Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

Máster en Ingeniería Industrial

# MÁSTER EN INGENIERÍA INDUSTRIAL

ESCUELA DE INGENIERIAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

## TRABAJO FIN DE MÁSTER

# Visión Artificial y su aplicación en la automatización de tareas de limpieza


Autor: Dña. Ana Misiego López

Tutor: D. Óscar Sánchez Uriarte


Valladolid, Junio 2017

# RESUMEN

El principal objetivo de este proyecto es la realización de un programa capaz de detectar la presencia de un robot aspiradora dentro de una sala, evaluar su actividad y de ser preciso, actuar sobre su trayectoria.

El método de detección está basado en un sistema de visión artificial programado en C++ que con ayuda de librerías OpenCV encuentra la presencia de elementos en movimientos dentro de un vídeo y mediante comparación filtra los resultados obtenidos consiguiendo el elemento buscado, el robot aspiradora.

Localizado el robot aspiradora en el vídeo, el programa comprueba si el mismo se encuentra dentro de una zona no permitida, previamente delimitada por el usuario. En función del resultado obtenido de esta comparación se actúa sobre el robot aspiradora enviándole señales infrarrojas con la ayuda de un microcontrolador Arduino.

La comunicación entre el microcontrolador y el programa de visión artificial es de tipo Puerto Serial.



Palabras clave: Visión Artificial, OpenCV, Arduino, Robot aspiradora, control.



# ABSTRACT

The main goal of this project is the development of a program which will be able of detecting a robot vacuum cleaner presence in a room, evaluate its activity and, if necessary, act over its path.

The detection method is based on a C++programmed computer vision system which which detects the elements' movement inside a video with the help of OpenCV libraries and filters the obtained results through comparison reaching the wanted element, the robot vacuum cleaner.

Once located the robot vacuum cleaner in the video, the program checks if it is inside of a previously defined by the user non-allowed area. Depending on the comparison obtained result there will be an interaction over the robot vacuum cleaner which consists in sending infrared signals with the help of an Arduino microprocessor.

Puerto Serial communication will be used between the microprocessor and the computer vision system.



Key Words: Artificial Vision, OpenCV, Arduino, Robotic Vacuum Cleaner, Control.

Agradecimientos:

# Contenido

# 1. INTRODUCCIÓN

## 1.1. Marco del proyecto

El objetivo del presente proyecto es el desarrollo de un sistema de visión artificial para la detección y localización de un robot aspiradora dentro de una habitación, así como el establecimiento de zonas no permitidas para el robot y control de este último para evitar que adentre en las mismas.

Cabe destacar que se trata de un proyecto genérico con el objetivo de poder controlar cualquiera que sea el modelo y la marca del robot aspiradora, centrándose ante todo en la visión artificial y desarrollando en pequeña medida el control del robot. Es por este motivo por el que se han utilizado en todo momento programas y librerías de programación gratuitas facilitando de este modo el posterior desarrollo de este proyecto.

El uso de software libre en un ámbito destinado a la investigación resulta de vital importancia, ya que de usar un software propietario, las posibilidades de aprendizaje, mejora y ampliación se verían notablemente reducidas, además de los inconvenientes económicos, de ahí, la elección del uso de un microcontrolador Arduino para el control del robot.

Además de ser de software libre y ser muy asequible económicamente hablando, Arduino tiene otras muchas ventajas. Es multi-plataforma, es decir, es capaz de funcionar en gran número de entornos cuando otro tipo de microcontroladores similares sólo son capaces de funcionar en Microsoft.

Tiene la capacidad de ser ampliable, es decir, si fuera necesaria la introducción de más microcontroladores, esto podría hacerse sin problemas, pudiendo comunicar los mismos entre ellos. Además de ser de software libre, es de hardware libre, facilitando el desarrollo de estas placas de forma que actualmente cuentan con una gran gama de productos, pudiendo elegir el que mejor se adapte a las condiciones de trabajo.

Por todos estos motivos, se ha considerado el uso de un microcontrolador Arduino para el control de bajo nivel del presente proyecto.

Actualmente, en el mercado existe una amplia variedad de robots aspiradora, todos ellos prometen una serie de características que tras la compra, los usuarios no ven cumplidas satisfactoriamente, sobre todo dentro del ámbito del sistema de navegación y de limpieza. El presente proyecto nace con el objetivo de solucionar los problemas referentes al sistema de navegación, evitando que el robot entre dentro de zonas problemáticas como pueden ser alfombras de pelo largo, cables, jarrones, etc…

Tras la lectura de múltiples opiniones de todo tipo de modelos, también se ve la necesidad de solucionar otro tipo de problemas, como el evitar que el robot se mueva sin sentido alguno por la superficie y que se centre sólo en la aspiración de una determinada área y deje el resto del suelo sin aspirar. Sin embargo, la creación de los algoritmos necesarios para solucionar estos problemas queda fuera de los objetivos del proyecto, sugiriendo la posibilidad de otro nuevo como ampliación de este.

## 1.2. Objetivos

El objetivo global del presente proyecto es el desarrollo de un programa de visión artificial para el posterior control del robot.

Para poder conseguir este objetivo final, será necesario el desarrollo de una serie de objetivos secundarios de manera independiente, los cuales se citan a continuación:

- Estudio previo de las librerías OpenCV, para posteriormente poder desarrollar correctamente un programa robusto en C++ de visión artificial.
- Desarrollo de un sistema de detección del movimiento dentro de un video.
- Localización de los elementos en movimiento dentro del espacio de análisis.
- Diseño de un sistema de detección del elemento deseado. De haber más dispositivos o animales en movimiento dentro de la sala en la que se está realizando el estudio, es necesario filtrarlos para realizar el tratamiento sobre el elemento adecuado.
- Desarrollo de una pantalla interactiva con el usuario para que puntee las zonas que quiere delimitar el paso del robot aspiradora.
- Diseño de un algoritmo que permita detectar si el robot se encuentra dentro o no de una zona permitida para él.
- Diseño de la comunicación entre el software C++ y el software Arduino.
- Mandar señales infrarrojas al robot aspirador mediante un dispositivo Arduino.

Cumpliendo todos estos objetivos, se garantizará una detección correcta del robot de interés, así como una comunicación con el mismo en caso necesario

## 1.3. Estructura de la memoria

Conocidos todos los objetivos presentes en el proyecto, se pueden diferenciar claramente todas las partes que forman la memoria.

A modo resumen, la memoria está formada por los siguientes capítulos:

- **Capítulo 1. Introducción.** Es el capítulo dónde se realizará una introducción a los robots aspiradora, indicando las razones de la creación de este proyecto, los objetivos necesarios para superar el mismo con éxito y adjuntado un breve resumen de toda la memoria.
- **Capítulo 2. Robot aspiradora.** En este capítulo se hablará sobre los diferentes modelos de robots aspiradora que existen en el mercado actualmente, abordando numerosas marcas así como sus características más destacadas. Se hablará también sobre las características generales que tienen todos estos robots y de los elementos adicionales que se pueden añadir a los mismos para mejorar la experiencia del dispositivo. Por

último se abordarán todos los aspectos negativos que han sido encontrados siendo algunos de ellos el punto de partida de este proyecto.

- **Capítulo 3. Sistema de visión artificial.** A lo largo del capítulo se explicará el proceso de diseño del programa de visión artificial, empezando por la detección del movimiento dentro de la imagen y finalizando con la detección de si el robot se encuentra o no en una zona no permitida.
- **Capítulo 4. Control del robot.** En este capítulo se desarrollará todo el proceso de control y comunicación con el robot aspiradora mediante sensores infrarrojos. Además, se explicará cual ha sido el método de comunicación del microcontrolador con el software de visión artificial.
- **Capítulo 5. Resultados experimentales.** En este capítulo, se mostrarán una serie de imágenes en las que se comprobará el correcto funcionamiento de los diferentes programas.
- **Capítulo 6. Estudio económico.** En este capítulo, se realiza un estudio del coste económico que ha tenido todo el proyecto, teniendo en cuenta tanto los costes directos como los indirectos.
- **Capítulo 7. Conclusiones.** El último capítulo está destinado a las conclusiones obtenidas tras la realización del proyecto, sugiriendo una serie de posibles mejoras.

# 2. ROBOT ASPIRADORA

## 2.1. Introducción

El ajetreado estilo de vida actual impide que las personas no tengan tanto tiempo que dedicar para las labores del hogar como en el pasado. Esta situación obliga a la creación de dispositivos autónomos que sean capaces de hacer estás tareas por nosotros, son los llamados robots domésticos.

Existen multitud de robots domésticos que realizan todo tipo de labores, algunas de ellas son las que se enumeran a continuación:

- Aspirar.
- Cocinar.
- Lavar la vajilla.
- Planchar.
- Fregar.
- Vigilancia.
- Lavar la ropa.
- Atención especializada a personas de edad avanzada.

Con el paso del tiempo se han logrado los avances tecnológicos suficientes para que dichos dispositivos están a disposición de cualquier hogar de clase media y funcionen de forma cada vez más eficiente.

## 2.2. Robots conocidos

El robot doméstico del que se va a tratar en este proyecto es el robot aspiradora, el cual se lanzó por primera vez en 2002 bajo el nombre Roomba, de la empresa iRobot. Se trata del robot aspiradora más conocido hasta la fecha y cuenta con numerosas versiones que han ido mejorando generación tras generación. En este apartado se hablarán de las mismas además de comentar brevemente otro tipo de marcas, algunas más conocidas que otras.

### *iRobot*

El primer robot que lanzó al mercado, 2002, contaba tan sólo con tres botones para indicar las dimensiones de la habitación en la cual se iba a mover el robot. Dicho robot puede verse en la Figura1. Posteriormente, en 2004, véase Figura2, lanzaron la segunda generación de este producto, la Serie 400 también llamada Roomba Discovery. Esta versión supuso una mejora significativa, puesto que el robot ya era capaz de calcular de manera automática el tamaño de la habitación, contaba con un detector de suciedad y cuando su batería estaba a punto de agotarse, volvía sólo a la base de carga.

No será hasta 2007, Serie 500, véase Figura 3, que los robots estén dotados de sensores infrarrojos para detectar obstáculos y sean programables temporalmente.

En la Serie 600 y 700, representados en las Figuras 4 y 5, se incluyeron mejoras como la incorporación de sensores ópticos para la detección de residuos y un mando a distancia para poder controlar ciertos movimientos del robot.

La Serie 800, lanzada en 2013, modificó algunos de los elementos de limpieza de forma que el sistema fuera mucho más eficaz, véase Figura 6.

Por último está la Serie 900, lanzada en 2015 y que se puede observar en la Figura 7. Este nuevo modelo cuenta con numerosas mejoras tecnológicas, puesto que el sistema de vuelta a la base de carga es mucho más eficaz, evitando que el robot se quede sin batería de camino a la base de carga, además de que cuenta con una memoria que permite continuar al robot limpieza en el punto donde terminó. El aspirador puede ser controlado desde el móvil y puede detectar objetos gracias a una cámara que tiene integrada.


Figura 1. Robot Roomba primera generación.

Figura 2. Robot Roomba serie 400.


Figura 3. Robot Roomba serie 500.


Figura 4. Robot Roomba serie 600.


Figura 5. Robot Roomba serie 700.


Figura 6. Robot Roomba serie 800.

Figura 7. Robot Roomba serie 900.

## *Vileda*

Esta empresa alemana ha sacado al mercado diversos robots con un diseño redondeado a precios bastante más económicos como la Vileda Virobi System, mopa robot que navega de forma autónoma retirando polvo, pelos y pelusas y que puede verse en la Figura 8.

Otro modelo muy conocido es el Relax Plus, robot aspiradora, que no mopa, que aspira y recoge el polvo del suelo de modo aleatorio. Cuenta con un mando a distancia para programar la hora de inicio de funcionamiento, con un modo de funcionamiento de vuelta a la base de carga cuando la batería se esté agotando y con un detector de obstáculos. Este último robot se puede observar en la Figura 9.

Figura 8. Vileda Virobi System.

*Figura 9. Vileda Relax Plus.*

## *Neato*

Se trata de una empresa californiana que fabrica robots aspiradoras con multitud de formas, tanto redondas como cuadradas e incluso una combinación de ambas. Tiene dos modelos principales:

- *Botvac Connected*. Dicho robot se puede ver en la Figura 10 y se puede controlar mediante conexión Wi-Fi y mediante una aplicación móvil. Cuenta con diversos modos de limpieza y destaca su filtro de gran rendimiento y su navegación inteligente mediante láser.

Figura 10. Botvac Connected.

- Botvac D. Se trata de una aspiradora muy similar a la anterior, sin embargo no cuenta con conexión WiFi. Esta especialmente indicada para hogares con animales ya que incorpora una escobilla combinada especial para esta tarea.

## *Taurus*

Su robot aspiradora más conocido es el Taurus Striker MINI 948.183, con un funcionamiento muy similar al Roomba pero mucho más asequible económicamente, que cuenta de dos cepillos laterales, un modo tanto de mopa como de aspiración y sensores anticaida. Sin embargo, hay que tener en cuenta que no es capaz de volver a la base cuando la batería está agotándose y no dispone de paredes virtuales. Dicho robot puede verse en la Figura 11.

Figura 11. Taurus Striker MINI 948.183.

## *LG*

Cuenta con una amplia gama de aspiradores HOMBOT, al igual que en el caso anterior, un robot muy similar al Roomba pero con forma cuadrada en vez de circular, contando con dos mopas y dos cepillos laterales, en vez de uno. Un modelo de esta gama puede observarse en la Figura 12, el cual memoriza por donde realiza su recorrido para garantizar limpiar todo el espacio.

Figura 12. LG Hombot Serie 9.

Su último modelo destaca por tener un sistema de video vigilancia desde el cual, el usuario puede tanto ver en tiempo real lo que ocurre en la habitación mientras el robot está limpiando como controlar su movimiento y todo desde el móvil. También cuenta con sensores contra obstáculos y anticaida de tipo infrarrojo, ultrasónico, de giro y desnivel, además de volver a la base de carga cuando su batería se está agotando. Es capaz de mapear las habitaciones para posteriormente realizar la limpieza con patrones ordenados. Dicho mapeo lo realiza gracias a las dos cámaras que lleva integradas, una en la parte superior que escanea la superficie del techo y otra en la parte inferior trasera que mide la distancia de desplazamiento. Otra característica muy importante es que software interno es actualizable mediante USB.

## *Samsung*

Entre los múltiples dispositivos que fabrica esta marca también se encuentran los robots aspiradores. Cuenta con diversos modelos, todos ellos circulares, con diferentes sensores para la localización del polvo. El último modelo, POWERbot VR7000 en la Figura 13, asegura la suficiente capacidad de aspiración para no tener que realizar una limpieza manual posterior así como de alcanzar los lugares de más difícil acceso. Este último modelo también cuenta con sistemas Visionary Mapping Plus y FullView Sensor que mapean y trazan contornos de la sala para garantizar una limpieza minuciosa, detectar áreas sucias y rodear obstáculos.


Figura 13. Navibot Corner Clean.

### *Xiaomi*

La empresa China también ha creada una versión económica del Roomba, llamada Mi Robot Vaccum. Su diseño es ovalado, en vez de circular, posee una gran autonomía, además de tener un radar ultrasónico y una torreta con láseres con el objetivo de mapear la habitación. Produce poco ruido y aunque no cuenta con mando, puede ser controlado mediante una aplicación de móvil. Este robot puede verse en la Figura 14.

Figura 14. Mi Robot Vaccum.

Como se puede observar, en el mercado existen numerosas marcas y modelos de robots aspiradoras y aunque en el proyecto se va a usar el modelo Roomba 866, el mismo está destinado a usarlo con cualquier robot aspiradora.

## 2.3. Características generales de los robots aspiradora

Existen una serie de características que independientemente del modelo y de la marca se repiten. Las mismas se van a proceder a explicar a continuación.

### *Sistema de navegación.*

Junto al sistema de aspiración, el sistema de navegación es una de las características más importantes del robot aspiradora. Existen numerosos modelos que se limitan a ir rebotando de obstáculo en obstáculo, limpiando de manera aleatoria y sin seguir un patrón predefinido. Este tipo de robots no realizan una aspiración ordenada y en muchos casos se dejan grandes áreas de suelo sin limpiar.

Dentro de los robots sin sistemas de navegación, también se pueden encontrar robots que sí siguen unos patrones predefinidos. No garantizan un barrido completo de la superficie pero si una aspiración algo más ordenada que en el caso anterior. Los patrones más comunes son el espiral, en forma de S, poligonal o siguiendo la pared.

Por último, están los robots aspiradora que sí que cuentan con un sistema de navegación. Este tipo de robots no suelen dejar zonas sin limpiar, no repiten la limpieza en espacios que ya están limpios y tienden a usar la batería de manera más eficiente.

Dentro de este apartado es interesante comentar que existen sistemas de navegación que se ayudan de cámaras incorporadas dentro del robot que mapean el techo y el suelo y también existen sistemas de navegación que hacen uso de todo tipo de sensores, ultrasónicos, infrarrojos, etc. También pueden estar dotados de sensores de desnivel para evitar que el robot se caiga por las escaleras, por ejemplo.

### *Sistema de limpieza.*

Dentro de esta característica uno se puede encontrar con diversos casos. Desde robots que no aspiran y sólo mueven la suciedad como si de una escoba se tratara y robots con distintos sistemas de aspiración.

Por norma general, este tipo de máquinas suelen contar con mínimo un cepillo lateral que recoge la suciedad de los lugares más inaccesibles. Además, en la parte central inferior cuentan con unos cepillos contra-rotantes que recogen la suciedad del suelo. Todos estos cepillos se encargan de dirigir la suciedad a la zona de succión, la cual será más o menos efectiva en función de la capacidad de aspiración del robot.

Dentro del sistema de limpieza se pueden incluir su capacidad de programabilidad. Existen robots que no son nada programables, simplemente les encienden y empiezan a aspirar y robots que son del todo programables, es decir, se le puede indicar a qué hora empezar y a qué hora acabar, el modo de limpieza, por cual habitación empezar, etc.

### *Autonomía y capacidad de depósito.*

La autonomía es el tiempo que el robot va a ser capaz de limpiar sin la necesidad de volver a cargarse. Hay que tener en cuenta que existen robots que detectan cuando su batería se está agotando y vuelven a su base de carga automáticamente para cargarse y robots que no tienen está capacidad. Dentro de los robots que sí que vuelven a la base de carga están los que una vez se han cargado vuelven a empezar a limpiar desde el principio y los que vuelven a la zona dónde lo dejaron por última vez para continuar con la limpieza.

Por otro lado, una característica muy importante es la capacidad de depósito de la máquina, puesto que cuanto mayor sea la misma, más suciedad podrá recoger.

Es muy importante una buena combinación de ambas características ya que no sirve de nada una gran autonomía si tiene un depósito de polvo reducido y viceversa.

### *Dimensiones.*

Se trata de una característica muy importante. Si se observa la evolución de estos dispositivos, cada vez tienen una altura más reducida, lo cual implica que podrá meterse por debajo de más muebles, como sofás y camas. También se puede observar como las formas circulares están evolucionando de cara a un mejor acceso a las zonas más difíciles de limpiar, como pueden ser esquinas u otras zonas angulosas del hogar.

### *Ruido.*

Aunque no es su característica principal, hay que tener en cuenta que a la hora de aspirar hay robots que hacen más o menos ruidos pudiendo llegar a ser muy molestos si te encuentras en casa.

### *Filtro HEPA.*

Existen partículas de reducido tamaño que aunque son aspiradas, vuelven a salir al exterior ya que el robot no las puede retener debido a su pequeño tamaño, lo cual puede resultar un problema en casas donde viven alérgicos al polvo.

Para solucionar este problema, algunos modelos incluyen el filtro HEPA (High Efficiency Particulate Air) que son capaces de atrapar partículas de hasta 0.3 micras.

## 2.4. Elementos adicionales

Este robot cuenta con diferentes tipos de elementos para mejorar el modo de limpieza. Entre ellos destacan dos de los cuales se va a hablar a continuación.

### *Pared virtual*

Se trata de un dispositivo que limita la zona por la que el robot puede moverse, de forma que crea una pared invisible que sólo puede verse por los sensores del robot. Así, podemos indicar al robot que no se mueva de una habitación o que no entre en unas zonas determinadas. Un ejemplo del mismo se puede observar en la Figura 15.

Figura 15. Pared Virtual.

Existen diversos tipos de paredes virtuales, están las clásicas, que mediante una luz infrarroja crean una barrera que hace las veces de una puerta. Existen otras paredes virtuales que son inteligentes y que impiden que el robot pase a la habitación siguiente hasta que no haya terminado de limpiar la sala en la que se encuentra. Y por último, están aquellas que crean una barrera circular, muy útil para comederos y bebederos de animales.

### *Mando a distancia*

Se trata de un mando que funciona por infrarrojos desde el cual puedes controlar determinadas funciones del robot. Entre ellas se encuentran:

- Encender y apagar el robot.
- Guiado del mismo con unos botones de dirección.
- Comandos varios como:
    - SPOT. El robot comienza a limpiar haciendo espirales en un área de un 1 metro de diámetro, aproximadamente.
    - CLEAN. El robot calcula de manera autónoma las dimensiones de la habitación que tiene que limpiar y adapta los tiempos de limpieza en función de estos parámetros.
    - DOCK. Se manda al robot que vuelva a la base de carga.

Son estos dos elementos adicionales los que se pretenden sustituir combinando la visión artificial y la electrónica a lo largo de este proyecto.

Mediante una simple cámara colocada en la habitación se registrará en todo momento la ubicación del robot, pudiendo detectar si el mismo se acerca a una zona indeseada y corrigiendo su trayectoria mediante un emisor de infrarrojos.

## 2.5. Desventajas

Viendo todas las características y elementos adicionales, puede parecer que este tipo de dispositivos van a conseguir que el usuario no vuelva a tener que encargase de las tareas de limpieza del suelo nunca más. Pero esto no es así.

Los cepillos de los que se ha hablado anteriormente, prometen llegar a todo tipo de esquinas y sitios inaccesibles, pero en realidad no sólo no llegan a estás zonas si no que tienden a quedarse atascados en las mismas. Además, este tipo de cepillos suelen enredarse en cualquier tipo de cables y flecos, llegando a arrastrar por la casa alfombras de poco peso e incluso, al quedarse enredados los cables, han tirado de ellos y han desenchufado todo tipo de dispositivos.

En muchas ocasiones los cepillos son capaces de remover la suciedad, sin embargo, la capacidad de succión de la aspiradora es reducida, por lo que la suciedad que antes no era visible ahora lo es.

Tampoco son tan silenciosas como prometen y los sistemas de navegación aunque han avanzado mucho, siguen siendo muy mejorables, puesto que muchos siguen repasando zonas que ya están completamente limpias, agotando su batería sin haber empezado a limpiar otras zonas. Muchas marcas prometen que su sistema de navegación memoriza los obstáculos de una sala no siendo así en muchas ocasiones, además de que los sensores contra obstáculos no son tan eficaces como dicen, no siendo del todo adecuado este tipo de robots en casas con decoración excesiva, incluso robots con sensor de desnivel se han caído por las escaleras.

