, pp. 138-143, 2018.
- [15] «HETPRO,» [En línea]. Available: <https://hetpro-store.com/TUTORIALES/sensor-de-fuerza-o-presion-mf01/>. [Último acceso: 17 04 2021].

- [16] D. Navarro, L. H. Ríos y H. Parra, «Sensores de ultrasonido usados en robótica móvil para la medición de distancias,» *Scientia Et Technica*, vol. X, nº 25, pp. 35-40, 2004.
- [17] «Proyectos con Arduino,» [En línea]. Available: <https://proyectosconarduino.com/sensores/sensores-de-distancia-infrarojos-para-arduino/>. [Último acceso: 10 03 2021].
- [18] R. Alake, «What AlexNet Brought To The World Of Deep Learning,» *towards data science, Medium*, 10 07 2020. Available: <https://towardsdatascience.com/what-alexnet-brought-to-the-world-of-deep-learning-46c7974b46fc> [Último acceso: 03 07 2021].
- [19] G. Karimi, «Introduction to YOLO Algorithm for Object Detection», *Section*, April 15, 2021. [En línea]. Available: <https://www.section.io/engineering-education/introduction-to-yolo-algorithm-for-object-detection/>. [Último acceso: 22 05 2021].
- [20] J. Du, «Understanding of Object Detection Based on CNN Family and YOLO,» *IOP Science. Journal of Physics: Conference Series*, vol. 1004, 2nd International Conference on Machine Vision and Information Technology (CMVIT 2018) 23–25 February 2018, Hong Kong
- [21] A. F. Joseph Redmon, «YOLOv3: An Incremental Improvement,» 08 04 2018. [En línea]. Available: <https://arxiv.org/pdf/1804.02767.pdf>. [Último acceso: 30 01 2021].
- [22] C. R. Qi, H. Su, K. Mo y L. J. Guibas, «PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation,» *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 652-660, 2017.

- [23] S. Shi, X. Wang y H. Li, «PointRCNN: 3D Object Proposal Generation and Detection From Point Cloud,» *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770-779, 2019.
- [24] J. Li, B. M. Chen y G. H. Lee, «SO-Net: Self-Organizing Network for Point Cloud Analysis,» *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9397-9406, 2018.
- [25] T. Jain, «K-Means Clustering,» *Data Driven Investor, Medium*, April 8, 2021. Available: <https://medium.datadriveninvestor.com/k-means-clustering-ac3ff1d3463d> [Último acceso: 06 07 2021]
- [26] X. Yin, Y. Sasaki, W. Wang y K. Shimizu, «3D Object Detection Method Based on YOLO and K-Means for Image and Point Clouds,» 21 04 2020. [En línea]. Available: <https://arxiv.org/pdf/2005.02132.pdf>. [Último acceso: 05 02 2021].
- [27] A. Bonci, P. D. Cen Cheng, M. Indri, G. Nabissi y F. Sibona, «Human-Robot Perception in Industrial Environments: A Survey,» *Sensors*, vol. 21, nº 5, 2021.
- [28] J. A. Corrales Ramón, «Safe human-robot interaction based on multi-sensor fusion and dexterous manipulation planning,» Tesis doctoral, Universidad de Alicante, 2011. <http://hdl.handle.net/10045/22770>
- [29] A. S. Modur, «An Integrated Camera and Radar on-Robot System for Human Robot Collaboration,» Tesis, Rochester Institute of Technology, 2020.
- [30] A. Cherubini, «Multimodal sensor-based control for human-robot,» Robotics [cs.RO]. Université de Montpellier, 2016. tel-01419810.