En muchas, también prometen que cuando el aspirador se está quedando sin batería, el mismo vuelve a su base de carga automáticamente para recargarse, en muchos casos esto no es así. Y no solo no vuelven a su base de carga si no que en muchas ocasiones, la autonomía es completamente variable, durando dos horas unos días y en otros no consigue limpiar ni la mitad del tiempo. Muchos son los usuarios que se quejan también de que su robot se queda parado en espacios abiertos sin que la batería se haya llegado a agotar.

Respecto a las paredes virtuales aunque efectivas, acaban resultando un estorbo, sobre todo si hay niños o mascotas por la casa.

Como se puede observar, muchos prometían quitar el trabajo de barrear y aspirar por completo, no siendo así en la mayoría de los casos. Aunque sí que es cierto que ayuda a mantener durante más tiempo la casa limpia, siempre será necesario barrer y aspirar de forma manual cada cierto tiempo.

# 3. SISTEMA DE VISIÓN ARTIFICIAL

## 3.1. Introducción

Como ya se ha dicho, el objetivo de este proyecto es la detección de un robot aspiradora dentro de una habitación siendo necesario conocer la localización del mismo.

Hay que tener en cuenta que dentro de esta habitación puede haber todo tipo de obstáculos y de zonas a las cuales el robot tiene prohibida la entrada y como se ha podido leer en el capítulo anterior, Roomba cuenta con una serie de accesorios para poder poner fin a este problema.

Lo que se quiere conseguir es poder eliminar estos accesorios, por lo que si se conoce la localización del robot, sabremos si el robot se está acercando a estas zonas prohibidas, pudiendo indicarle que realice determinados movimientos para evitar las mismas.

Para poder realizar todo este procesamiento, desde detectar el robot aspiradora, hasta conocer si este está en zonas en las que no debería, será necesario el uso de la programación en la visión artificial, usando un lenguaje de programación en C++ y las librerías OpenCV, las cuales se introducirán a continuación.

## 3.2. Librerías OpenCV

Se tratan de unas librerías desarrolladas a principios de 1999 por Intel y de código abierto. Son unas librerías multiplataforma con versiones en Linux, Mac y Windows y de gran utilidad para el tratamiento de imágenes a tiempo real, que es justo lo que se necesita.

Al tratarse de una librería de código abierto resulta más sencilla la difusión de este tipo de conocimiento dando lugar a una infraestructura común en la que todos los investigadores pueden aportar sus desarrollos, ayudando además, al avance de la visión por ordenador.

Son unas librerías ampliamente utilizadas en todo tipo de sectores y con múltiples aplicaciones, entre las cuales se encuentran:

- Reconocimiento de objetos, tanto en el sector de la seguridad como de la industria.
- Reconocimiento facial.
- Videojuegos.
- Calibración de cámaras.
- Visión robótica.
- Sistemas de visión para vehículos no tripulados.

Como se puede observar, su aplicación es extensa, al igual que sus funciones, entre las cuales, las principales son:

- Captura en tiempo real.

- Importación de archivos de vídeo.
- Tratamiento básico de imágenes.
- Detección de objetos.
- Blob detección.

La librería empezó escribiéndose en C, incluyendo más tarde nuevas interfaces en lenguaje C++, Python y CUDA. Como se puede ver, existen múltiples posibilidades para poder programar el presente proyecto, pero finalmente se eligió programar en C++ por ser el lenguaje que mayor rendimiento tiene, por ser un lenguaje de propósito general que permite la programación tanto en bajo como alto nivel y por tener una programación orientada a objetos.

Antes de comenzar a explicar el programa de visión artificial realizado, comentar que el entorno de programación utilizado para desarrollar todo este apartado ha sido el Visual Studio 15 Community. Ha sido elegido por ser un programa gratuito de Microsoft que ofrece muchas posibilidades, no tiene restricciones al nivel de programación que tiene por objetivo este proyecto y nos permite programar en C++.

## 3.3. Tratamiento

Como ya se ha dicho en la introducción de este capítulo, el objetivo prioritario del presente proyecto es la localización de un robot aspiradora dentro de una habitación.

Hay que tener en cuenta que dentro de la habitación puede haber más objetos a parte del robot y que estos objetos pueden ser tanto estáticos, por ejemplo muebles, como dinámicos, por ejemplo una mascota.

Para poder alcanzar este objetivo será necesaria la colocación de una cámara estática en la habitación que grabe lo que está pasando en la misma, a la vez que se realiza el procesamiento en tiempo real de la grabación. Hay que tener en cuenta que dicho tratamiento de video consta a su vez de un conjunto de tratamientos secundarios que se proceden a explicar a continuación.

Antes de comenzar con la explicación de los diferentes tratamientos secundarios, comentar que la enumeración de los mismos va en orden temporal a su desarrollo, añadiéndolos a los tratamientos anteriormente desarrollados y sirviendo como complemento de los mismos. Los tratamientos secundarios son:

- Detección del movimiento
- Ubicación del movimiento detectado.
- Detección del elemento deseado.
- Localización de zonas prohibidas.
- Detectar si el robot se encuentra en zona permitida.

### *Tratamiento 1. Detección del movimiento.*

En primera instancia se supone que el único elemento móvil que se encuentra en la sala es el robot, por lo que para localizar el mismo, o lo que es lo mismo, para detectar el elemento en movimiento, se recurre a la diferencia de imágenes. Se trata de un método muy sencillo en el cual simplemente se restan dos imágenes para ver lo que cambia una de la otra. Es decir, se resta una imagen patrón previamente almacenada con las imágenes que forman el video que se quiere analizar, de forma que la única variación que hay entre una imagen y la otra es la aparición del elemento en movimiento.

Se trata de un método muy adecuado para este proyecto porque aunque no es muy preciso, es un método bastante simple, muy rápido y que tiene una gran versatilidad ya que permite el tratamiento tanto en entornos de iluminación controlada, como en aquellos con variaciones bruscas.

En este caso, la imagen patrón es una foto de la habitación en la cual no habrá ningún objeto móvil, es decir, se parte de una foto en la cual sólo habrá muebles y plantas. Un ejemplo de ello se puede ver la Figura 16.

Hay que tener en cuenta que una habitación puede tener todas las entradas de luz cerradas y estar iluminada con una bombilla o que la habitación puede estar únicamente iluminada por la luz natural, teniendo que tener en cuenta que esta luz natural puede variar en función de la hora, es decir, si se resta una imagen patrón con gran luminosidad, tomada por ejemplo a las 12:00 y cuyas sombras se encuentran en una dirección, con una imagen en la que se encuentra el robot y que fue tomada a las 21:00, cuyas sombras se encuentran en otra dirección, la diferencia entre ambas no dará sólo el robot, sino las sombras que tampoco son iguales y otros pixeles dispersos a lo largo de la imagen que no tienen la misma luminosidad, color…. o lo que es lo mismo, aparecerán ruidos.

Por este motivo, es muy importante ir captando imágenes patrón cada cierto tiempo, en intervalos de tiempo no constantes y que dependerán de la hora del día. Hay que tener presente que la variación de la iluminación puede ser mayor o menor en función de la hora del día. En el momento del amanecer o del atardecer está variación es mucho más rápida que en cualquier otro momento del día.

Una vez se tiene la imagen patrón, debe ser cargada en el programa a través de la función:

Mat **imread**(const string& **filename**, int **flags**=1 )

Se trata de una función que devuelve una matriz en la cual se almacenan los píxeles que forman la imagen *filename*. Sus parámetros son:

- filename, nombre del fichero que se quiere cargar.
- flags, parámetro que especifica el tipo de color de la imagen cargada, a saber:
  - CV_LOAD_IMAGE_ANYDEPTH, convierte una imagen de 16 o 64 bits en una de 8 bits.
  - CV_LOAD_IMAGE_COLOR, no realiza modificación alguna.
  - CV_LOAD_IMAGE_GRAYSCALE, convierte la imagen a escala de grises.
  - >0, convierte la imagen en una de 3 canales.
  - =0, convierte la imagen a escala de grises.

- <0, no realiza modificación ninguna.

Es importante que esta carga se realice correctamente, por lo que se comprobará previamente a cualquier tratamiento posterior mediante la siguiente estructura:

```
if (!imagen.data)
{
        cout << "Error al cargar la image.. " << endl;
        exit(1);
}
```

Una vez cargada, está debe ser pasada a escala de grises a través de la siguiente función:

*void* ***cvtColor****(InputArray* ***src****, OutputArray* ***dst****, int* ***code****, int* ***dstCn****=0 )*

Esta función se encarga de pasar una imagen de un espacio de color a otro y tiene los siguientes parámetros:

- Src, matriz donde se almacenan los píxeles de la imagen que se quiere modificar.
- Dst, matriz donde se almacenan los píxeles de la imagen modificada.
- Code, espacio de color al que se quiere convertir la imagen.
- dstCn, número de canales que formarán la imagen destino. Este parámetro puede ser obviado, considerando que la imagen destino tendrá el mismo número de canales que la imagen origen.

A continuación, se muestra un ejemplo gráfico de este proceso con imágenes de este proyecto para mayor claridad.

Dicho proceso se puede ver representado en las Figuras 16 y 17, dónde se pueden observar la imagen patrón captada tanto en formato a color como en escala de grises, respectivamente.

*Figura 16. Imagen patrón original.*

*Figura 17. Imagen patrón en escala de grises.*

Una vez tratada la imagen patrón se deberá restar con las diferentes imágenes que forman el video, por lo que se realizará un tratamiento dentro de un bucle de forma que de manera constante se realice el procesamiento hasta el usuario pulse a la tecla *escape*.

Este procesamiento consiste en primer lugar, tomar una imagen del video con el siguiente código:

VideoCapture::read(Mat& **image**)

Esta línea se encarga de guardar una imagen de un video y está formado por un único parámetro y es la matriz *image* que es dónde se almacenan los píxeles de esta captura del video.

Para poder restar está imagen obtenida del video con la imagen patrón, es necesario que se encuentren en el mismo formato, es decir, ambas imágenes deben estar en escala de grises, por lo que de nuevo se vuelve a realizar el tratamiento de conversión con la misma función.

*void* ***cvtColor****(InputArray* ***src****, OutputArray* ***dst****, int* ***code****, int* ***dstCn****=0 )*

De nuevo, se vuelve a mostrar un ejemplo gráfico con imágenes de este proyecto para mayor claridad.

En la Figura 18 se encuentra representada una de las múltiples capturas del video en color y en la Figura 19 se puede observar esta misma imagen pero en escala de grises.

*Figura 18. Captura original.*

*Figura 19. Captura en escala de grises.*

Una vez están las dos imágenes que se pretenden restar en el mismo formato, se procede a restarlas a través de la siguiente función:

void **absdiff**(InputArray **src1**, InputArray **src2**, OutputArray **dst**)

Se trata de una función que resta de manera absoluta dos vectores o matrices, en este caso, entre ellos. Como se puede observar, consta de 3 parámetros diferentes:

- Src1, uno de los vectores o matrices que se van a restar.
- Src2, uno de los vectores o matrices que se van a restar.
- Dst, vector o matriz dónde se va a almacenar la resta de los dos datos de entrada. Hay que tener en cuenta que al ser los dos datos de entrada del mismo formato, el dato de salida tendrá el mismo formato que los dos anteriores.

A fin de ver gráficamente y de forma más clara el valor de esta resta, se adjunta un ejemplo de la misma en la Figura 20.

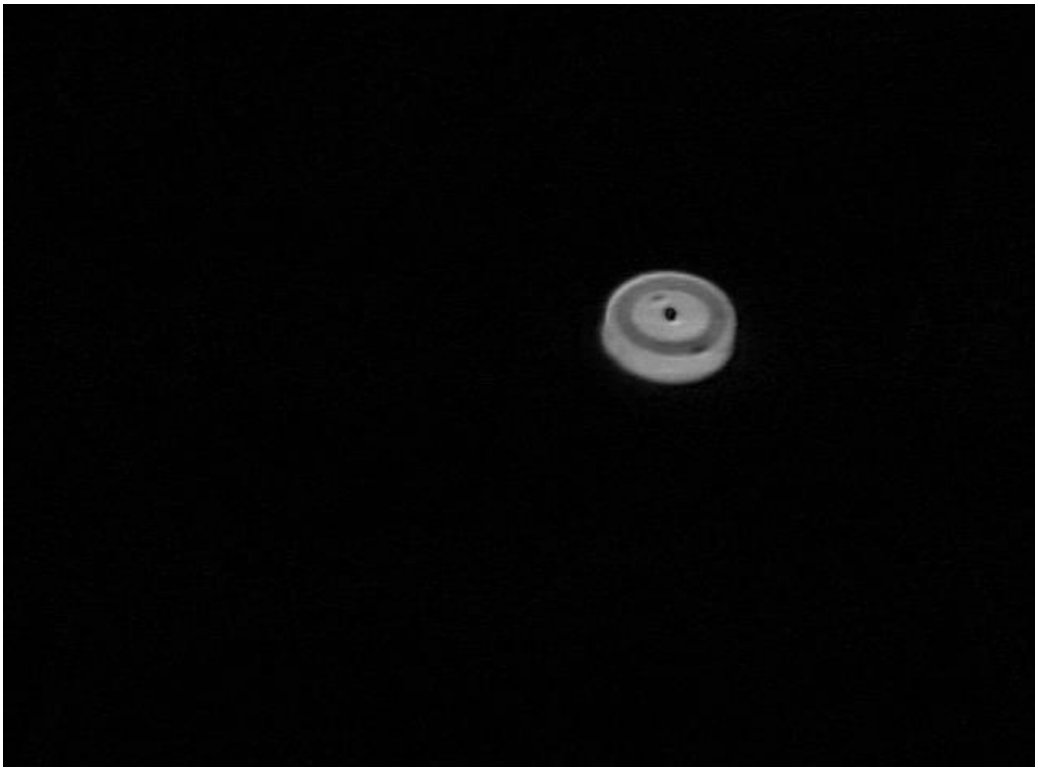

*Figura 20. Imagen resta resultante.*

Como se puede observar, esta resta muestra al robot aspiradora con bastantes detalles, con el fin de eliminar los mismos se ha decidido usar la función blur.

Esta función es un filtro lineal que tiene por objeto el suavizado de imágenes mediante el filtro kernel normalizado. Dicho filtro kernel se puede expresar matemáticamente mediante la siguiente matriz:

$$Kernel = \frac{1}{ksize.ancho \cdot ksize.alto} \cdot \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}$$

*void* ***blur****(InputArray* ***src****, OutputArray* ***dst****, Size* ***ksize****, Point* ***anchor****=Point(-1,-1), int* ***borderType****=BORDER_DEFAULT )*

Esta función tiene los siguientes parámetros:

- src: Matriz o vector dónde se almacenan los píxeles de la imagen de entrada.
- dst*:* Matriz o vector dónde se almacenan los píxeles de la imagen resultante.
- Ksize, es el tamaño de la máscara usada para realizar el desenfoque.
- Anchor, es el punto que se va a indicar como centro del filtro. Por defecto toma el valor Point(-1,-1), es decir, toma por defecto el centro de la matriz.
- borderType, forma en la que se van a tratar los pixeles que se encuentran fuera de la imagen que se va a difuminar.

El resultado gráfico de toda esta explicación se encuentra representado en la Figura 21, donde se puede observar al robot difuminado.

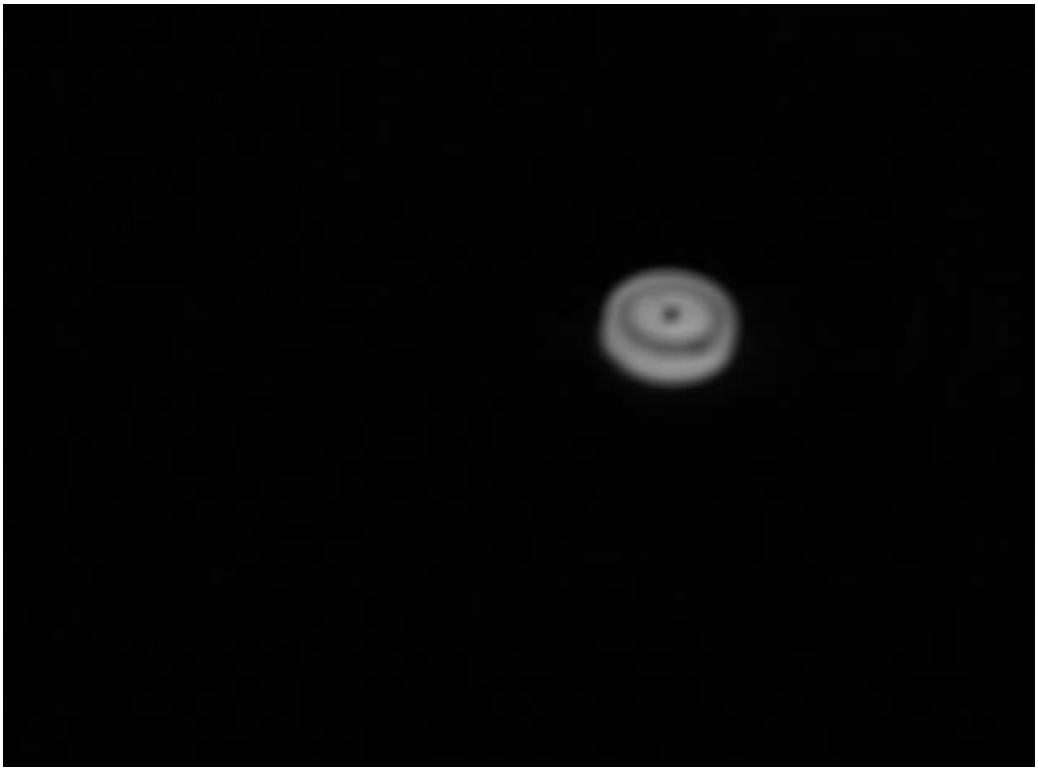

*Figura 21. Diferencia difuminada.*

Difuminada la imagen del robot, se podrá realizar el último paso de este tratamiento que consiste en la binarización. Este último paso resulta de gran importancia debido a que:

- Trabajar con imágenes binarias reduce al mínimo los datos necesarios para representar la imagen, lo que implica un menor coste computacional.
- Las propiedades tanto geométricas como topológicas de los objetos de una imagen binaria se pueden obtener con mayor rapidez y de manera más sencilla que en una imagen con mayor número de niveles de gris.

Para realizar esta binarización se procede a usar la función:

double **threshold**(InputArray **src**, OutputArray **dst**, double **thresh**, double **maxval**, int **type**)

Esta función se encarga de aplicar un umbral para cada elemento de la matriz y está formado por los 5 siguientes parámetros:

- Src, es el vector o matriz donde se almacenan los pixeles de la imagen que se quiere umbralizar.
- Dst, es el vector o matriz donde se almacenan los pixeles de la imagen umbralizada.
- Thresh, es el valor que se quiere umbralizar.
- Maxval, valor máximo que se va a aplicar en la umbralización.
- Type, es el tipo de umbralización que se va a realizar. Existen 6 tipos diferentes de umbralización de las cuales se va a escoger la umbralización binaria, *THRESH_BINARY*, que consiste en que si el nivel de gris de un pixel de la imagen de entrada supera el valor *thresh*, ese pixel pasa a tomar el valor *maxval* en la imagen de

salida, en caso contrario, toma el valor 0, es decir, se representa de color negro. En el caso de este proyecto, como se quiere que la imagen sea binaria sea en blanco y negro, el valor *maxval* será igual a 255. De manera un poco más esquemática, el párrafo se podría resumir en:

$$dst(x,y) = \begin{cases} maxval & si\ src(x,y) > thresh \\ 0 & si\ src(x,y) \leq thresh \end{cases}$$

El resultado de la umbralización binaria de la imagen difuminada se encuentra representada en la Figura 22, donde se puede ver el área del robot en blanco y el resto en negro.

Tras toda esta explicación, se puede entender con mayor claridad la necesidad de una difuminación previa, puesto que sin ella, no aparecería toda el área del robot en blanco. Si se observa detenidamente la Figura 19, la mayor parte del robot se encuentra en niveles de gris intermedios, pero por ejemplo, en el centro del robot, hay un conjunto de pixeles que son completamente negros. Si se realizará la umbralización directamente sobre esa imagen, saldría una imagen similar a la de la Figura 22 pero con zonas internas del robot en negro, lo cual podrá suponer problemas en tratamiento posteriores, como por ejemplo, a la hora de calcular el área o el centro del robot, la cual se realizará sobre la *"mancha"* blanca. Al difuminar previamente, ayudamos a que estos pixeles totalmente negros que se encuentran en el interior del robot pasen a niveles de gris intermedios, pudiendo binarizar esos pixeles a blanco.

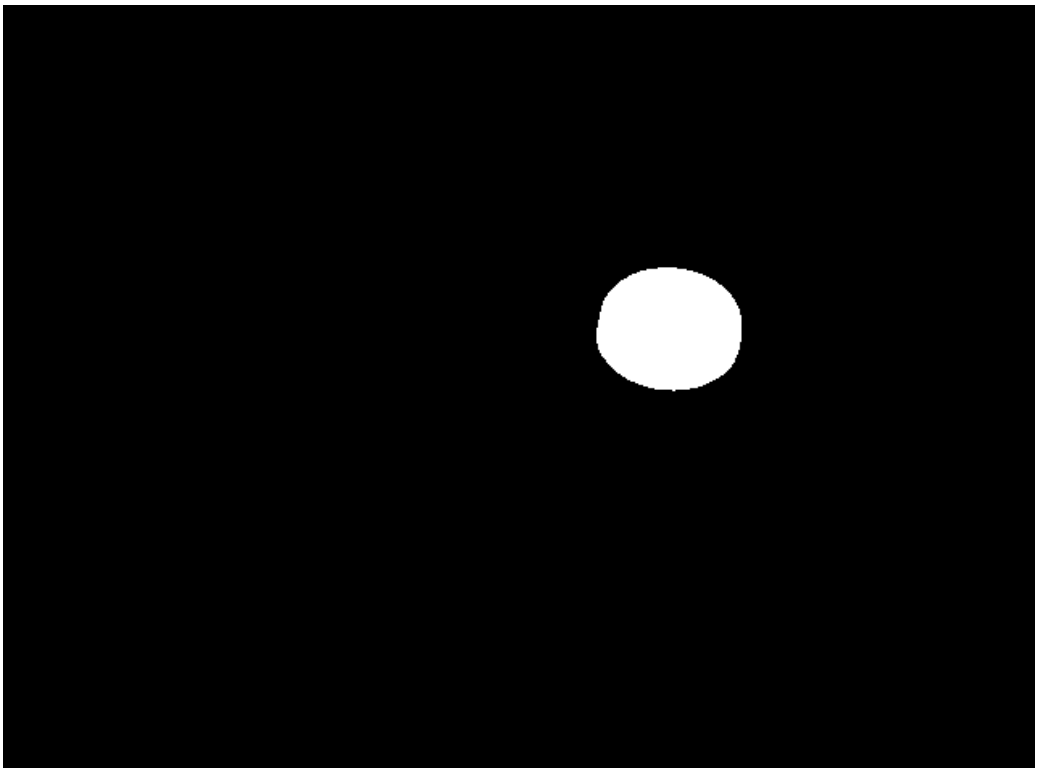

*Figura 22. Diferencia umbralizada.*

Para finalizar con el tratamiento se adjunta un diagrama de bloques, Diagrama 1, con el fin de tener una visión rápida del mismo.

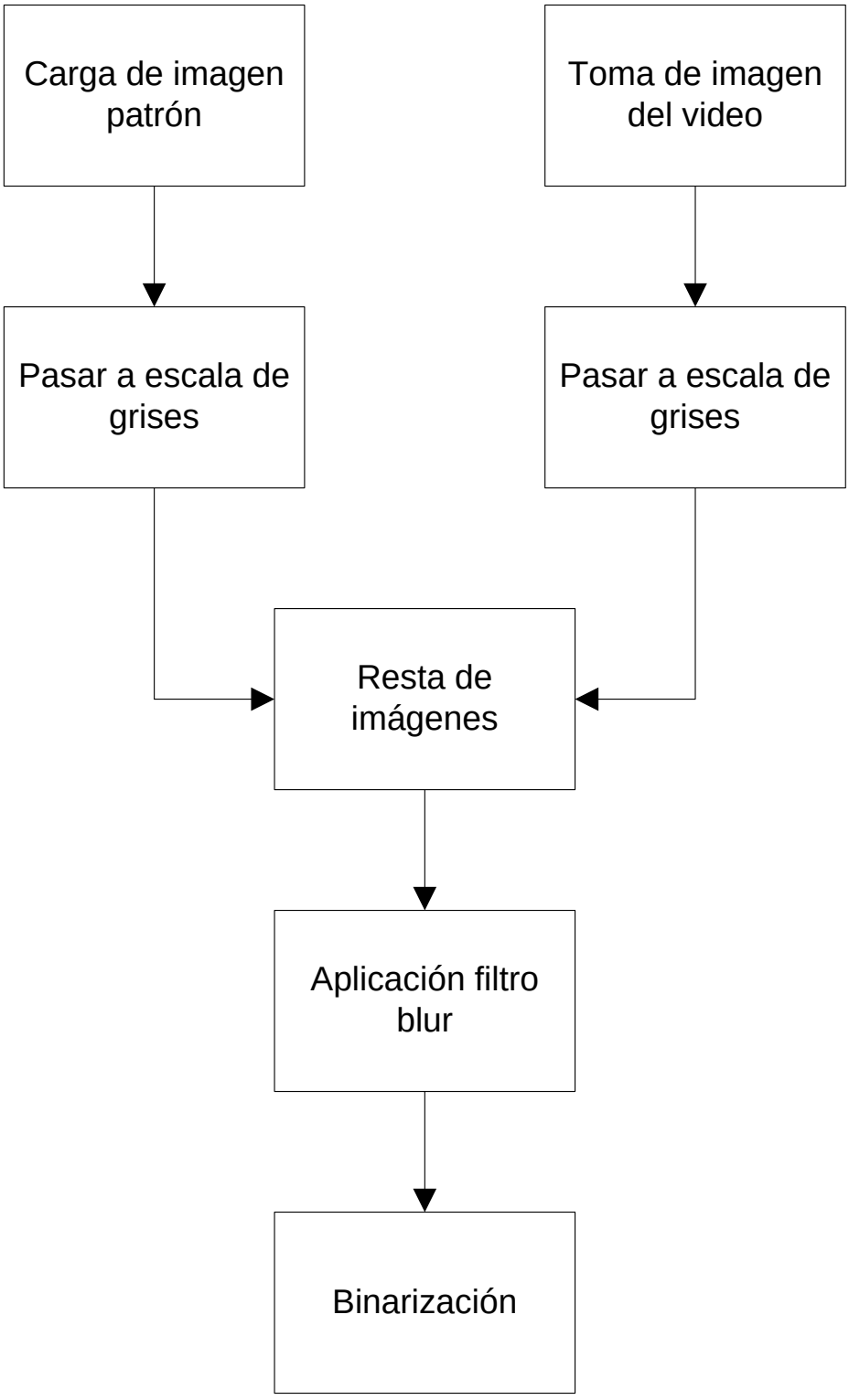


Diagrama 1. Tratamiento 1.

### *Tratamiento 2. Ubicación del movimiento detectado.*

Una vez se tiene el objeto que se encuentra en movimiento claramente definido y binarizado se puede continuar con el siguiente proceso de tratamiento de imagen, el cual va a consistir en determinar la ubicación del robot dentro del escenario para más adelante poder conocer su posición respecto a una zona prohibida.

Todo este proceso se puede resumir en los siguientes pasos, los cuales se proceden a explicar en detalle a continuación:

- Detectar el contorno del objeto en movimiento.
- Dibujar el contorno.
- Calcular el centro de coordenadas del área que bordea el contorno.
- Representar el centro de coordenadas en la imagen.

Como se ha podido leer en el resumen, en primer lugar es necesario contornear el objeto en movimiento, o lo que es lo mismo, hay que contornear los pixeles en blanco. Para conseguirlo, se deberá usar la siguiente función:

void **findContours**(InputOutputArray **image**, OutputArrayOfArrays **contours**, OutputArray **hierarchy**, int **mode**, int **method**, Point **offset**=Point())

Esta función se encarga de encontrar contornos en una imagen binaria y consta de los siguientes parámetros:

- Image, es la matriz o vector donde se almacenan los pixeles de la imagen de entrada, es decir, de la imagen en la cual se quiere encontrar el contorno.
- Contours, es un vector de puntos donde se almacena cada contorno que se detecta en la imagen de entrada.
- Hierarchy, es un vector de salida que contiene información sobre la tipología de la imagen. El vector estará formado por tantos elementos como contornos se encuentren.
- Mode, parámetro mediante el cual se indica a la función el método para encontrar el contorno. Hay diversos tipos:
    - CV_RETR_EXTERNAL: Obtiene únicamente el contorno más exterior de un objeto. Se trata del modo utilizado en el proyecto.
    - CV_RETR_LIST: Obtiene todos los contornos de un objeto, tanto exteriores como interiores, sin jerarquías.
    - CV_RETR_CCOMP: Obtiene todos los contornos de un objeto, tanto exteriores como interiores, jerarquizándolos en dos niveles.
    - CV_RETR_TREE: Obtiene todos los contornos del objeto y los jerarquiza en niveles anidados.
- Method, a través de este parámetro, el usuario indica a la función el método de aproximación del contorno. Al igual que con el parámetro *mode*, existen varios tipos:
    - CV_CHAIN_APPROX_NONE: almacena todos los puntos del contorno.

    - CV_CHAIN_APPROX_SIMPLE: comprime segmentos horizontales, verticales y diagonales y deja sólo sus puntos finales.
    - CV_CHAIN_APPROX_TC89_L1/ CV_CHAIN_APPROX_TC89_KCOS: se aplica el algoritmo de aproximación Teh-Chin.
- Offset, es un parámetro opcional con el cual se puede indicar si se quieren desplazar todos los puntos del contorno.

Encontrado el contorno, se procede a su representación en la imagen mediante la función:

void **drawContours**(InputOutputArray **image**, InputArrayOfArrays **contours**, int **contourIdx**, const Scalar& **color**, int **thickness**=1, int **lineType**=8, InputArray **hierarchy**=noArray(), int **maxLevel**=INT_MAX, Point **offset**=Point() )

Esta función se encarga de dibujar los contornos y está formada por las siguientes variables:

- Image, es la matriz o vector donde se almacenan los pixeles de la imagen de salida en la cual se va a dibujar el contorno.
- Contours, es un vector donde se encuentran todos los puntos de los diferentes contornos detectados en la función anterior.
- ContourIdz, en el anterior parámetro se introduce un vector donde se encuentran todos los contornos encontrados, por lo que es necesario indicar que contornos son los que se quieren dibujar. Mediante este parámetro indicamos cuales son los contornos que interesan.
- Color, indica el color en el cual se va a dibujar el contorno.
- Thickness, anchura con la cual se va a dibujar el contorno.
- lineType, forma de las líneas que unen los puntos que forman el contorno.
- Hierarchy, es un vector obtenido en la función anterior y que opcionalmente se puede introducir en está si se desean dibujar sólo alguno de los contornos.
- maxLevel, máximo nivel hasta el cual se pueden dibujar los contornos. Se trata de un parámetro opcional y que sólo es necesario en caso de indicar el parámetro hierarchy. Tiene tres posibles valores:
    - 0, se dibuja tan sólo el contorno especificado.
    - 1, se dibuja el contorno especificado y todos los contornos anidados.
    - 2, se dibuja el contorno especificado, todos sus contornos anidados, los contornos anidados de estos últimos, y así sucesivamente.
- Offset, se trata de un parámetro opcional para modificar el contorno.

El resultado de estas dos funciones se puede ver representado en un caso real del proyecto en la Figura 23. En esta imagen se puede observar como el programa es capaz de bordear perfectamente en color azul el robot que se encuentra en movimiento.

Figura 23. Contorno del objeto en movimiento.

Una vez obtenido y dibujado el contorno del objeto, se procede a calcular el centro de coordenadas del mismo, para posteriormente indicarlos en la imagen.

Para ello, en primer lugar será necesario recuadrar el área que encierra el contorno mediante la función:

Rect **boundingRect**(InputArray **points**)

Se trata de una función muy sencilla la cual calcula el rectángulo que delimita un conjunto de puntos. Consta de tan sólo un parámetro que es dicho conjunto de puntos, en el caso del presente proyecto, será el vector donde se almacenan los puntos que forman el contorno.

Una vez se tiene el rectángulo que delimita el contorno, resulta muy sencillo calcular el centro de coordenadas del mismo. Al conocer la posición de un vértice (x,y), la altura y la anchura del rectángulo, el cálculo del centro es tan sencillo como se indica en las ecuación que se detallan a continuación:

$$coordenadaX = x + ancho/2$$

$$coordenadaY = y + altura/2$$

Para mayor claridad se adjunta una imagen explicativa, véase Figura 24.

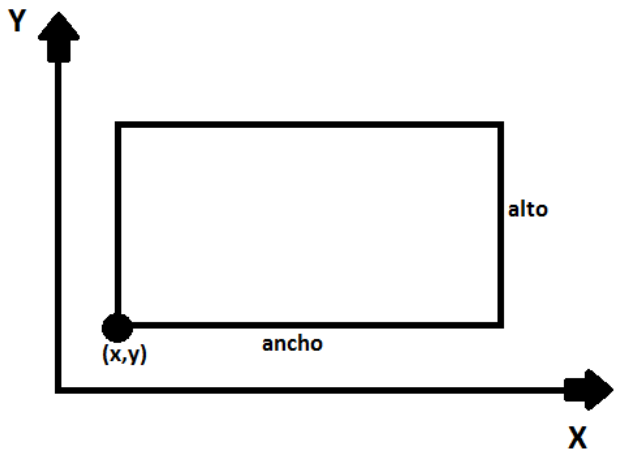


Figura 24. Representación centro de coordenadas.

Con todos estos datos, se continúa el programa dibujando una "X" en el centro de coordenadas e indicando numéricamente la posición del mismo.

Para dibujar la X se ha dibujado en primer lugar un punto mediante la función:

void **circle**(Mat& **img**, Point **center**, int **radius**, const Scalar& **color**, int **thickness**=1, int **lineType**=8, int **shift**=0)

Está función dibuja una circunferencia y consta de los siguientes parámetros:

- img, matriz donde se almacenan los pixeles de la imagen en la cual se va a dibujar la circunferencia.
- center, parámetro en el cual se indica el centro de la circunferencia.
- radius, parámetro en el cual se indica el radio de la circunferencia.
- color, parámetro en el cual se indica el color con el que se va a dibujar el círculo.
- thickness, indicador del grosor de la circunferencia.
- lineType, forma de la línea con la que se va a representar la circunferencia.
- shift, número de fracciones en las que se divide tanto el valor del radio como del centro de coordenadas.

Tras dibujar el punto y con el objetivo de que visualmente el centro se represente con una "X", se han dibujado 4 líneas, dos horizontales y dos verticales, que salen del centro de coordenadas. Dichas líneas se ha dibujado con ayuda de la función:

void **line**(Mat& **img**, Point **pt1**, Point **pt2**, const Scalar& **color**, int **thickness**=1, int **lineType**=8, int **shift**=0)

Se trata de una función que dibuja un segmento que une dos puntos y sus parámetros son los siguientes:

- Img, matriz donde se almacenan los pixeles de la imagen en la cual se va a dibujar la línea.
- Pt1, primer punto que va a servir para definir el segmento.

- Pt2, segundo punto con el que se va a definir el segmento.
- color, parámetro en el cual se indica el color con el que se va a dibujar el segmento.
- thickness, indicador del grosor de la línea.
- lineType, forma de la línea con la que se va a representar el segmento.
- shift, número de fracciones en las que se divide las coordenadas del punto.

Dibujada la "X" se procede a indicar numéricamente la posición de este centro de coordenadas mediante la función:

void **putText**(Mat& **img**, const string& **text**, Point **org**, int **fontFace**, double **fontScale**, Scalar **color**, int **thickness**=1, int **lineType**=8, bool **bottomLeftOrigin**=false )

Se trata de una función que dibuja en una imagen una cadena de texto y está formado por los siguientes parámetros:

- Img, matriz donde se almacenan los pixeles de la imagen en la cual se va a dibujar el texto.
- text, cadena de caracteres que se van a incluir en la imagen.
- Org, esquina inferior izquierda de la cadena de texto en la imagen.
- fontFace, parámetro en el que se indica la estructura de la fuente.
- fontScale, parámetro en el que se indica el tipo de fuente
- color, parámetro en el cual se indica el color con el que se va a dibujar el texto.
- thickness, indicador del grosor del texto.
- lineType, forma de la línea con la que se va a representar el texto.
- bottomLeftOrigin, indica el origen de los datos de la imagen en la esquina inferior izqueirda.

Como se puede observar, en esta última función el texto se indica en formato string. Hay que tener en cuenta que el texto que se va a introducir son números que no están en este formato, por lo que es necesario una conversión. Para realizar dicha conversión se ha desarrollado una pequeña función que se encarga de convertir las variables número (int) en variables texto (string) y que se encontrará anidada en la función *putText*. La función de conversión es la siguiente:

```
string conversion(int number)
{
        std::stringstream ss;
        ss << number;
        return ss.str();
}
```

Como se lleva realizando en toda la memoria, se adjunta a continuación una imagen del proyecto en la que se puede ver el resultado de todo el tratamiento. En la Figura 25 se ve

cómo se rodea el objeto en movimiento, se dibuja el centro de dicho objeto y se indican las coordenadas del mismo.

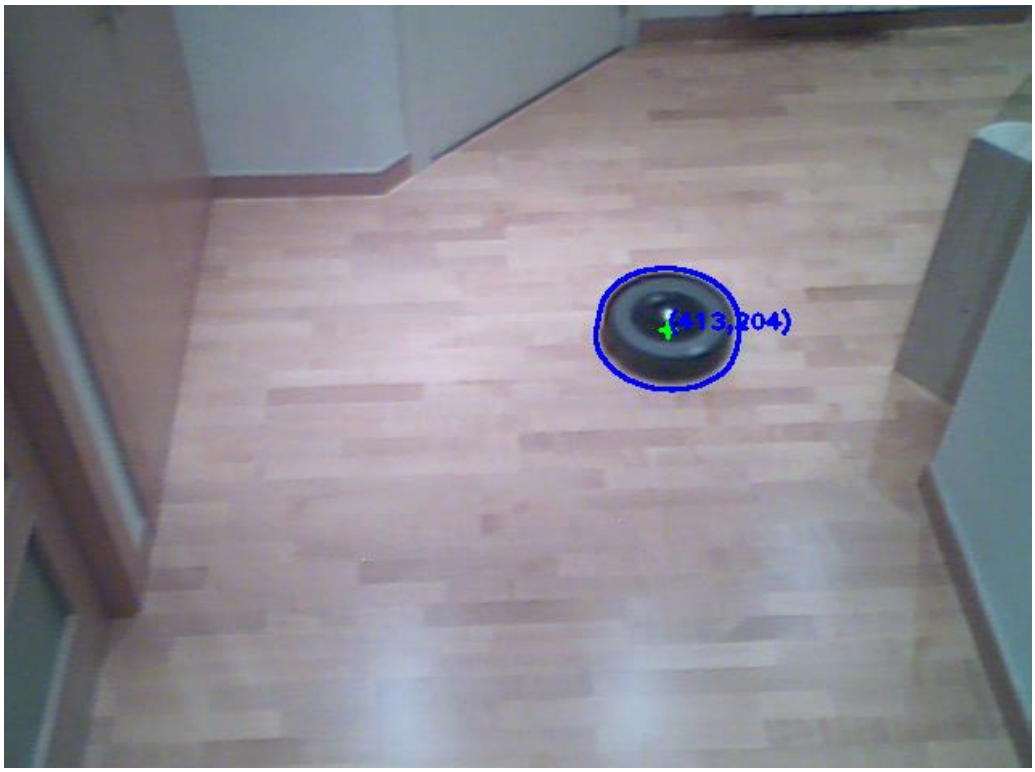


Figura 25. Coordenadas del objeto en movimiento.

De nuevo, con el fin de tener una visualización rápida de este tratamiento, se adjunta un diagrama de bloques del mismo, véase Diagrama 2.

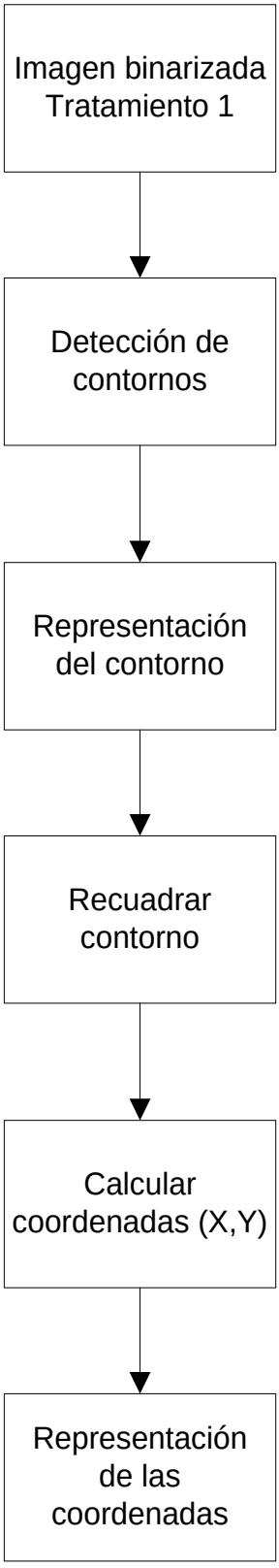


Diagrama 2. Tratamiento 2.

### *Tratamiento 3. Detección del elemento deseado.*

Con los dos primeros tratamientos, el programa ya es capaz de detectar cualquier objeto en movimiento de la imagen, contornearlo y representar su centro de coordenadas.

Pero, ¿qué ocurre si hay más de un objeto en movimiento en la imagen?

En breves palabras, para evitar que el programa detecte un objeto que no sea el buscado, se ha creado una matriz de tamaño en la que se indica un intervalo del área del contorno del robot en unas coordenadas determinadas. De esta forma, si hay otro objeto en la escena, el programa le ignorará.

Obviamente, en función del lugar dónde se coloque la cámara, las áreas en cada intervalo de coordenadas serán diferentes, por lo que resulta necesaria la realización de un programa que acompañe al programa principal de visión artificial que proporcione todos estos datos fácilmente.

Este programa para obtener la matriz de tamaño funciona de la siguiente manera:

- En primer lugar, el programa indicará por pantalla las instrucciones a seguir.
- A continuación, se mostrará un video en tiempo real de lo que se está captando por la cámara. En esta imagen, se podrá ver una cuadrícula, tal y como se observa en la Figura 26.
- Con ayuda del mando, guiaremos al robot aspiradora por todos y cada uno de los cuadros de forma que al colocar el robot en uno de ellos, el usuario pulsará la barra espaciadora. Al pulsar esta tecla, el programa almacenará en un documento las coordenadas y el área de este cuadro. De esta forma, podremos rellenar fácilmente la tabla de tamaño que se encuentra adjunta en la Tabla 1.
- Cuando se haya colocado y guardado toda la información del robot aspiradora en los diferentes puntos de la cuadrícula, el usuario pulsará la tecla *escape* para cerrar el programa.

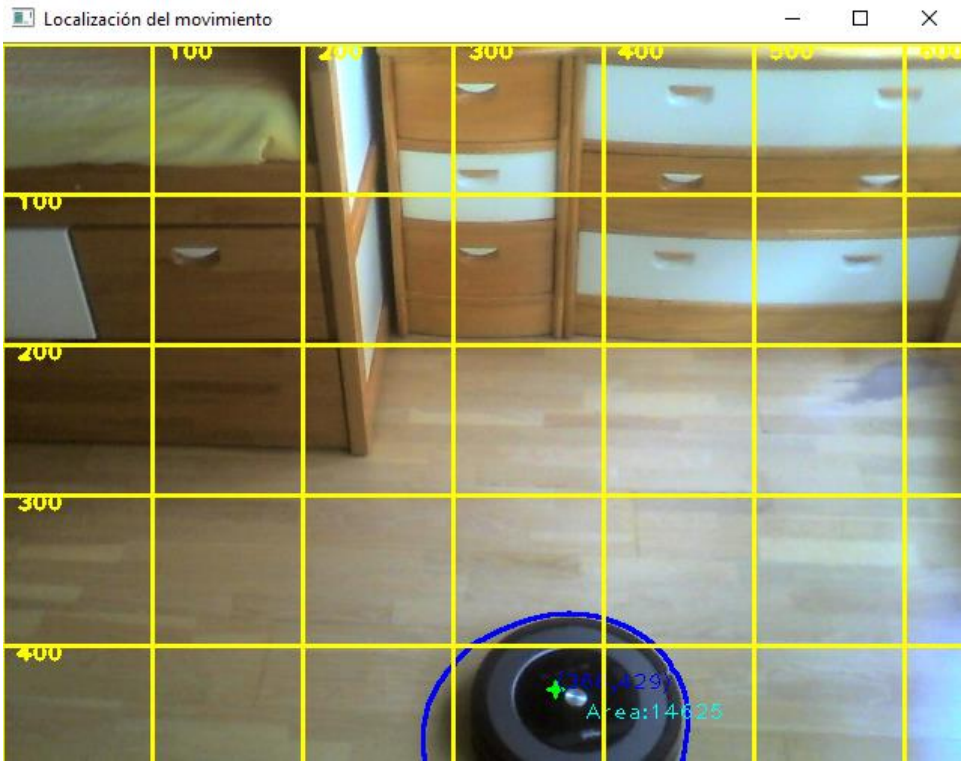


Figura 26. Obtención de matriz de tamaño.

| | | x | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 0-100 | 100-200 | 200-300 | 300-400 | 400-500 | 500-600 | 600-650 |
| y | 0-50 | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* |
| | 50-100 | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* |
| | 100-150 | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* |
| | 150-200 | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* |
| | 200-250 | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* |
| | 250-300 | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* |
| | 300-350 | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* |
| | 350-400 | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* |
| | 400-450 | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* | *AreaBaseMAX* *AreaBaseMIN* |

Tabla 1. Relación de áreas y posición.

Volviendo con el programa de visión artificial principal, la detección del elemento deseado sigue una estructura determinada que se procede a explicar a continuación.