- [31] P. Long, C. Chevallereau, D. Chablat y A. Girin, «An industrial security system for human-robot coexistence,» *Industrial Robot: An International Journal, Emerald*, vol. 45, nº 2, pp. 220-226, 2018.
- [32] C. Frese, A. Zube y C. Frey, «Workspace monitoring and planning for safe mobile manipulation,» 02 06 2020. [En línea]. Available: <https://arxiv.org/pdf/2006.01546.pdf> [Último acceso: 09 07 2021]
- [33] A. C. Leite, T. B. Almeida-Antonio, P. J. From, F. Lizarralde y L. Hsu, «Control and Obstacle Collision Avoidance Method applied to Human-robot Interaction,» de 2015 *IEEE International Workshop on Advanced Robotics and its Social Impacts (ARSO)*, Lyon, France, 30 June-2 July 2015, pp. 1-8, doi: 10.1109/ARSO.2015.7428222
- [34] «Veo Robotics,» [En línea]. Available: <https://www.veobot.com/freemove>. [Último acceso: 02 07 2021].
- [35] E. Matheson, R. Minto, E. G. G. Zampier, M. Faccio y G. Rosati, «Human–Robot Collaboration in Manufacturing Applications: A Review,» *Robotics*, vol. 8, nº 4: 100, 2019.
- [36] «Intel RealSense,» [En línea]. Available: <https://www.intelrealsense.com/lidar/>. [Último acceso: 22 06 2021].
- [37] «Encycolorpedia,» [En línea]. Available: <https://encycolorpedia.es/ff8000>. [Último acceso: 07 01 2021].
- [38] C. Góngora Luque, «Aplicación de la visión artificial a la seguridad de un robot,» Trabajo de fin de grado, Universidad Pontificia de Comillas-ICAI, 2019.

- [39] «Programa de las Naciones Unidas para el Desarrollo,» [En línea]. Available: <https://www1.undp.org/content/undp/es/home/sustainable-development-goals.html>. [Último acceso: 30 06 2021].

ANEXO I. OBJETIVOS DE DESARROLLO SOSTENIBLE

Los objetivos de desarrollo sostenible [39], conocidos por sus siglas ODS, persiguen acabar con la pobreza, proteger el planeta y garantizar la paz y la prosperidad para el año 2030. Hay un total de 17, según muestra la Ilustración 0-1. Objetivos de Desarrollo Sostenible [39], y se encuentran integrados unos con los otros, es decir, que los resultados de uno pueden influir en los de otro. Con ellos se busca equilibrar la sostenibilidad medio ambiental, económica y social, para lo cual es necesaria la intervención de todo el mundo, desde grandes organizaciones o empresas hasta cada individuo por separado.



Ilustración 0-1. Objetivos de Desarrollo Sostenible [39]

El presente proyecto se identifica principalmente con los objetivos 8 y 9, aunque como se ha mencionado, todos acaban teniendo relación, de manera que influye en menor medida en el resto de los objetivos.

- **Industria, innovación e infraestructura.** La tecnología desarrollada tiene aplicaciones directas en la industria ya que está pensada para mantener la seguridad en el entorno de trabajo de un sistema robótico que esté llevando a cabo alguna tarea. Además, pretende innovar facilitando que máquinas y operarios trabajen al mismo tiempo sin necesidad de que ninguno de los dos reduzca su actividad para evitar accidentes.
- **Trabajo decente y crecimiento económico.** Como bien se comenta en la Introducción de este documento, el proyecto es un grano de arena dentro de la 4ª revolución industrial, y hasta ahora estas siempre han propiciado una mejora en la economía. Asimismo, con una mayor automatización de los procesos industriales, el trabajo llevado a cabo por el factor humano se va dedicando poco a poco a tareas menos mecánicas o manuales, y por lo tanto con un menor número de riesgos asociados. En su lugar surge la necesidad de puestos de trabajo más intelectuales para, por ejemplo, diseñar las máquinas o supervisarlas, lo cual requiere una mayor formación y especialización y por ende son trabajos más valorados, ergo mejor remunerados. Si bien, haría falta una inversión inicial para proporcionar a los actuales obreros la formación necesaria para que ocupen estos nuevos puestos de trabajo y facilitar su adaptación a este nuevo enfoque laboral.