Como ya se ha explicado en el tratamiento anterior, mediante la función *findContours* se pueden detectar todos los contornos que se encuentran en la imagen, los cuales se guardan

en el vector denominado *contours.* Tras ejecutar está función, el programa se puede encontrar con tres posibilidades:

- Que no detecte ningún contorno. En ese caso, en el video no hay ningún objeto en movimiento, por lo que no es necesario realizar ningún tratamiento más en la imagen.
- Que detecte tan sólo un contorno. En este caso, se pueden dar dos nuevas posibilidades:
    - El contorno detectado hace referencia al objeto que se está buscando, es decir, el Roomba 866.
    - El contorno detectado hace referencia a otro objeto que está en movimiento, ya sea una mascota, persona o cualquier otro objeto.
- Que detecte varios contornos. En este último caso se pueden dar de nuevo dos posibilidades, a saber:
    - Ninguno de los contornos detectados es el objeto buscado.
    - Uno de ellos es el objeto buscado y el resto no lo son.

Ahora bien, ¿cómo se puede saber en C++ si se ha encontrado algún contorno?

El parámetro *contours* es un vector, por lo que conociendo su tamaño se sabrá el número de contornos detectados. De forma específica, el programa en primera instancia observa si en el video hay algún objeto en movimiento mediante el parámetro *contours.size* y con la variable booleana *movimiento_detectado* de forma que:

- Si *contours.size* es mayor que cero, implica que en el vector hay al menos un contorno guardado, haciendo que la variable *movimiento_detectado* tome el valor *true*.
- Si *contours.size* es menor o igual que cero, implica que en el vector no se ha almacenado ningún dato, haciendo que la variable *movimiento_detectado* tome el valor *false*.

Una vez se sabe si el programa ha detectado o no movimiento en el video, se continúa con el tratamiento.

En caso de no haber encontrado ningún objeto en movimiento no se realiza nada más, pero en caso contrario, es necesario saber si entre estos contornos detectados se encuentra el robot que se busca. Para ello, para cada contorno detectado se realiza el siguiente estudio:

- Se calcula el área del contorno en cuestión mediante la siguiente función:

    double **contourArea**(InputArray **contour**, bool **oriented**=false )

    La función *contourArea* es utilizada para calcular el área de un contorno. Está formada por dos parámetros:

    - Contour, vector de entrada dónde está almacenada la información sobre el contorno que se quiere analizar.
    - oriented, sirve para indicar el formato en el cual está función nos da el valor del área. Puede tomar dos tipos de valores:

        - True, la función devuelve el valor del área en función de la orientación, sentido horario u anti-horario.
        - False, la función devuelve el valor del área en valor absoluto.

- Después se recuadra el contorno, para así conocer los datos de su centro de coordenadas, tal y como se ha explicado en el punto anterior.
- En este punto, ya se conocen tanto las coordenadas como el área del contorno, por lo que ya se puede consultar en la matriz de tamaño si para las coordenadas en las que el robot se encuentra, el área está dentro del intervalo de áreas preestablecidas.

  La matriz de tamaño se hará mediante un conjunto de condicionales *if-else if.* Un ejemplo del mismo puede ser:

```
if (coordenadaY <= 50 && coordenadaY > 0)
{
	area_baseMIN = 2800;
	area_baseMAX = 3600;
}

else if (coordenadaY <= 100 && coordenadaY > 50)
{
	area_baseMIN = 2500;
	area_baseMAX = 4200;
}
else if . . .
```

  En función de las coordenadas del contorno, se establecen unas áreas base mínima y máxima. Para determinar si el área del contorno se encuentra dentro del intervalo de áreas seleccionado anteriormente se realiza un condicional *if* como el que se muestra:

```
if (area > area_baseMAX || area < area_baseMIN)
{
	// Ignorar objeto, no se ha encontrado el objeto deseado
}
else
{
	// Objeto encontrado.
}
```

  Si no se encuentra dentro del intervalo de áreas, ignora el objeto detectado.

  Si se encuentra dentro del intervalo, se dibuja el contorno, su centro y se indican las coordenadas del mismo, tal y como se ha indicado anteriormente.

En la Figura 27 se muestra un caso real de este proyecto en el que hay dos objetos en movimiento, el Roomba que nos interesa y otro robot de menor tamaño. Como se puede

observar, tan sólo contornea e indica las coordenadas del robot que interesa, ignorando el otro completamente.


*Figura 27. Búsqueda del objeto correcto.*

Por último, como se viene realizando en los tratamientos anteriores, se adjunta un diagrama de bloques del mismo, véase Diagrama 3.

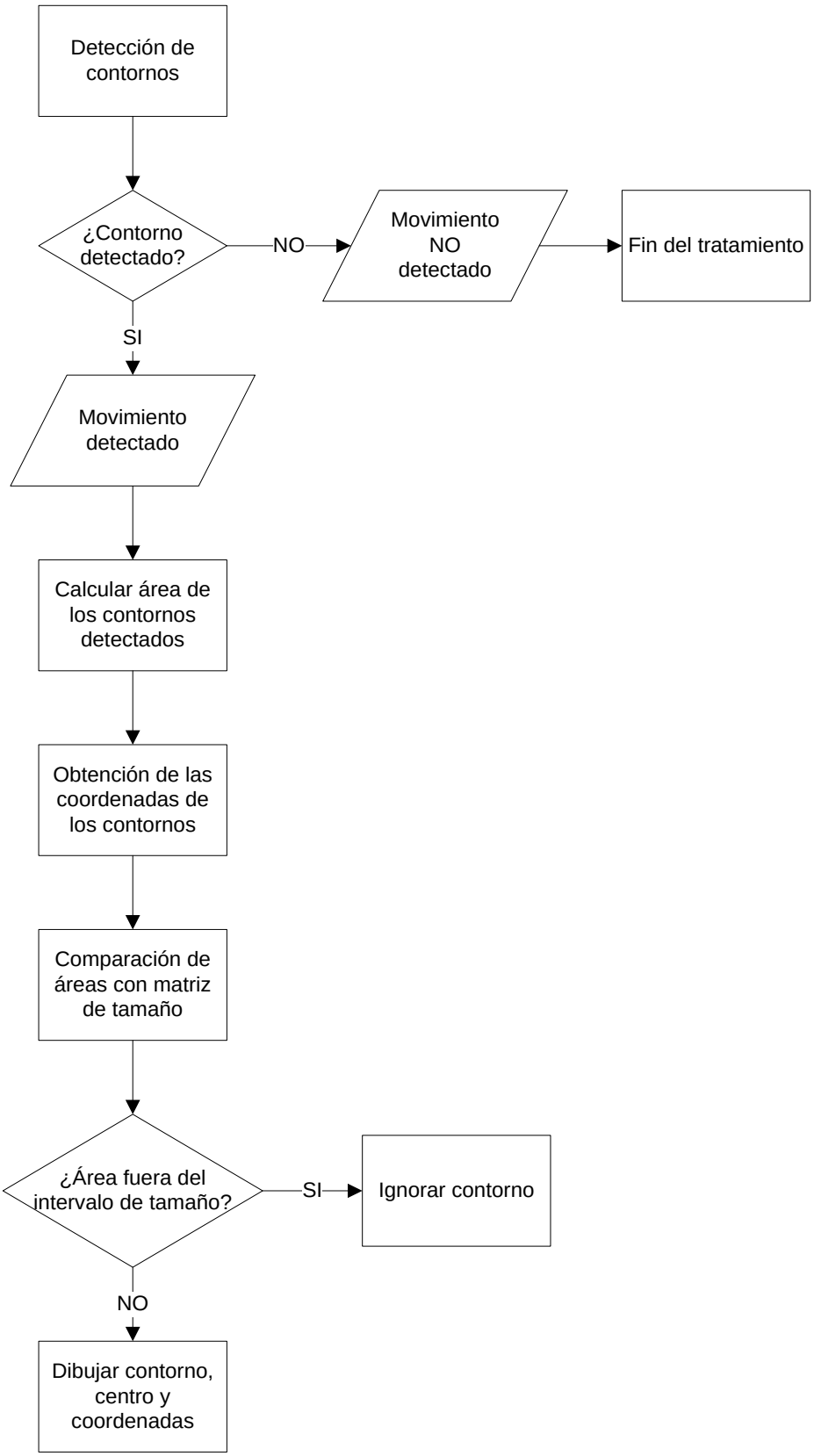


Diagrama 3. Tratamiento 3.

### *Tratamiento 4. Localización de las zonas prohibidas.*

Llegados a este punto, el programa ya es capaz de detectar el robot sin problemas, existan o no otros objetos en movimiento. También es capaz de rodear su contorno y de conocer la posición de su centro de coordenadas y su área. Es decir, lo que se ha conseguido hasta ahora es la obtención de los datos referentes al robot, por lo que el siguiente paso es la obtención de datos referentes a las zonas permitidas y prohibidas para el robot.

Para ello, el programa se ha diseñado de forma que el usuario pueda elegir las zonas prohibidas al tránsito del robot de forma manual, es decir, el usuario bordeara el contorno de esta zona prohibida mediante unos puntos.

Se le indicará al usuario con cuantos puntos quiere definir la zona prohibida, imponiendo un intervalo de entre 3 y 6. Indicado este valor, permitirá al mismo puntear tantas veces como puntos ha indicado en el paso anterior, ni uno más, ni uno menos.

Realizado este paso, el programa dibujará sobre la imagen patrón, de la cual ya se habló en el Tratamiento 1, el polígono que define la zona prohibida. Es importante tener en cuenta que el polígono sólo se puede dibujar en la imagen patrón y nunca en el video, si se hiciera de esta última forma, el programa lo detectaría como otro contorno dando lugar a errores.

A continuación se procede a explicar todo el procedimiento de forma más visual e indicando los detalles de la programación.

En primer lugar, al ejecutar el programa, aparece el mensaje que se observa en la Figura 28 en la ventana de comandos.

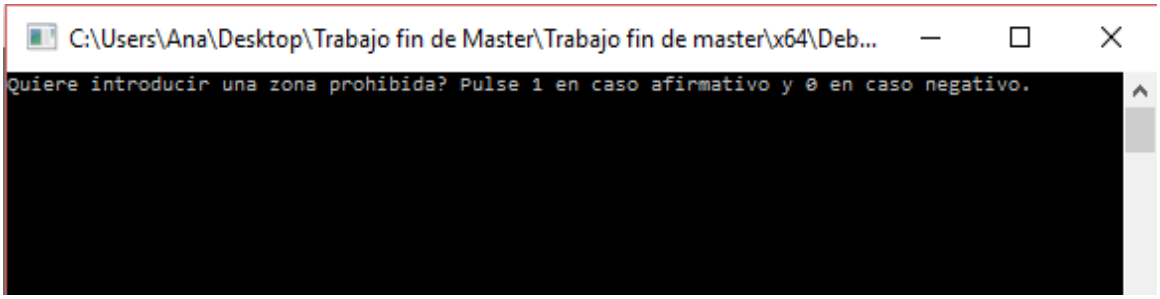


Figura 28. Introducción zona prohibida.

En caso de que se quiera introducir una zona prohibida, aparecerá el mensaje que se observa en la Figura 29.

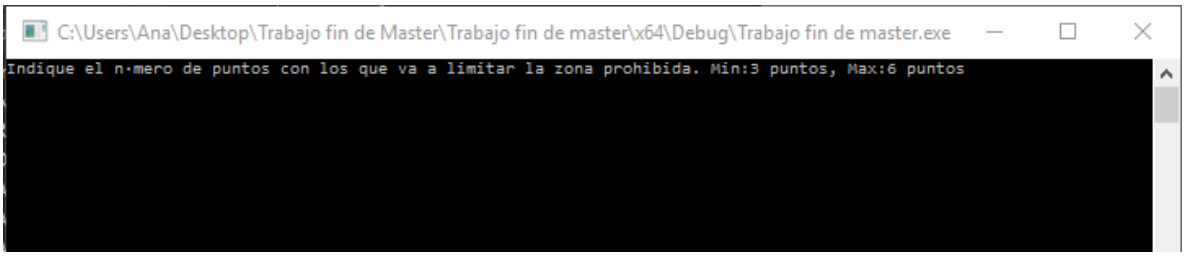


Figura 29. Ventana de comandos 1.

Se deberá introducir por el teclado el número de puntos con los cuales se definirá la zona prohibida, en el caso de este ejemplo, se ha optado por 5 puntos. Al introducir el valor, aparecerá un mensaje en el cual se indica que ha llegado el momento de dibujar los 5 puntos

con los que se definirá la geometría de la zona prohibida, tal y como se observa en la Figura 30.

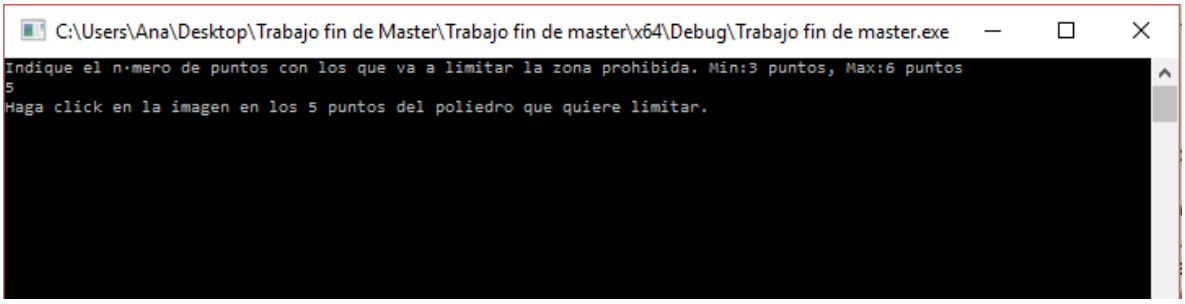


Figura 30. Ventana de comandos 2.

Todo el proceso de entrada y salida de información por la ventana de comandos se realizará mediante un programa secundario llamado *mensaje* que es llamado por el programa principal al inicio del mismo de la siguiente manera:

**mensaje**();

Como se puede observar, a dicha función no es necesario enviarle ningún tipo de información.

El programa *mensaje* sacará por pantalla los dos mensajes que se han visto anteriormente mediante la función:

int **puts** ( const char * **str** );

Está función está formada por un solo parámetro, *str*, el cual indica la cadena de caracteres que se quiere imprimir en la ventana de comandos.

Y leerá el número de puntos que se han introducido por el teclado mediante la función:

char * **gets** ( char * **str** );

Al igual que con la función anterior, esta tiene tan solo un parámetro, *str*. Se trata de un parámetro de destino en el cual se almacenará la información introducida por el teclado.

Antes de continuar con el resto del tratamiento, se adjunta el programa *mensaje* por si fuera necesario consultarle.

```
//Manda mensaje
int mensaje()
{
	puts("Indique el número de puntos con los que va a limitar la zona prohibida. Min:3 puntos, Max:6 puntos");


	scanf_s("%i", &num_puntos);


	printf_s("Haga click en la imagen en los %i puntos del poliedro que quiere limitar.", num_puntos);
	return 0;
}
```

En este punto del programa ya se ha salido del programa *mensaje* y se ha vuelto al programa principal, apareciendo la imagen patrón en la cual se deberán indicar los puntos que definirán la geometría del polígono haciendo clic con el botón izquierdo del ratón. Tras indicar todos los puntos, aparecerá la imagen patrón junto al polígono dibujado tal y como se observa en la Figura 31.

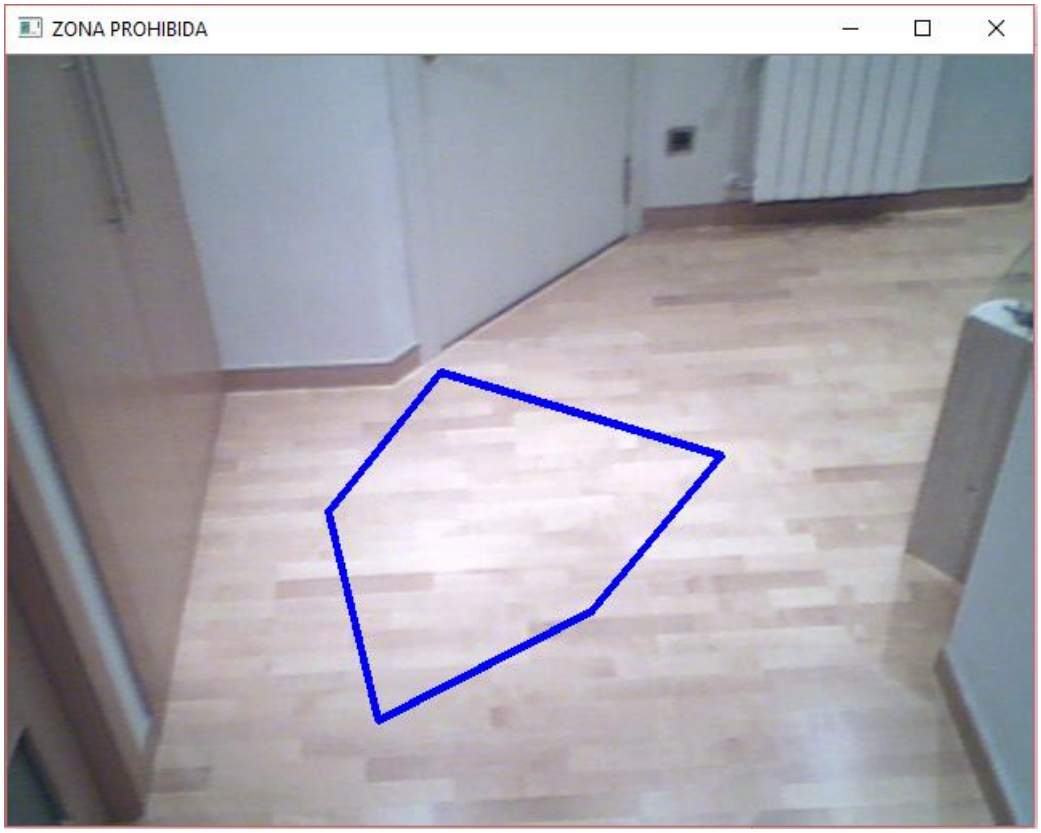


Figura 31. Contorno de la zona prohibida.

Para que la imagen patrón aparezca será necesario usar la función:

void **imshow**(const string& **winname**, InputArray **mat**)

Se trata de una función que muestra una imagen en una ventana especificada y consta de dos parámetros:

- Winname, es el nombre que se le asigna a la ventana.
- Mat, es la imagen que se va a mostrar en esa ventana.

Una vez mostrada la imagen, hay que tener en cuenta que en función de los puntos indicados en el paso anterior, el número de líneas a dibujar y el número de puntos que el programa se espera que sean clicados será diferente, por lo que en función de este número de puntos el programa variará ligeramente. Dicha diferenciación se ha realizado mediante condicionales *if*, y en cada uno de ellos se realiza lo siguiente:

- En primer lugar, se crea un vector para guardar las coordenadas x e y de cada uno de los puntos. En función del número de puntos que se vayan a dibujar el vector tendrá un tamaño u otro.

- A continuación, se llama a la función:

  void **setMouseCallback**(const string& **winname**, MouseCallback **onMouse**)

  Esta función sirve para controlar los eventos que se producen en una determinada ventana introduciéndole el siguiente par de parámetros:

  - Winname, es el nombre de la ventana donde se van a analizar los diferentes movimientos del ratón.
  - onMouse, es el nombre de la función que controlará los eventos.

  En la función *onMouse*, se va a hacer de nuevo una distinción mediante condicionales *if* en función del número de puntos que se quieran indicar, haciendo las siguientes acciones:

  - Creación de una variable tipo *Point*.
  - En el momento en el que se detecte un evento tipo *CV_EVENT_LBUTTONDOWN*, o lo que es lo mismo, cada vez que se pulse el botón izquierdo del ratón, se almacenará en la variable *point* las coordenadas x e y del lugar dónde se ha pulsado el ratón en la ventana *winname*.
  - Almacenados estos dos datos, se asignarán a unas variables globales llamadas *coordenadaX0, coordenadaY0, coordenadaX1, coordenadaX2…* Para saber en qué coordenada se deben asignar los datos, se ha creado un contador de forma que, si sólo se ha pulsado una vez el ratón, el contador tendrá valor 0 y las variables x e y se asignarán a las *coordenadaX0* y *coordenadaY0.* Si se ha pulsado el ratón por segunda vez, el contador pasará a tomar el valor 1 y las variables x e y se asignarán a las *coordenadaX1* y *coordenadaY1*, y así sucesivamente.

  Estas acciones se realizan de la misma forma en todos los condicionales salvo que en función del número de puntos a guardar, el tercer paso anteriormente indicado se realizará un mayor o menor número de veces.

- Hay que tener en cuenta que la función anterior funciona a partir de eventos realizados por el ratón, por lo que se necesita avisar de alguna manera a la función principal de que ya no necesitamos atender más a lo que haga el ratón y que podemos continuar con el resto del proceso. Para ello, se ha creado un bucle *while* que obligue a la función principal a esperar hasta que la variable contador sea igual al número de puntos que se quiere representar. La función espera es:

  void **waitkey** (time)

  Y realiza una pausa en el programa equivalente al número introducido en el parámetro *time* en milisegundos. Llegados a este punto, se continua con el resto del proceso.

- El proceso continúa eliminando la ventana donde se encontraba la imagen patrón mediante la función:

  void **destroyAllWindows**()

- Después se almacenan todos los valores de las variables *coordenadaX0, coordenadaY0, coordenadaX1, coordenadaX2…* en el vector citado en el primer punto.

- Por último, se dibujan un conjunto de líneas que unan los diferentes puntos mediante la función *line* ya explicada anteriormente. Y se muestra la imagen patrón con el borde del polígono resultante mediante la función *imshow*.

  Obviamente, el número de coordenadas a almacenar y el número de líneas a dibujar dependerá del número de puntos que se han querido definir, de aquí las pequeñas diferencias en cada condicional.

Para finalizar con la explicación, se adjunta el diagrama de bloques del tratamiento, véase Diagrama 4.

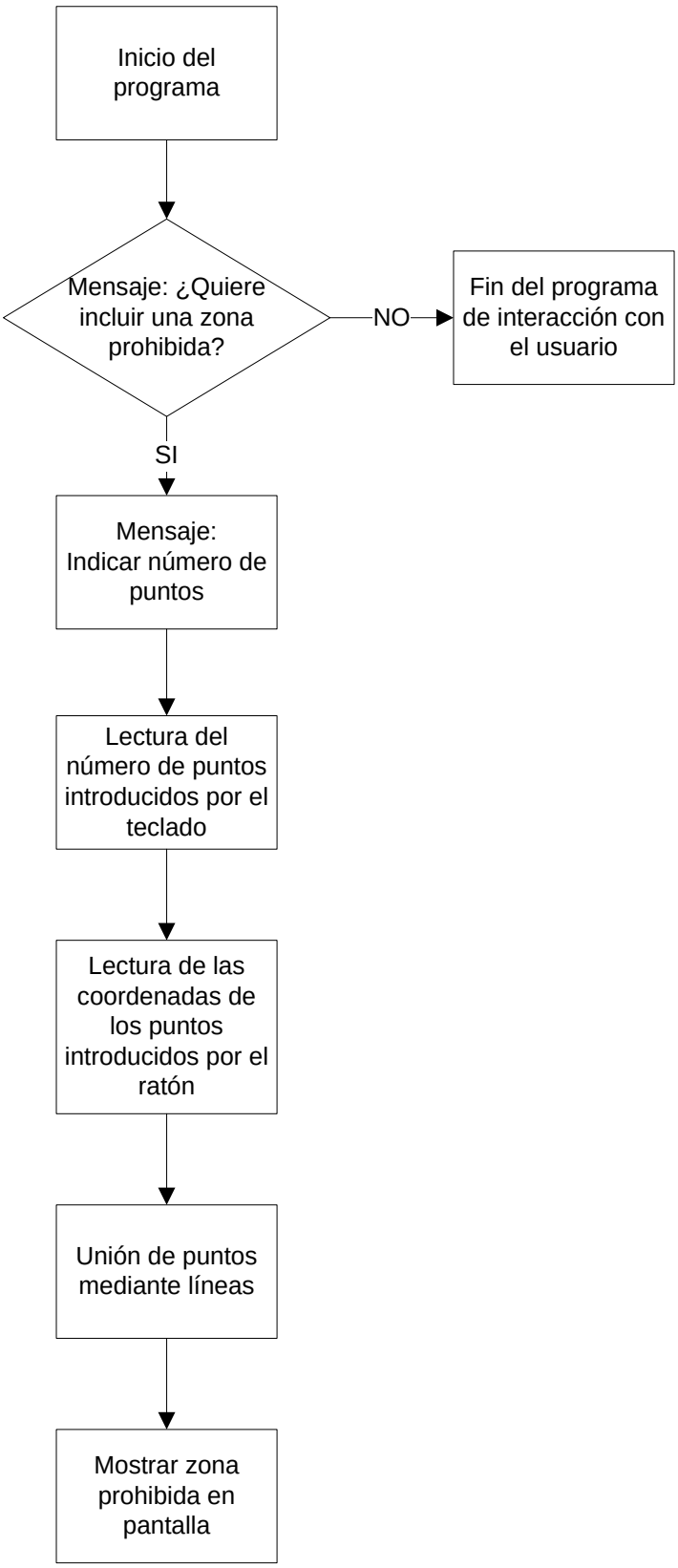


Diagrama 4. Tratamiento 4.

## *Tratamiento 5. Detectar si el robot se encuentra en zona permitida.*

Además de detectar el robot y conocer su área y posición dentro de la imagen, ya se puede introducir o detectar la localización de las zonas prohibidas dentro de la escena.

Todos estos parámetros son de vital importancia para conseguir el objetivo principal de este proyecto, la eliminación de elementos complementarios del Roomba para evitar que se mueva en determinadas zonas. Para ello, se necesita saber si el robot ha entrado o no en estas zonas, problema que se resolverá en este tratamiento.

Para saber si un punto del robot se encuentra en el interior de la zona prohibida será necesario contar cuantas veces una línea trazada desde este punto se cruza con el borde del polígono. Si lo cruza un número par de veces, el punto estará fuera y si lo cruza un número impar de veces, el punto estará dentro. En la Figura 32 se puede observar este razonamiento de manera más gráfica.

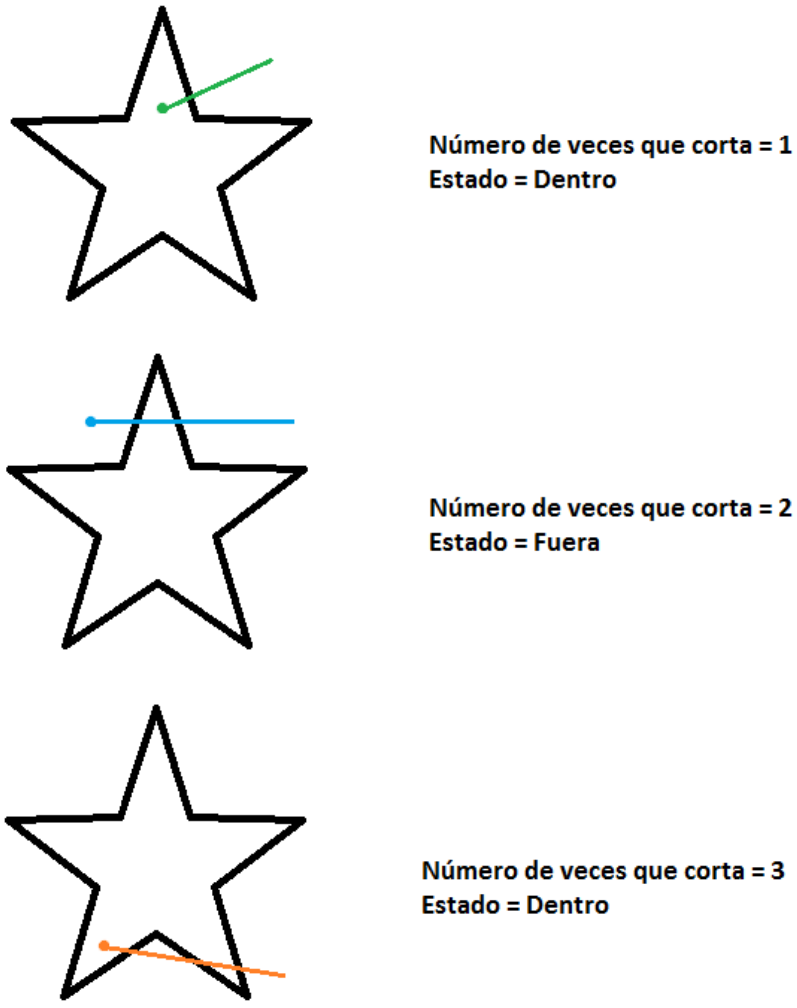


Figura 32. Representación estado del punto dentro del polígono.

Ahora la cuestión es, ¿Qué punto del robot se estudia para ver si está dentro o fuera del polígono?

Hasta el momento sólo se conoce el centro de coordenadas del robot, pero al tener la información de su contorno, se puede saber también su perímetro mediante la función:

double **arcLength**(InputArray **curve**, bool **closed**)

Se trata de una función que nos da información sobre el perímetro de una curva, en este caso, del contorno del robot. Está formado por dos parámetros:

- Curve, vector de puntos que forman la curva.
- Closed, es un parámetro con el que se indica si la curva está o no cerrada.

Conocida la longitud del perímetro se puede obtener el radio del "circulo" que envuelve al robot mediante la expresión:

$$radio = \frac{perímetro}{2 \cdot \pi}$$

Conocido el radio y el centro de esta circunferencia, se pueden tomar un número determinado de puntos del contorno, pudiendo conocer las coordenadas de cada uno de ellos mediante trigonometría:

$$Punto(X,Y) = Centro(X,Y) + radio \cdot (\sin(\alpha), \cos(\alpha))$$

Se ha decidido tomar 6 puntos colocados estratégicamente a lo largo del contorno, barriendo el mismo por completo cada 45°. En la Figura 33 se puede observar la posición de los 6 puntos citados.

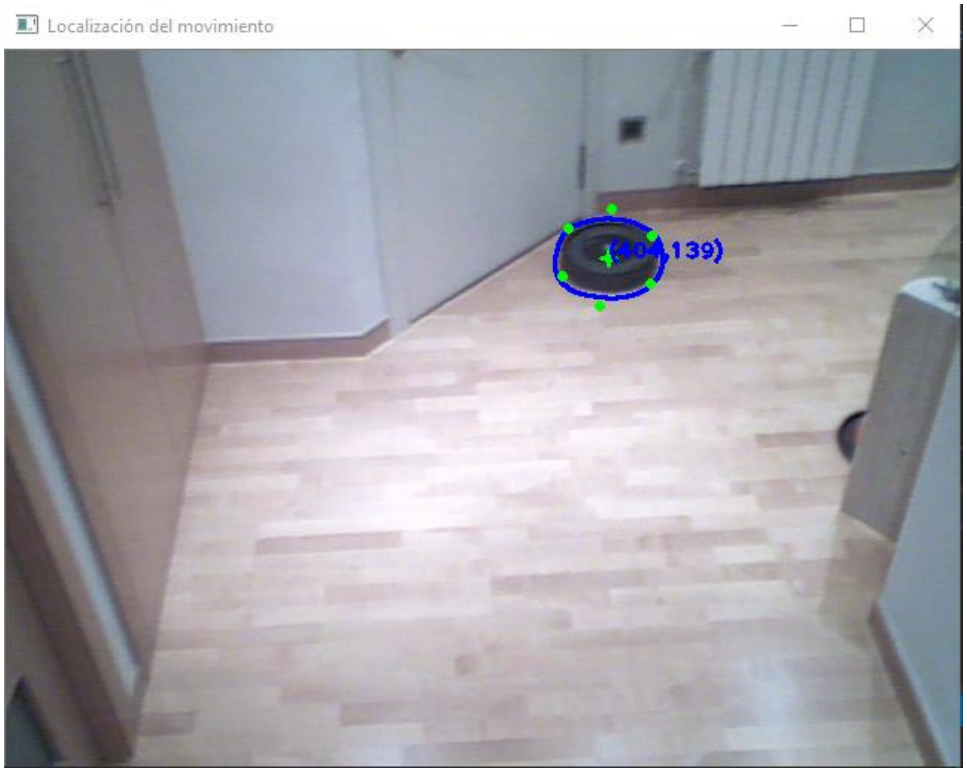


Figura 33. Puntos clave de estudio.

Con las coordenadas de estos 6 puntos ya se puede analizar si el robot está dentro de la zona prohibida. Con que uno de estos puntos se encuentre dentro, el programa indicará al usuario que el robot está dentro de esta zona.

Para conocer el número de cortes en formato C++ se ha recurrido al uso de condicionales *if*, adjuntando un ejemplo del mismo a continuación:

```
if (b > MIN(P0.y, P1.y))
{
	if (b <= MAX(P0.y, P1.y))
	{
		if (a <= MAX(P0.x, P1.x))
		{
			if (P0.y != P1.y)
			{
				xinters=(b - P0.y)*(P1.x - P0.x) / (P1.y - P0.y) + P0.x;

				if (P0.x == P1.x || a <= xinters)
					cortes++;
			}
		}
	}
}
```

Una vez conocidos el número de cortes, se verá si este número es par o impar con la siguiente estructura:

Numero_cortes % 2

Si el resultado de esta estructura es 0, significará que el número es par y en caso contrario, impar.

Como ya se ha dicho, se han tomado 6 puntos representativos del contorno, por lo que se debe estudiar el número de cortes para cada uno de sus puntos. Con que uno de ellos este dentro vale para considerar que el robot está en la zona prohibida.

En caso de que ninguno de los puntos tenga un número de cortes impar, es decir, el robot está dentro de la zona habilitada, el programa continúa con normalidad. En caso contrario, el programa indica por pantalla que el robot está dentro de una zona no habilitada.

En las Figuras 35 y 36 se pueden observar dos ejemplos reales de este proyecto en los cuales en un caso el robot se encuentra fuera de la zona prohibida y el programa continúa sin más y en otro caso, el robot se encuentra dentro de la zona prohibida y el programa nos indica con letras rojas que el robot se encuentra dentro. En la Figura 34 se ha indicado cual es la zona prohibida señalada.

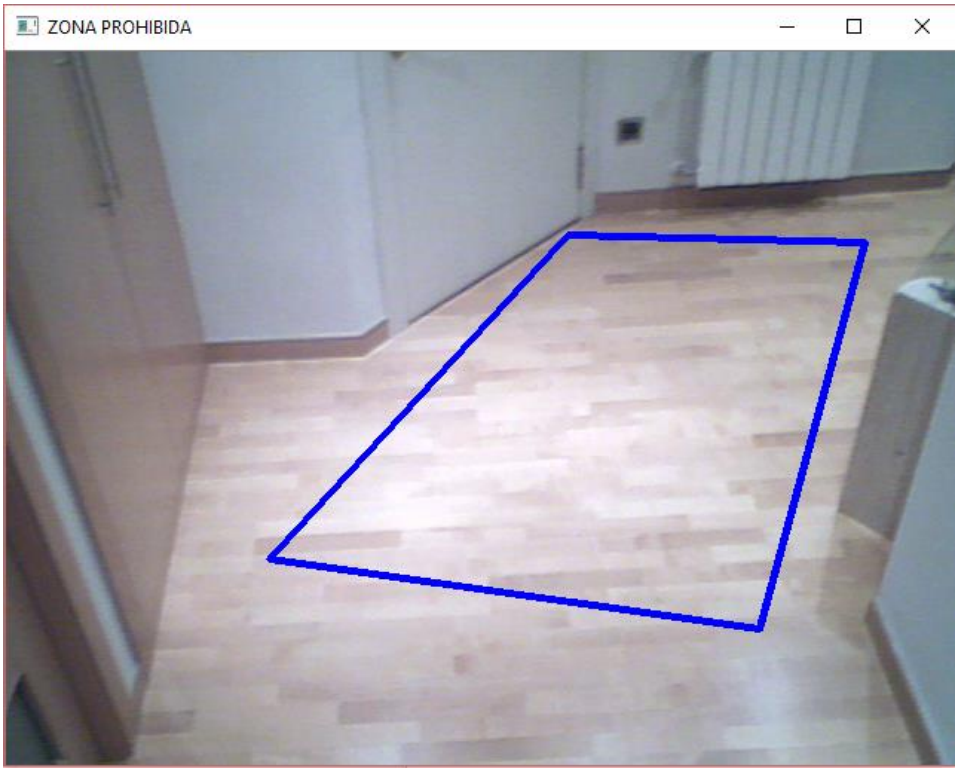


Figura 34. Zona prohibida.

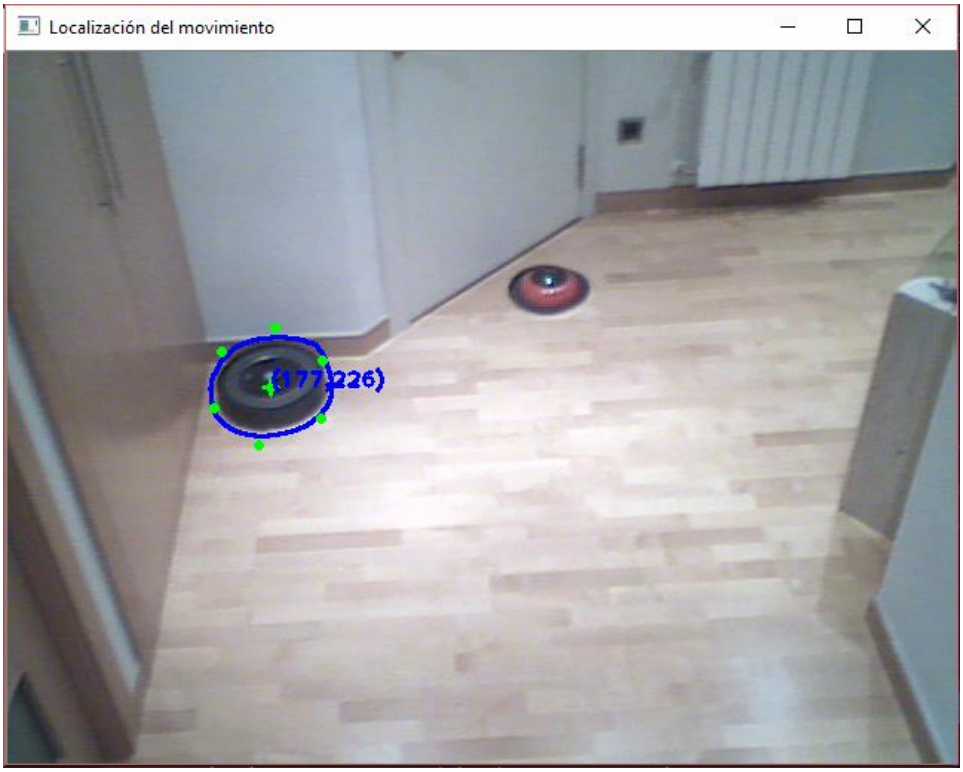


Figura 35. El robot se encuentra fuera de la zona prohibida.


Figura 36. El robot se encuentra dentro de la zona prohibida.

Por finalizar con el tratamiento se adjunta un diagrama de bloques del mismo, véase Diagrama 5.

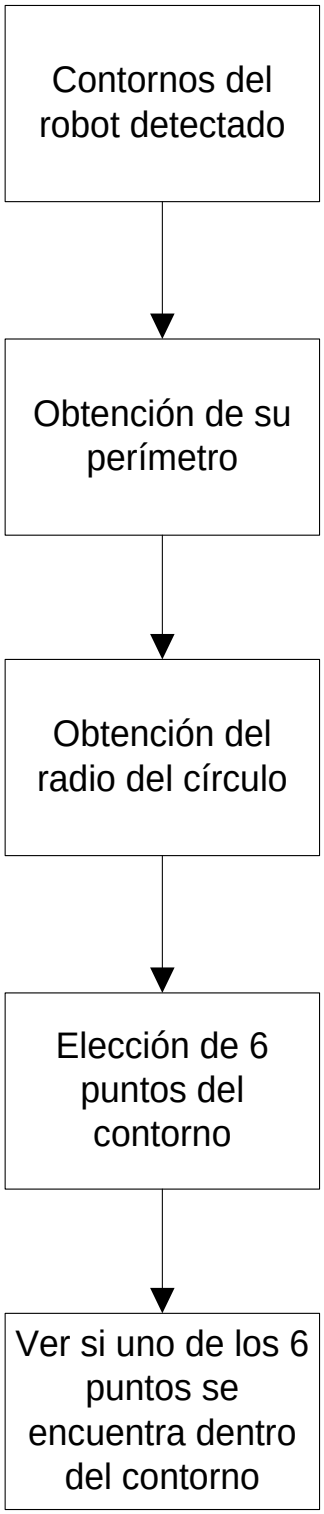


Diagrama 5. Tratamiento 5.

# 4. CONTROL DEL ROBOT

## 4.1. Introducción

En el capítulo anterior se habló de la parte de visión artificial del proyecto, de cómo se ha conseguido saber si el robot está entrando en una zona prohibida o por el contrario, detectar si el robot está limpiando por una zona permitida.

Todo este proceso de detección tiene un fin último, impedir que el robot aspire por zonas no deseadas por el usuario, por lo que una vez se ha detectado que está llegando a estas zonas, será necesario controlar el robot de forma que le alejemos de ellas.

Como la mayoría de los robots aspiradora pueden ser controlados por un mando a distancia que funciona por infrarrojos y los modelos que tienen paredes virtuales también funcionan emitiendo este tipo de señales, se ha decidido usar un emisor de infrarrojos conectado a un microcontrolador Arduino para realizar el control del mismo.

También existe la posibilidad de conectar el microcontrolador directamente al robot, y controlar al mismo mandándole señales al microcontrolador mediante señales bluetooth. Sin embargo, este tipo de control sólo es compatible con algunos modelos de aspiradoras Roomba, por lo que se deshecho está opción ya que el objetivo del proyecto es la localización y control de cualquier tipo de robot, independientemente de su marca y modelo.

A lo largo de este capítulo se va a proceder a la explicación del microcontrolador elegido así como el programa que hay en el mismo.

## 4.2. Elección del microcontrolador

Como ya se ha dicho, es necesario un control de bajo nivel que sirva de intermediario entre el controlador de alto nivel y el hardware del robot aspiradora. Este control de bajo nivel ha sido implementado por un microcontrolador Arduino.

Actualmente, en el mercado existe una gran variedad de microcontroladores Arduino, desde el famoso Arduino Uno hasta el Arduino Leonardo, pasando por el Arduino Nano, entre otros muchos. La principal diferencia entre todos ellos es el número de pines analógicos y digitales que tienen, por lo que la elección del mismo se realizará en función del número de pines que necesitemos.

Para el proyecto actual, tan sólo necesitamos conectar un emisor infrarrojos, por lo que serán necesario 3 pines, los cuales son:

- 1 pin digital tipo PWM.
- 1 pin GND.
- 1 pin 5V.

Sabiendo esto, se puede proceder a la elección del microcontrolador Arduino más adecuado para este proyecto.

Se han tenido en cuenta, a la hora de realizar esta elección los Arduinos Uno, Due, Leonardo, Mega 2560, y Nano.

Se ha realizado una tabla comparativa, véase Tabla 2, donde se muestran las características de estos microcontroladores que interesan en este proyecto, para facilitar el proceso de exclusión de microcontroladores y poder ver de manera más visual el microcontrolador adecuado.

| | | Características | | | | | |
|---|---|---|---|---|---|---|---|
| | | Pines Digitales | | | Pines Anal. | Puertos serie | Potencia |
| | | Normal | Con interrup. | PWM | | | |
| Tipo de Arduino | Uno | 14 | 2 | 6 | 6 | 1 | Baja |
| | Leonardo | 20 | 5 | 7 | 12 | 1 | Media |
| | Mega 2560 | 54 | 6 | 15 | 16 | 4 | Alta |
| | Nano | 14 | 2 | 6 | 8 | 1 | Baja |

Tabla 2. Tipos de Arduino.

Todos los microcontroladores cumplen con creces las necesidades del proyecto, por lo que con un Arduino Uno bastará para desarrollarlo ya que no son necesarios ni un número excesivo de pines digitales y analógicos, ni una potencia excesiva. Toda la información de este microcontrolador se encuentra adjunta en la sección de anexos *Anexo 1.- DataSheet Arduino Uno.*

En la Figura 37 se puede observar un microcontrolador como el usado en el proyecto, donde se encuentran detallados todos los pines que han sido utilizados para el control del emisor, adjuntado a la imagen una tabla, véase Tabla 3, con la misión que tiene cada conexión.

| Arduino Uno | |
|---|---|
| **Pin** | **Descripción** |
| 3 | Pin digital. Conexión con SIG emisor infrarrojos |
| GND | Conexión GND para el emisor |
| 5V | Conexión 5V para el emisor |

Tabla 3. Conexionado Arduino Uno.

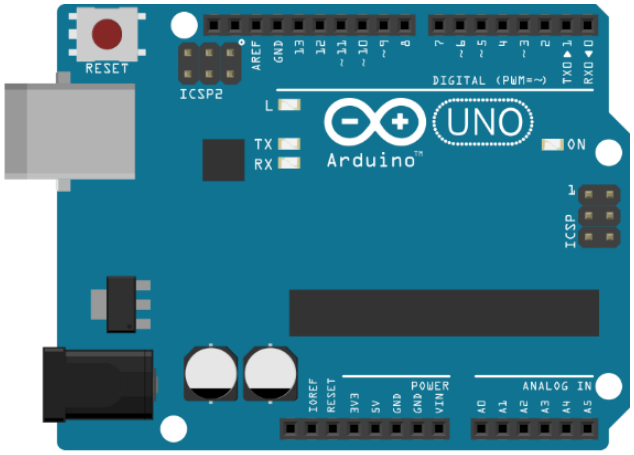


Figura 37. Arduino Uno.

## 4.3. Emisión de señales infrarrojas

Para realizar la emisión de señales infrarrojas será necesario en primer lugar conectar el emisor infrarrojos al microcontrolador. El emisor usado en este proyecto es el que se observa en la Figura 38.

Figura 38. Sensor Infrarrojos.

Como se puede observar, se trata de un led formado por dos patillas, a saber:

- Patilla corta, cátodo.
- Patilla larga, ánodo.

Se trata de un led de baja potencia, que añadido a la poca corriente que puede proporcionar el microcontrolador, resulta un dispositivo emisor de infrarrojos de muy bajo alcance. Para mejorar esto último, se ha añadido al circuito emisor un transistor NPN 2N2222, mejorando notablemente la distancia de emisión de señales, al amplificar la intensidad. La información de este transistor se encuentra adjunta al final del informe, en la sección de Anexo, *Anexo 2.- DataSheet transistor 2N222.*

Su esquema eléctrico es muy sencillo. Se debe alimentar el led emisor a través de las patillas Vcc y GND conectándolos a la salida de 5V y GND del Arduino, respectivamente. Se ha colocado una resistencia intermedia de 220 ohm para evitar que el emisor se dañe. Por último, se debe conectar el emisor de infrarrojos al colector del transistor, de esta forma, se aumentará la intensidad que pasa por el led, aumentando su distancia de acción. Por otro lado, el transistor además de estar conectado al emisor infrarrojo debe estar conectado al pin digital del microcontrolador en la base y al GND en el emisor. El conexionado del circuito se puede observar en la Figura 39.

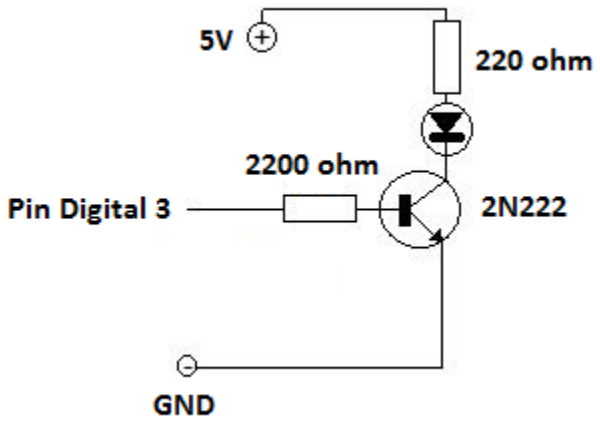


Figura 39. Conexionado Arduino.

Una vez realizado el conexionado, se procede a la programación del microcontrolador. Dicho programa lo que hace es que cada vez que el robot entra dentro de una zona prohibida envía al robot una señal mediante el emisor infrarrojos, de forma que este se aleje de la zona y no entre en ella.

Gracias a la existencia de un mando a distancia y de paredes virtuales que funcionan mediante emisión de infrarrojos, es sencillo conocer las diferentes longitudes de onda que es capaz de reconocer el robot, a saber:

- Movimiento hacia la derecha.
- Movimiento hacia la izquierda.
- Movimiento hacia delante.
- Modo Clean.
- Pared Virtual.

Tras probar con el propio mando del robot los diferentes modos de funcionamiento, se optó por usar solamente el modo de pared virtual, puesto que al tratar de controlar el desplazamiento hacia diferentes lados, al soltar el botón el robot se reseteaba por completo, caso que no ocurre cuando el robot detecta una pared virtual, alejándose suavemente de la zona no permitida con suavidad y sin movimiento bruscos.

El comportamiento de la pared virtual es muy sencillo. Consiste en un comportamiento repetitivo en el que se emiten ondas con una frecuencia de 38 kHz que permanecen un milisegundo encendidas y un milisegundo apagadas.

Cuando el robot detecta esta señal, véase Figura 40, se aleja y no traspasa la barrera creada.

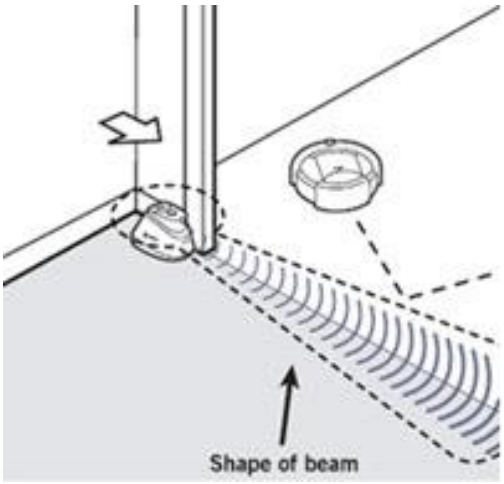


Figura 40. Funcionamiento Pared Virtual.

Para imitar este comportamiento con el Arduino, será tan sencillo como inicializar una librería de señales infrarrojas creada especialmente para Arduino, la cual obliga a conectar el sensor al pin digital 3, y caracterizarla de forma que siempre emita a 38 kHz.

$$\#include < IRremote >$$

$$\boldsymbol{IRsend}\ irsend;$$

Tras realizar esta inicialización, habrá que esperar a que el programa de Arduino le llegue una señal por parte del programa en C++ de visión artificial indicándole que el robot ha entrado en zona prohibida. Cuando le llegue dicha señal, representada por el valor "1", comenzará a imitar la señal de onda que emite la pared virtual, manteniendo la señal en alto durante 1 milisegundo y después manteniendo la señal en bajo durante otro milisegundo, de forma repetitiva, tal y como se observa en Figura 41. El programa se encuentra adjunto a continuación por si se requiere su consulta.

```
if( Serial.available()> 0)
{
    int palabra = Serial.read();  // Lectura de la señal enviada por C++

    // ENTRADA EN ZONA PROHIBIDA
    if (palabra == '1')
    {
       digitalWrite(13,HIGH);  // Encendido de un led indicador.
       for(int i=0;i<40;i++)
       {
         irsend.mark(1000);    // Señal en alto durante 1ms
         irsend.space(1000);   // Señal en bajo durante 1ms
       }
    }
    else
    {
      digitalWrite(13,LOW);    // Apagado de un led indicador

      //No se envía ningún tipo de señal
    }
}
```

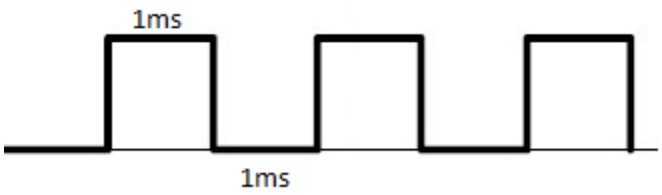


Figura 41. Señal Pared Virtual.

Cuando el robot detecte esta señal, el mismo girará hasta que deje de detectar la señal y continuará aspirando hacía delante.

Comentar, que también se han conseguido simular todos y cada uno de los botones del mando, por si de cara a un futuro proyecto que continúe este mismo, se quisiera por ejemplo teledirigir el robot desde el móvil.

El funcionamiento de este último programa se podría esquematizar de la siguiente forma:

1. Lectura del código por puerto serie. Cada código simula un botón diferente del mando.
2. En función del código recibido, una función externa decodifica el tipo de pulso que se debe mandar.
3. Envía esta señal infrarroja a 38kHz.

Ambos programas se pueden encontrar adjuntos al final de la memoria en la sección de anexos, *Anexo 3.- Programa Arduino, Pared Virtual* y *Anexo 4.- Programa Arduino, Mando.*

## 4.4. Comunicación Arduino – C++

Ya se tienen los programas de visión artificial y de Arduino por separado, por lo que el último paso es comunicarles para dar por finalizado el proyecto.

Para ello será necesaria la instalación de una nueva librería que permita conectarnos con el microcontrolador llamada, *SerialClass.h*.

Dentro del programa principal también será necesario indicar el puerto serie al cual está conectado el Arduino de la siguiente manera:

**Serial*** Arduino = new **Serial**("COM3")

Para finalizar, se debe mandar al Arduino la palabra "1" cuando se detecte que el robot se encuentra dentro de la zona prohibida y "0" en caso contrario. Hay que tener en cuenta que cuando se detecte que el robot está dentro de una zona no permitida, si se emite la señal de forma continuada, el robot empezará a dar vueltas sin salir de este bucle. Para evitar esto, cuando está dentro, se manda la palabra "1" cada cierto tiempo, para dar la posibilidad de alejarse al robot. Para mandar estos mensajes, en primer lugar se debe verificar que el Arduino está conectado mediante el condicional:

**while** (Arduino->IsConnected())

Mientras está condición sea cierta, se mandará el código "1" o "0" mediante la siguiente función incluida en la librería anteriormente mencionada:

**Arduino -> WriteData**(palabra, sizeof(palabra)-1)

Se trata de una función que manda información al Arduino que se encuentra conectado al puerto indicado. Está formado por dos parámetros:

- palabra, es una variable tipo *char* y que en este caso será el "1" en caso de que el robot se encuentre en la zona prohibida o "0" en caso contrario.
- Sizeof(palabra)-1, es un parámetro que hace referencia a la longitud del dato que se manda.

## 4.5. Diagrama de bloques final

Concluida la comunicación entre los dos programas, ya se puede adjuntar el diagrama de bloques completo del proyecto, véase Diagrama 6.

De forma esquemática, la programación se podría resumir en los siguientes puntos:

- Carga de vídeo y tratamiento de la imagen patrón.
- Creación de la zona no transitable por el robot aspiradora.
- Tratamiento de las capturas de vídeo
- Resta de imágenes y tratamiento del resultado.
- Detección de bordes y obtención de sus propiedades.

- Comparación del tamaño del contorno con la matriz de tamaños.
- Determinación de si el elemento detectado es el robot buscado. De ser así:
    - Representación gráfica del contorno y su centro de coordenadas.
    - Obtención de las propiedades del contorno.
    - Selección de 6 puntos del contorno.
    - Averiguar si alguno de los puntos está dentro de la zona prohibida. En caso afirmativo, mandar la señal "1" al Arduino de forma intermitente, en caso contrario, mandar la señal "0".

Antes de comenzar con los resultados experimentales, comentar que al final de la memoria se encuentran en el apartado de Anexos tres programas de visión artificial:

- El programa de visión artificial de tratamiento. *Anexo 5.- Programa de tratamiento de Visión Artificial.*
- El programa para obtener la información sobre la matriz de tamaño. *Anexo 6.- Programa matriz de tamaño.*
- El programa para realizar foto de la escena. *Anexo 7.- Programa fotografía.*

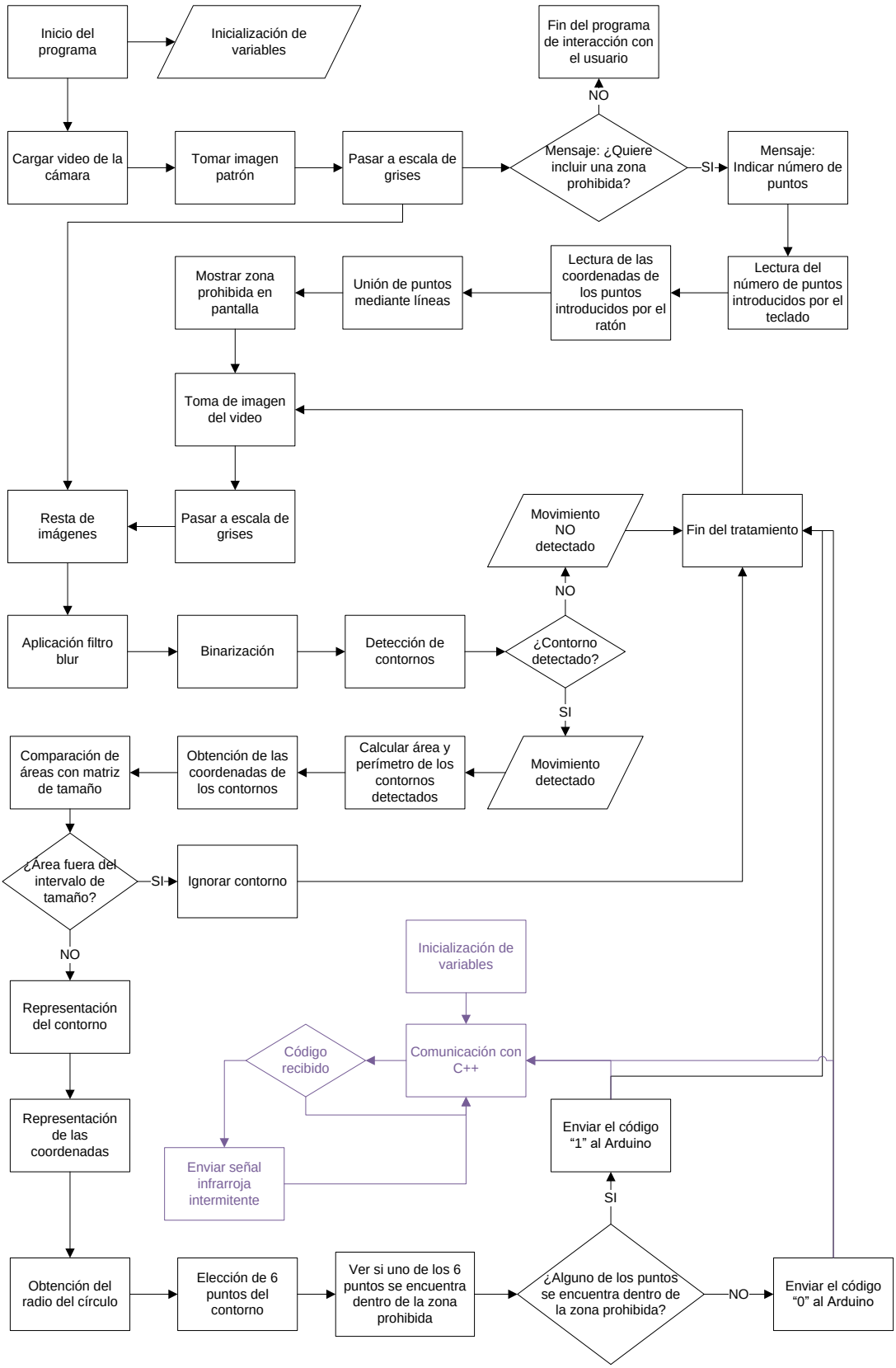


*Diagrama 6. Diagrama de bloques completo.*

# 5. RESULTADOS EXPERIMENTALES

A lo largo de los capítulos anteriores, se han ido describiendo y desarrollando los diferentes códigos realizados. Se ha hablado de códigos escritos tanto en C++ como en Arduino y se han ido escribiendo cada uno de los códigos de forma secuencial y aislada, de forma que no se continuará avanzando hasta no haber comprobado el correcto funcionamiento.

Sin embargo, en ningún momento se han descrito los resultados obtenidos al poner a prueba todos y cada uno de los códigos. Por este motivo, se ha realizado el presente capítulo para abordar todos estos resultados.

## 5.1. Resultados experimentales sobre videos

En primer lugar, los resultados obtenidos se realizaron sobre videos ya grabados previamente y no sobre videos en directo, con el fin de probar previamente los programas según se iban desarrollando.

Para poder abarcar el mayor abanico de posibilidades, se realizaron videos de todo tipo, a saber:

- Video en el que aparece el robot en una sala vacía.
- Video en el que aparece el robot con una alfombra.
- Video en el que aparece el robot con una silla que refleja.
- Video en el que aparece el robot en una sala vacía junto a otro robot aspiradora.
- Video en el que aparece el robot con una alfombra junto a otro robot aspiradora.

Como se puede observar, en cada video se va incrementando la dificultad de detección del robot, por lo que se fueron probando los diferentes tratamientos en el orden citado para de esta manera ir depurando el código progresivamente.

A continuación se va a mostrar gráficamente cómo se probaron todos los códigos en todos los videos mencionados y como los resultados obtenidos fueron satisfactorios.

### *Detección del movimiento*

Como se puede observar en las siguientes imágenes adjuntas, el robot es detectado correctamente, ya esté el robot sólo en una habitación sin obstáculos o con ellos, sean de superficie reflexiva o no. En caso de que existan dos robots en movimiento detecta ambos, que se traten o no luego se verá en los resultados experimentales del siguiente apartado.

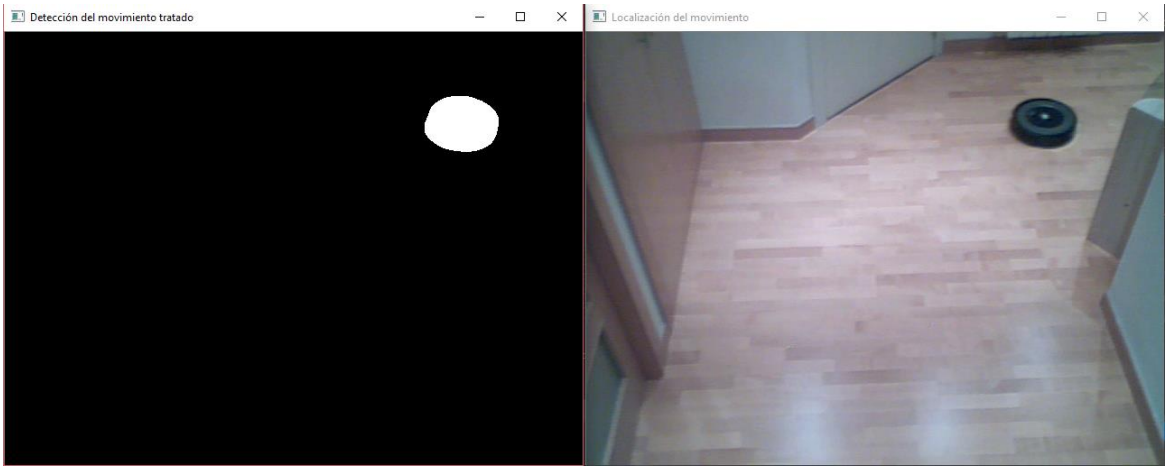


*Figura 42. Detección del movimiento. Caso 1.*

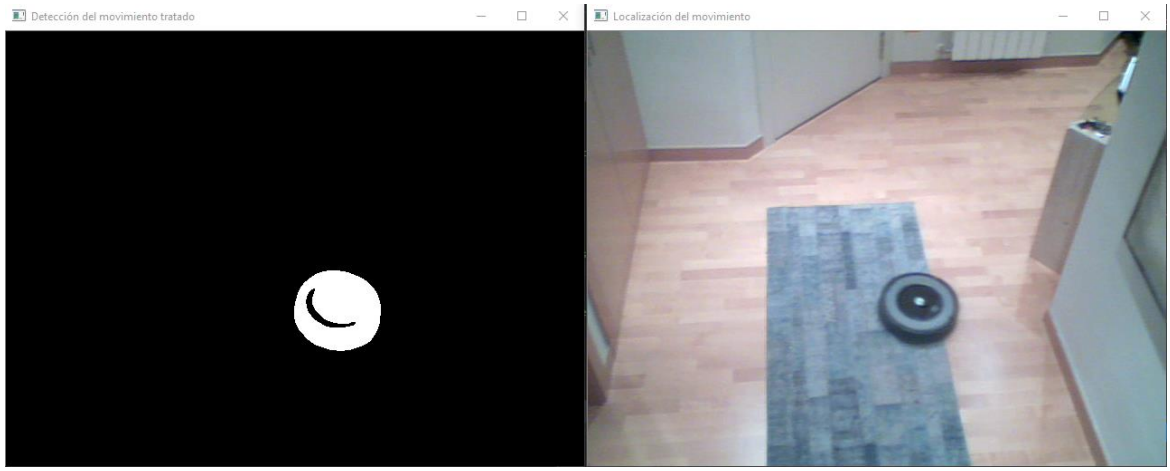


*Figura 43. Detección del movimiento. Caso 2.*

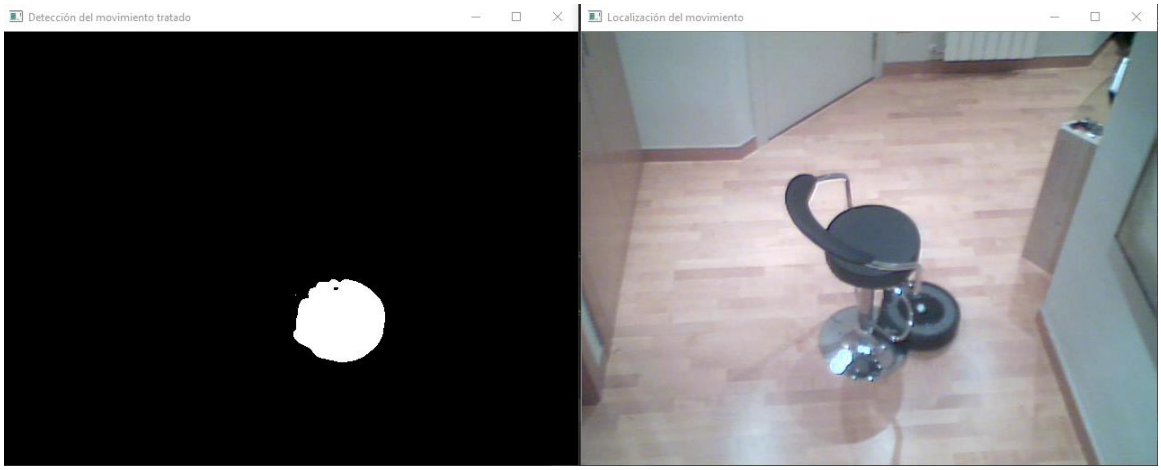


*Figura 44. Detección del movimiento. Caso 3.*

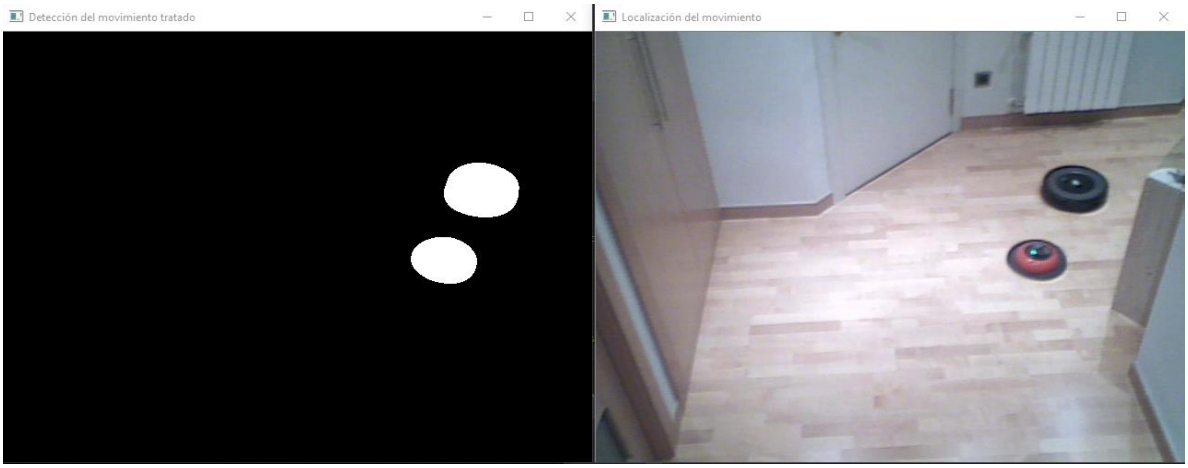


*Figura 45. Detección del movimiento. Caso 4.*

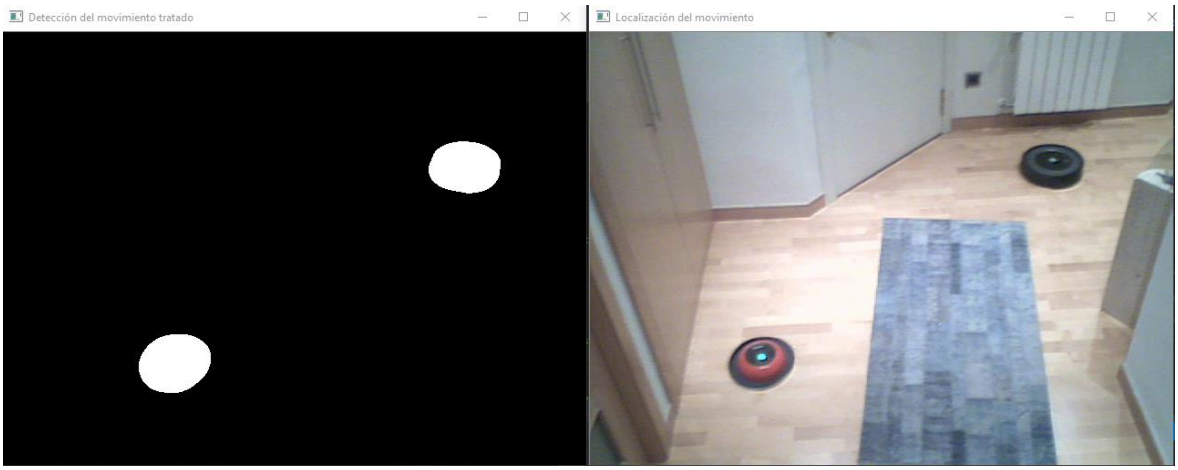


Figura 46. Detección del movimiento. Caso 5.

## *Tratamiento del objeto adecuado*

Conseguido que en todos los casos detecte el movimiento de cualquier objeto, hay que asegurarse de que ese objeto es el buscado. Ya que como se ha visto en la enumeración de los videos tratados, en la sala puede haber más objetos en movimiento que el que se desea tratar. Este paso es muy importante que salga correctamente, ya que de detectar el objeto inadecuado, se puede mandar una señal equivocada al robot y entrar en zonas no deseadas.

A continuación, se adjuntan una serie de imágenes donde se muestra que efectivamente, siempre se detecta el robot correcto.

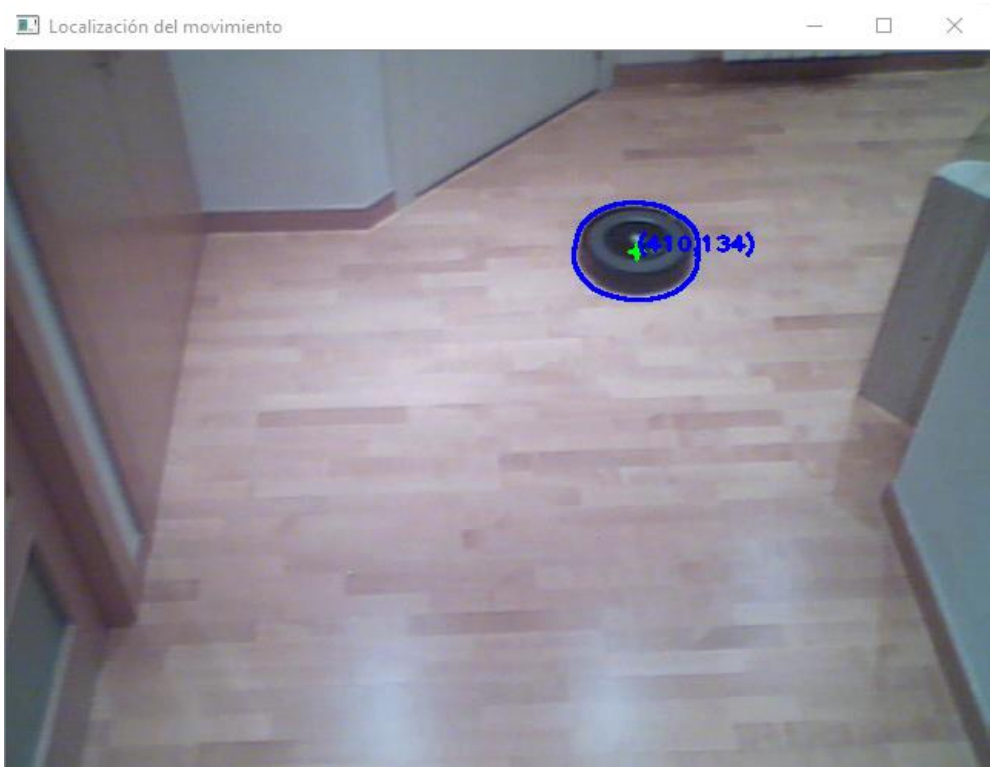


Figura 47. Tratamiento del objeto adecuado. Caso 1.

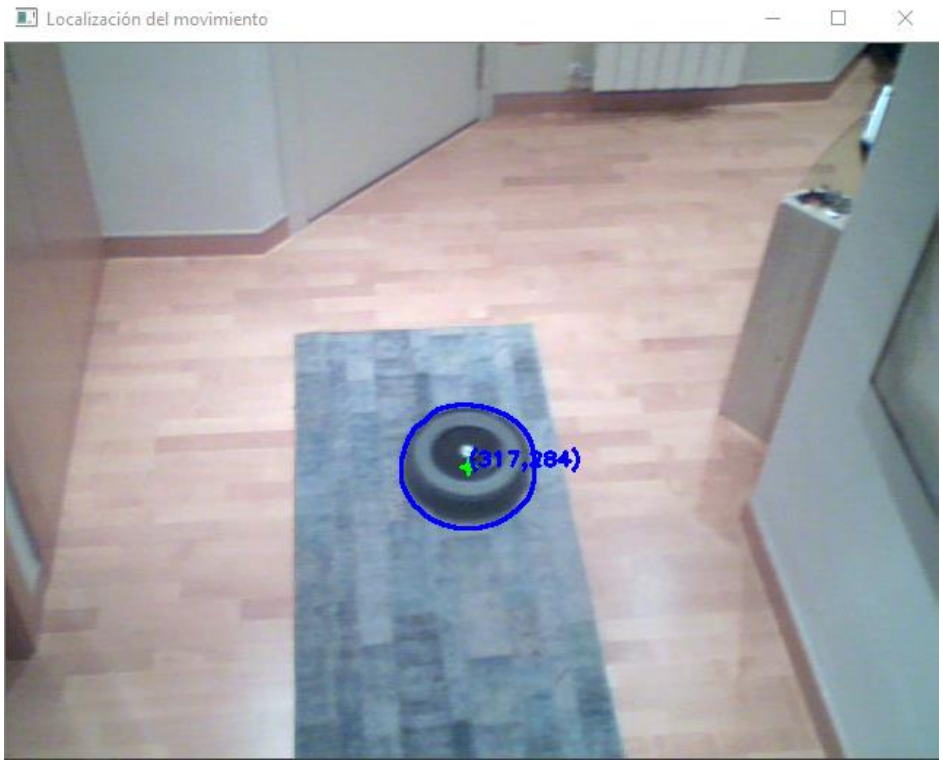


Figura 48. Tratamiento del objeto adecuado. Caso 2.

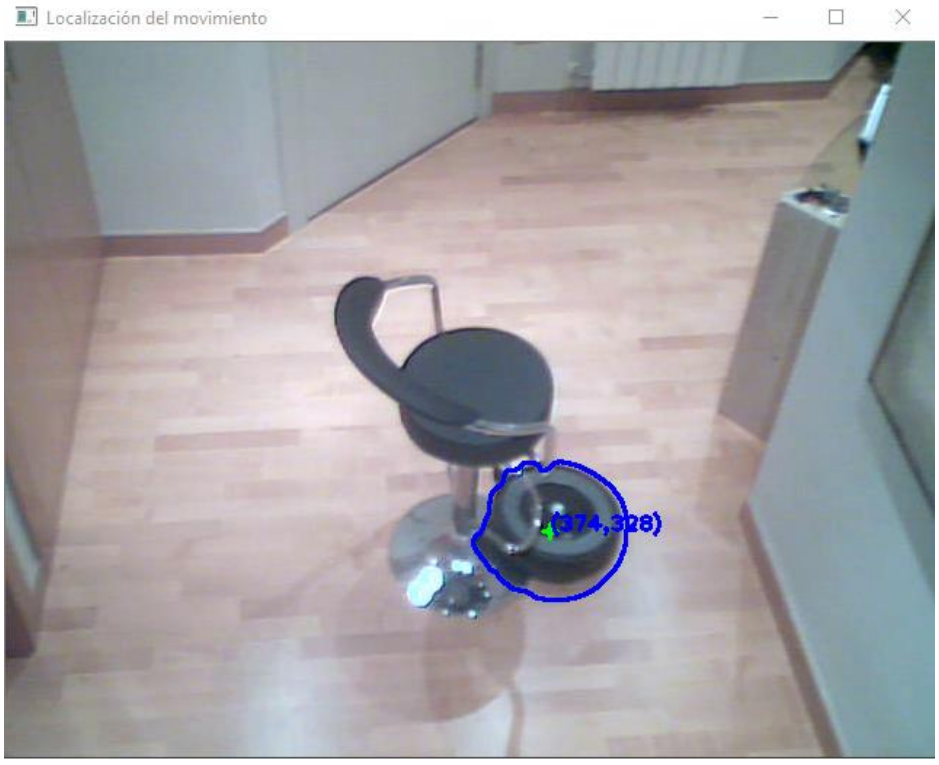


Figura 49. Tratamiento del objeto adecuado. Caso 3.

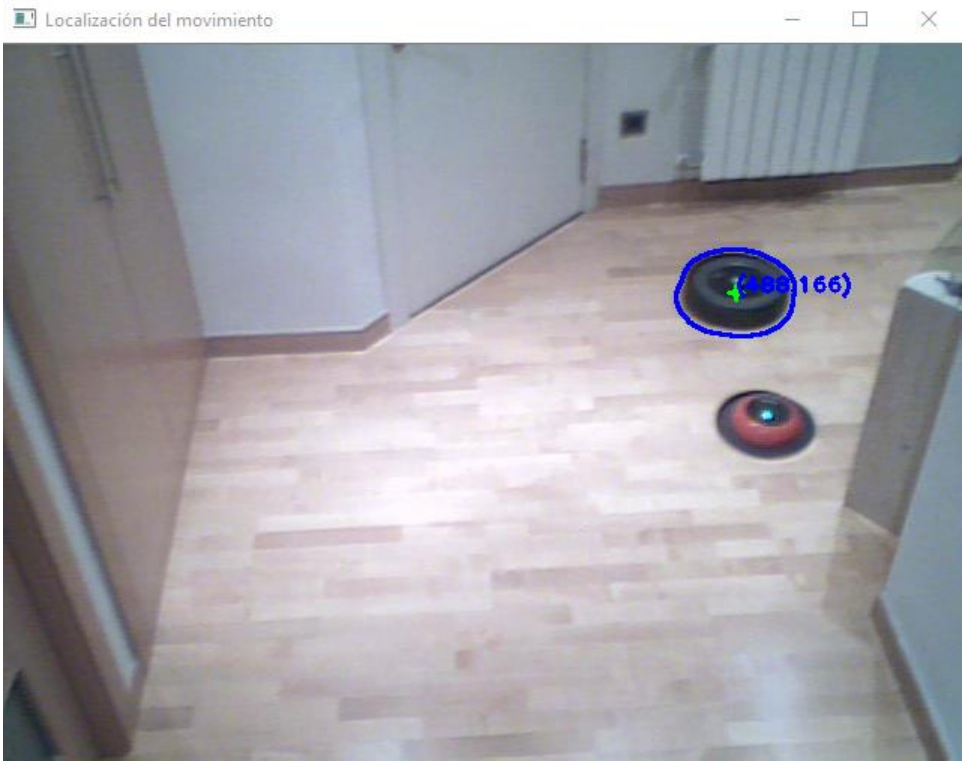


Figura 50. Tratamiento del objeto adecuado. Caso 4.

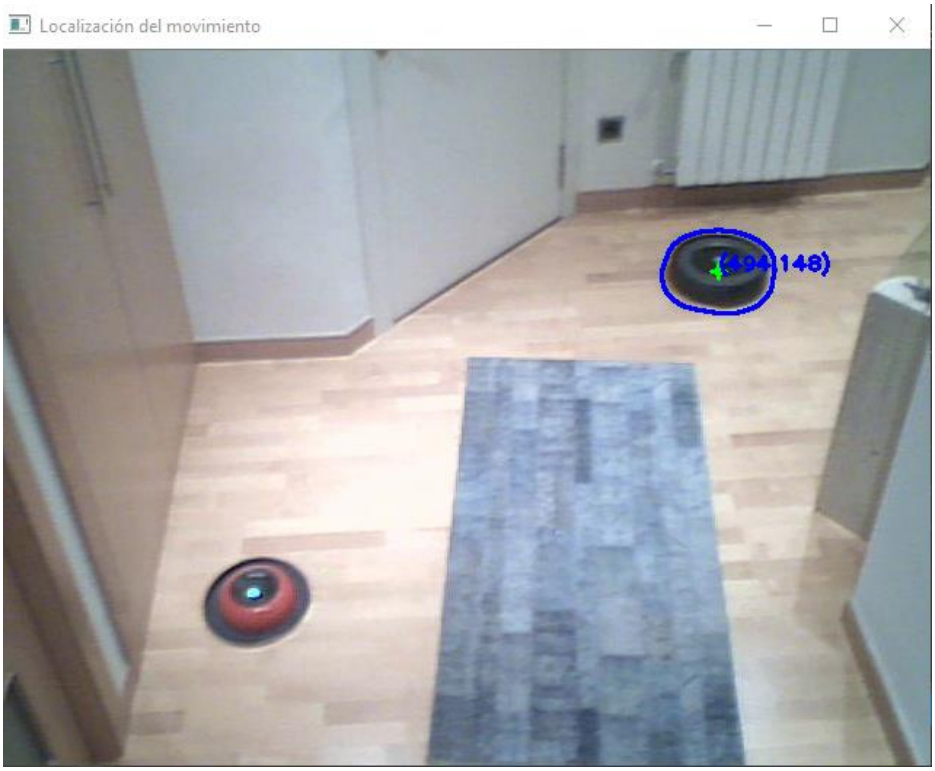


Figura 51. Tratamiento del objeto adecuado. Caso 5.

## *Detección de lo que ocurre en cada zona*

Una vez se ha mostrado que siempre detecta el movimiento y que siempre detecta el robot correcto, hay que verificar si el programa es capaz de identificar que el robot está caminando por la zona habilitada para el mismo. En caso contrario, se debe mandar una señal infrarroja que al tratarse de pruebas experimentales sobre videos, se ha sustituido el envío de esta señal por la indicación de un mensaje por pantalla.

En las siguientes imágenes se muestra como estando el robot únicamente en la habitación o con obstáculos o incluso con otro robot limpiando por ella, identifica si el robot buscado se encuentra dentro o fuera de la zona no habilitada.

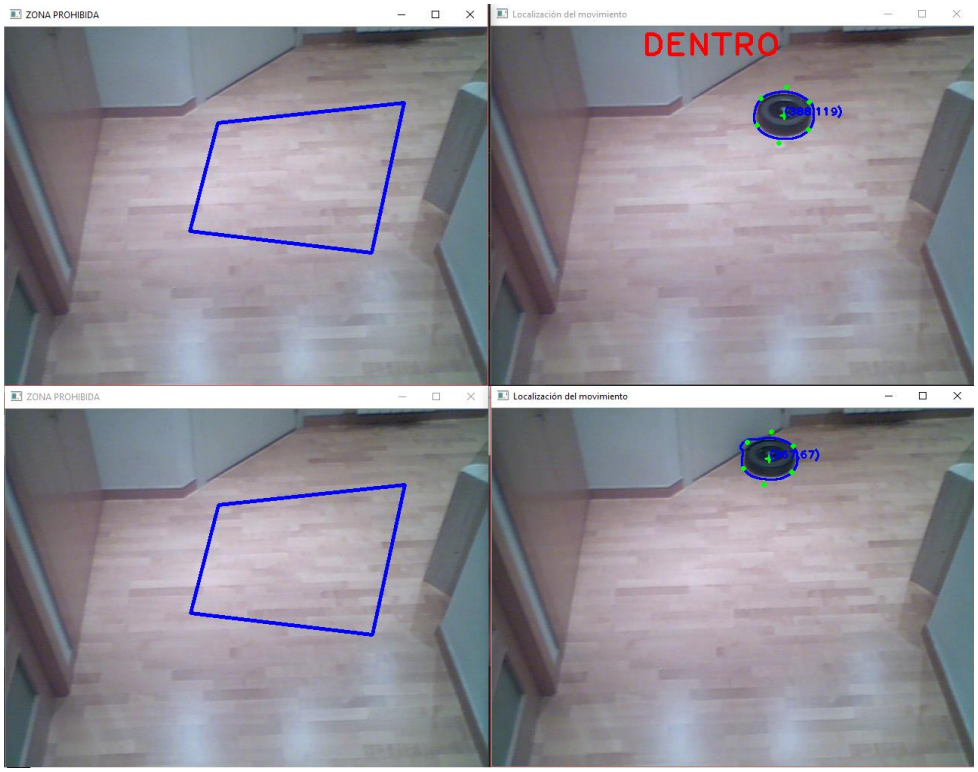


Figura 52. Detección de lo que ocurre en zona. Caso 1.


Figura 53. Detección de lo que ocurre en zona. Caso 2.

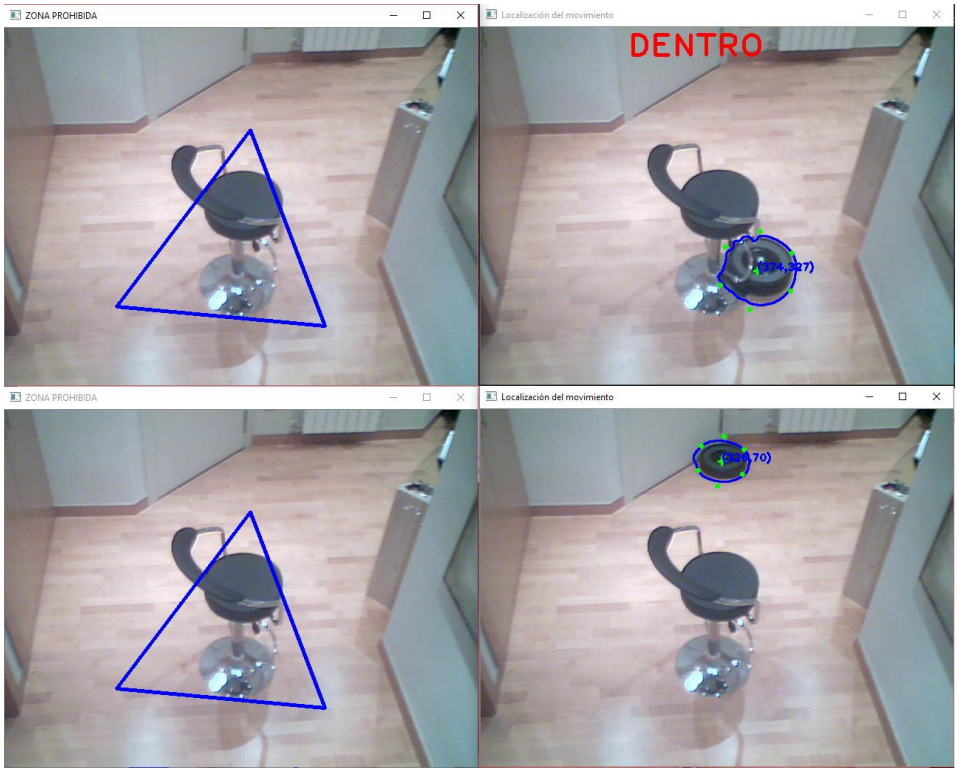


Figura 54. Detección de lo que ocurre en zona. Caso 3.


Figura 55. Detección de lo que ocurre en zona. Caso 4.


Figura 56. Detección de lo que ocurre en zona. Caso 5.

### *Comunicación Arduino – C++*

Por último, antes de comenzar a probar los programas en tiempo real, se ha comprobado la correcta comunicación entre el Arduino y el programa de visión artificial.

En primer lugar se comprobó la comunicación con un par de programas muy sencillos. Por un lado, el programa realizado en Visual Studio mandaba un "1" o un "0" por el puerto serie al Arduino, y por otro lado, el Arduino al recibir un "1" o un "0" encendía o apagaba un led, respectivamente.

Al ver que este programa funcionaba correctamente, se dio el siguiente paso, implementar esta comunicación sobre el programa de visión artificial desarrollado en Visual Studio. Se implementó de forma que al detectar que el robot se encontraba dentro de la zona prohibida, mandará un "1" al Arduino y en caso contrario, un "0". Al ejecutar un video cualquiera sobre el programa, se comprobó cómo al detectar el programa que el robot estaba dentro del área determinada como zona prohibida, el led se encendía y en caso contrario se mantenía apagada.

Realizadas todas estas pruebas, se dio paso a los resultados experimentales en tiempo real, los cuales se proceden a explicar a continuación.

## 5.2. Resultados experimentales en tiempo real

Como se ha visto en el apartado anterior, todos los resultados experimentales realizados sobre videos fueron un éxito, por lo que se dio el siguiente paso, probar todos los programas anteriores en tiempo real, captando los videos a través de una webcam.

A continuación, al igual que se ha hecho anteriormente, se van a adjuntar unas serie de imágenes mostrando el correcto funcionamiento del programa en tiempo real. Estas pruebas se han realizado también de manera progresiva, por si en caso de error, poder detectarlo con mayor facilidad.

### *Detección del movimiento*

En la imagen adjunta, se puede observar como el movimiento es detectado correctamente. En ella se puede ver con nitidez el área del robot, no habiendo ruidos ni interferencias.

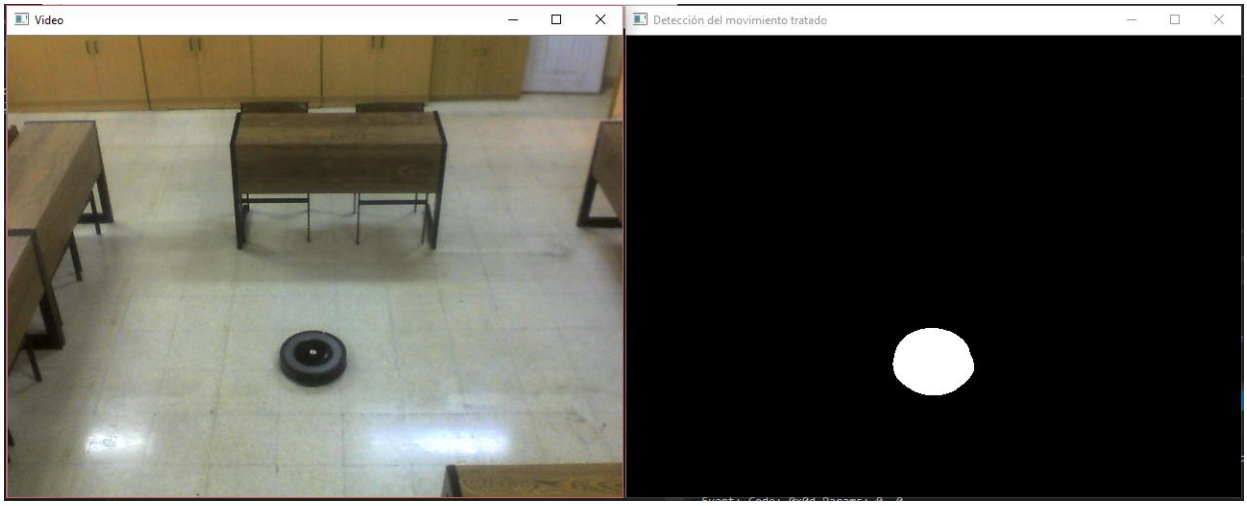


Figura 57. Detección del movimiento caso 1.

## *Tratamiento del objeto adecuado*

A continuación, se muestran una serie de imágenes en las que se muestra como de nuevo, el objeto es detectado correctamente, habiendo actualizado previamente la matriz de tamaño con el programa correspondiente.

Como se puede observar, el objeto es detectado en todo tipo de situaciones, con o sin obstáculos, fijos o móviles.

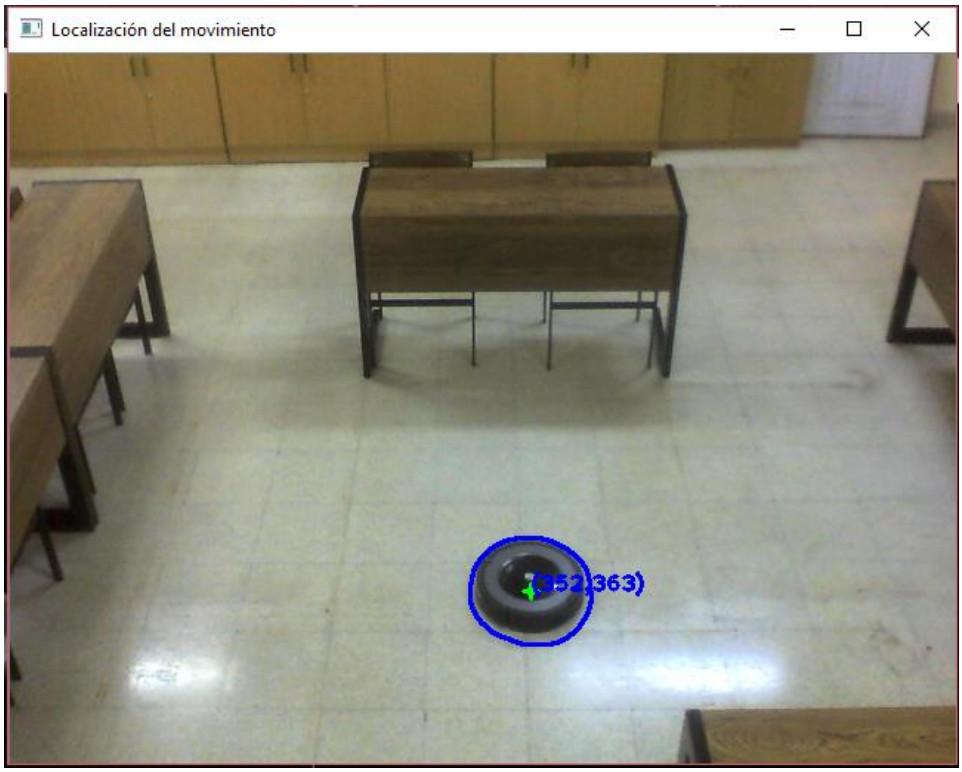


Figura 58. Tratamiento del objeto adecuado. Caso 1.

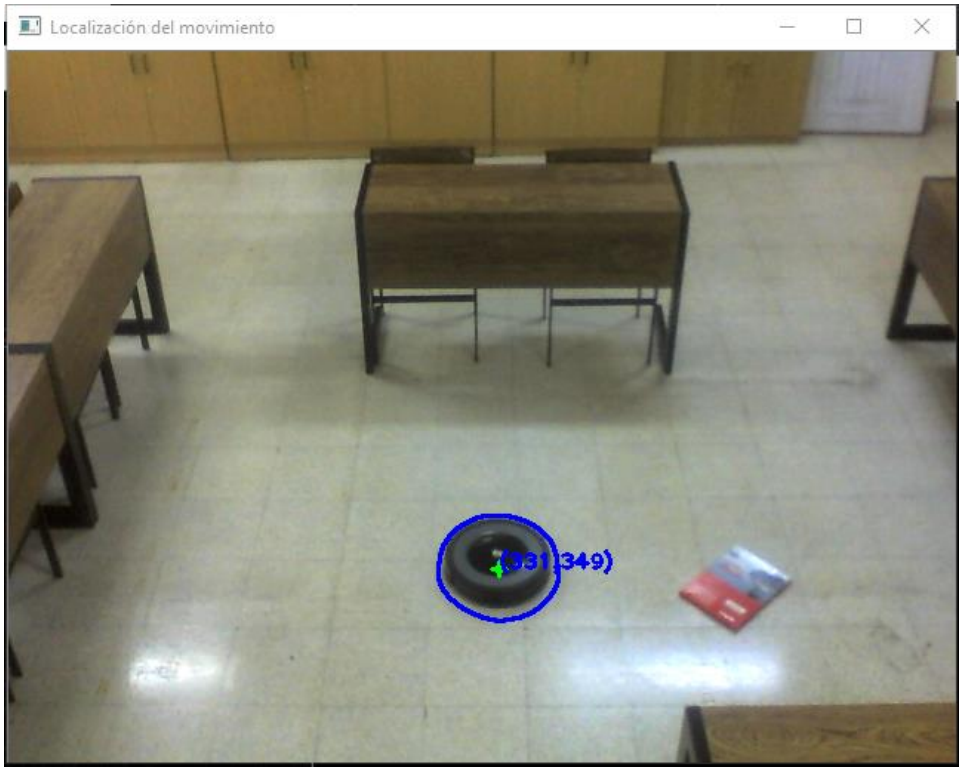


Figura 59. Tratamiento del objeto adecuado. Caso 2.

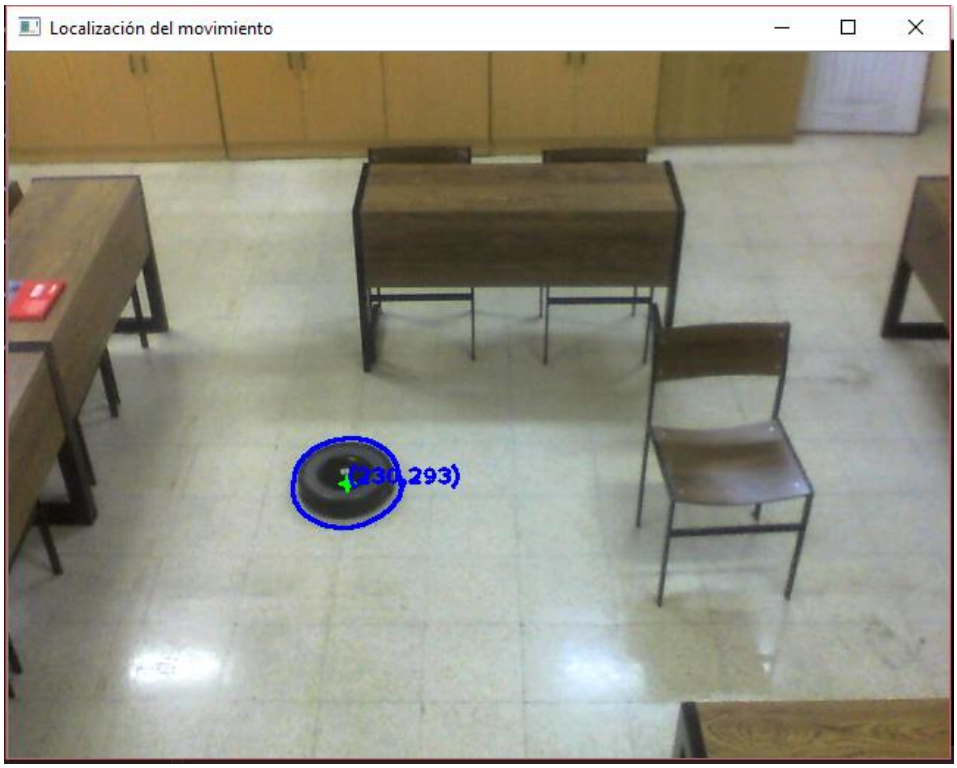


Figura 60. Tratamiento del objeto adecuado. Caso 3.

## *Detección de lo que ocurre en cada zona*

Tras comprobar que se detecta correctamente el robot de interés, se procede a comprobar si detecta cuando el mismo entra dentro de una zona prohibida en tiempo real.

Como se puede observar en las imágenes, el resultado fue positivo y al igual que en el tratamiento anterior, las pruebas fueron un éxito tanto con obstáculos fijos como en movimiento.

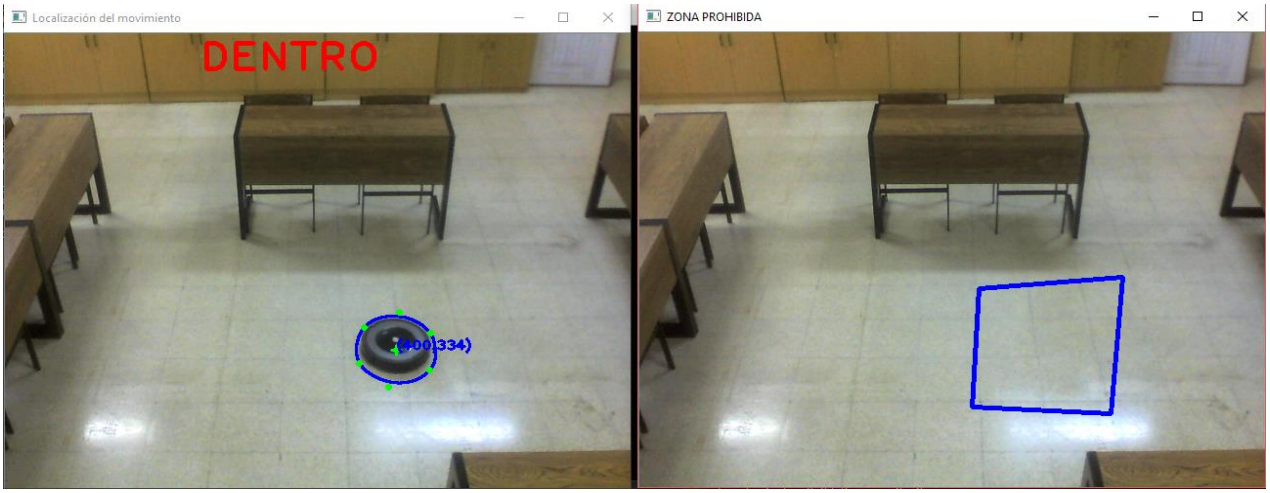


Figura 61. Detección de lo que ocurre en cada zona. Dentro.

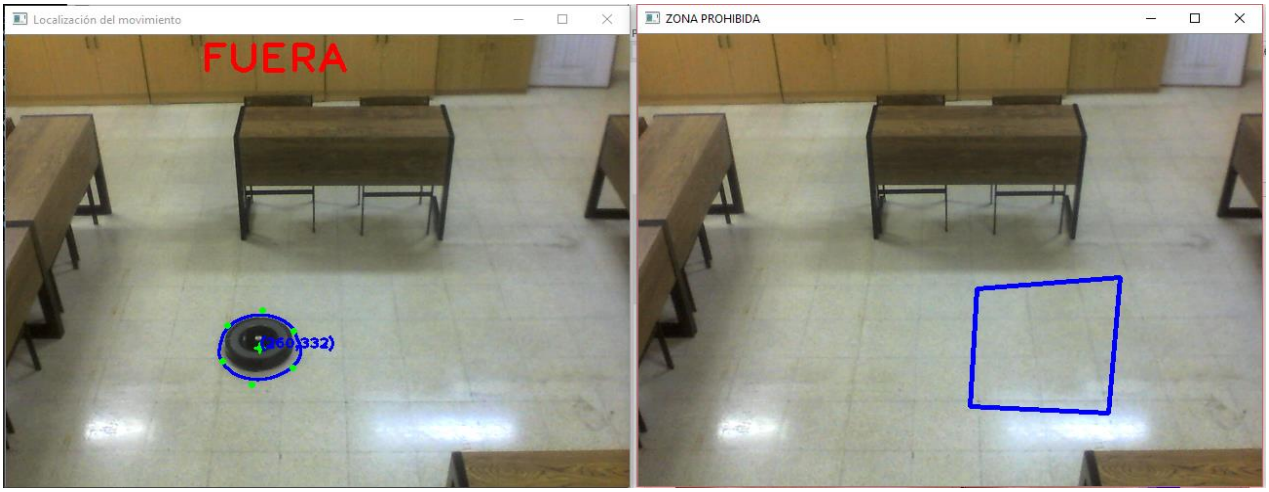


Figura 62. Detección de lo que ocurre en cada zona. Fuera.

## *Comportamiento del robot aspiradora ante la emisión de infrarrojos*

En posesión del robot, ya se podía probar el programa realizado en Arduino.

Como se ha dicho en el capítulo anterior, se han creado dos programas:

- Un programa que simula la pared virtual, haciendo que el robot gire cuando detecta la misma.
- Un programa que simula el mando compatible con el robot, pudiendo iniciar el modo *clean* y *spot* así como dirigirle hacía delante, derecha e izquierda.

Ambos programas funcionaron correctamente y permitieron el control del robot.

### *Comunicación Visual Studio – C++*

Comprobado el correcto funcionamiento del programa en Visión artificial y en Arduino por separado y en tiempo real, se procedió a la comunicación de ambos tal y como se hizo en la prueba de comunicación previa con un led.

Combinados ambos programas, se pudo comprobar el correcto funcionamiento de todo el proyecto, es decir, al detectar el programa de visión artificial que el robot se encontraba dentro de la zona prohibida, el Arduino ordenaba emitir las señales infrarrojas a través del led, y en caso contrario, no se emitía ningún tipo de señal. Al mandar las señales infrarrojas, el robot giraba tal y como se había conseguido anteriormente con el programa de Arduino por separado. Para poder saber cuándo se emitían o no las señales infrarrojas, se usó un led luminoso integrado en el Arduino, de forma que al emitir señales infrarrojas, el led se encendía y en caso contrario, el mismo se apagaba.

Por último, comentar que, al realizar esta última prueba, para asegurar que el robot no entrara dentro de la zona prohibida, se decidió mantener un cierto margen de seguridad, alejando los puntos de estudio del contorno del robot un poco hacía afuera.

# 6. ESTUDIO ECONÓMICO

## 6.1. Introducción

A lo largo de este capítulo se va a realizar el estudio del coste económico que supone la realización del proyecto desarrollado.

Hasta este momento, se ha estudiado todos los aspectos teóricos y técnicos necesarios para el control del robot aspiradora Roomba, sin embargo, no se ha analizado su viabilidad económica, aspecto también de gran importancia. Por este motivo, se ha escrito un capítulo sólo para este estudio.

En los siguientes apartados se analizarán por separado los costes directos e indirectos para finalmente sumarlos y obtener los costes totales del proyecto.

Puesto que se trata de un proyecto de investigación, indicar que se valorará por separado aquellos costes de investigación que se reparten entre todos los dispositivos y los costes fijos de cada uno.

## 6.2. Recursos empleados

A lo largo de la realización del proyecto se utilizaron una serie de recursos hardware y software. Cabe destacar que aunque algunos de estos recursos no son gratuitos, su uso no se limita al desarrollo del proyecto, es decir, dicho recursos han sido y seguirán utilizándose durante un periodo indeterminado de tiempo para otros ámbitos tanto académicos como personales, por tanto, es necesario tener presente la amortización del material durante el tiempo que ha sido utilizado en el proyecto, para sólo añadir dicho coste al total.

Los recursos software utilizados han sido:

- Sistema operativo: Windows 10.
- Software del control de bajo nivel: Arduino v1.6.4.
- Software del control de alto nivel: Visual Studio 2015 Community.

Los recursos hardware utilizados han sido:

- Ordenador portátil: Acer Aspire E15
- Material de laboratorio: polímetro.
- Dispositivos electrónicos: Arduino Uno, transistor 2N2222, módulo emisor receptor de infrarrojos, resistencias, WebCam, Roomba 866.

## 6.3. Costes directos

Los costes directos deben evaluarse en tres secciones diferentes, a saber:

- Coste personal.
- Costes amortizables de programas y equipos.
- Coste de materiales directos empleados.

Todos ellos se van a desarrollar a continuación.

### *Coste de personal*

Para la realización del presente proyecto ha sido necesario un ingeniero que llevara a cabo todo el proceso de diseño, montaje eléctrico, programación de los códigos necesarios, realización de pruebas y validación final del robot. Como se puede observar, en este apartado se está cuantificando el valor de la investigación llevada a cabo por el ingeniero, por lo que se trata de un coste que se reparte entre todos los dispositivos.

Para calcular el coste de este trabajo, en primer lugar se supondrá un coste anual para un ingeniero y una vez calculado, se adecuará este coste al total de horas trabajadas por el mismo.

Si se tiene en cuenta el sueldo anual bruto con posibles incentivos por su trabajo, y la cotización a la Seguridad Social, la cual es un 35% del sueldo bruto total, se tiene el siguiente coste total anual de un ingeniero:

| COSTE ANUAL DE UN INGENIERIO | |
|---|---|
| Sueldo anual bruto e incentivos | 30.000 € |
| Seguridad Social | 10.500 € |
| **Sueldo total** | **40.500 €** |

Tabla 4. Coste anual de un ingeniero.

Una vez calculado el coste de un ingeniero durante un año, es necesario conocer el número de horas trabajadas por el mismo durante el mismo.

Realizando una estimación de los días trabajados:

| DÍAS EFECTIVOS POR AÑO | |
|---|---|
| Año medio | 365,25 días |
| Sábados y Domingos | -104,35 días |
| Días de vacaciones efectivos | -20 días |
| Días festivos reconocidos | -15 días |
| Días perdidos estimados | -5 días |
| **Total días efectivos estimados** | **220,89 días** |

Tabla 5.Días efectivos por año.

Considerando que la jornada laboral es de 8 horas diarias, se puede saber cuántas horas trabaja un ingeniero medio al año:

$$220{,}89\,\frac{días}{año}\cdot 8\,\frac{horas}{día}=1.767{,}12\,\frac{horas}{año}$$

Sabiendo que en un año, un ingeniero medio gana 40500 €, se tiene que:

$$\frac{\frac{40.500\ €}{año}}{\frac{1.767{,}12\ horas}{año}}=\mathbf{22.91}\,\frac{€}{\boldsymbol{hora}}$$

A continuación, se adjunta una tabla orientativa del número de horas empleadas en los diferentes sectores que se requieren para el desarrollo del presente proyecto:

| DISTRIBUCIÓN TEMPORAL DE TRABAJO | |
|---|---|
| Formación y documentación | 100 horas |
| Desarrollo del software | 300 horas |
| Desarrollo del hardware | 20 horas |
| Elaboración de la documentación | 100 horas |
| **Tiempo total** | **520 horas** |

Tabla 6. Distribución temporal del trabajo.

Por tanto, el coste final de personal es:

$$22.91\,\frac{€}{hora}\cdot 520\ horas=\mathbf{11.913{,}2}€$$

| COSTE PERSONAL DIRECTO | 11.913,2 € |
|---|---|

### *Costes amortizables de programas y equipos*

Para poder calcular estos costes es necesario calcular los costes totales de los programas y equipos amortizables, para posteriormente calcular dicha amortización, es decir, el precio de cada equipo y programa en función del tiempo utilizado y su tiempo de vida útil. En el apartado *6.2 Recursos empleados* se desarrolló una lista detallada de todo el hardware y el software utilizado que es amortizable, sin especificar los precios de los mismos. Por este motivo, en el presente apartado se adjunta una tabla donde el lector puede consultar el precio de cada producto con su correspondiente amortización.

Se ha estimado que el tiempo de amortización considerada para todos los útiles, herramientas, equipos para tratamiento informático, sistemas y programas informáticos ha sido de 5 años, cuyo coeficiente lineal es del 40%.

| MATERIAL | IMPORTE | AMORTIZACIÓN |
|---|---|---|
| Sistema operativo Windows 10. | 95.95 € | 38,38 € |
| Software Arduino v1.6.4. | 0 € | 0 € |
| Visual Studio 2015 Community. | 0 € | 0 € |
| Microsoft Office Professional Plus 2013. | 119 € | 47,6 € |
| Ordenador portátil Acer Aspire E15 | 500 € | 200 € |
| Polímetro | 20 € | 8 € |
| **Total** | **734,95 €** | **293,98 €** |

Tabla 7.Costes amortizables.

Calculado el precio total de todos los materiales amortizables, es posible el cálculo de los mismos por hora mediante la siguiente expresión:

$$\frac{\textbf{Coste final}}{\textbf{hora}} = \frac{293{,}98\,\frac{€}{\text{año}}}{1.767{,}12\,\frac{horas}{año}} \simeq \mathbf{0{,}17}\,\frac{€}{\textbf{hora}}$$

Si por último, se multiplica este dato por el número de horas trabajadas, se obtendrá el coste total de los equipos y programas amortizables, el cual se va a considerar coste general y no fijo de cada dispositivo.

$$0{,}17\,\frac{€}{hora} \cdot 520\ horas = \mathbf{88{,}4}\ €$$

| COSTE DE AMORTIZACIÓN | 88,4 € |
|---|---|

### *Coste de materiales directos empleados*

En esta sección se van a tener en cuenta aquellos materiales que no son amortizables y que han sido utilizados para que el proyecto fuera posible. Estos son los únicos costes que se van a considerar fijos para todos los dispositivos.

| COSTES DE MATERIALES DIRECTOS | |
|---|---|
| Arduino Uno | 21,16 € |
| Módulo emisor-receptor | 2,85 € |
| Resistencias | 0,20 € |
| Transistor 2N222 | 1,10 € |
| Cableado | 0,71 €/m · 0,15 m = 0,1065 € |
| WebCam | 29,99 € |
| Roomba 866 | 679,99 € |
| **Coste total** | **735,4 €** |

Tabla 8. Costes de materiales directos.

| COSTE DE MATERIALES DIRECTOS | 735,4 € |
|---|---|

### *Coste derivado de otros materiales*

En este apartado se tendrán en cuenta todos los materiales que no se han detallado en apartados anteriores, pero que han sido también necesarios para la realización del proyecto. En ellos, se encuentran materiales como bolis, lápices, gomas, cuadernos, folios, cartuchos de tinta, pegamento, cinta adhesiva, entre otros.

Se ha estimado que el precio final de todos ellos ha sido de 10 €.

| COSTE DE CONSUMIBLES | 10 € |
|---|---|

### *Coste directos totales*

Realizado todo el desglose de costes directos, se puede calcular el coste directo total del proyecto, teniendo en cuenta la separación entre los costes fijos y generales.

| COSTES DIRECTOS TOTALES | |
|---|---|
| Coste personal directo | 11.913,2 € |
| Coste de amortización | 88,4 € |
| Coste de materiales directos | 735.4 € |
| Coste de consumibles | 10 € |
| **Costes directos variables** | **735.4 €** |
| **Costes directos fijos** | **12011,6 €** |

Tabla 9. Costes directos totales.

| COSTES DIRECTOS FIJOS | 735.4 € |
|---|---|
| COSTES DIRECTOS VARIABLES | 12011,6 € |

## 6.4. Costes indirectos

Estos costes hacen referencia a aquellos gastos producidos mientras se desarrolla el proyecto y que no pueden incluirse dentro de los gastos directos. En estos gastos se tiene en cuenta el consumo eléctrico (alimentación de dispositivos electrónicos, calefacción, luz…), el consumo telefónico (internet) y los servicios adicionales (transporte, servicios administrativos…).

| COSTES INDIRECTOS | |
|---|---|
| Consumo telefónico | 70 € |
| Consumo eléctrico | 180 € |
| Servicios auxiliares | 150 € |
| **Costes indirectos totales** | **400 €** |

Tabla 10. Costes indirectos.

| COSTES INDIRECTOS | 400 € |
|---|---|

## 6.5. Costes totales

Una vez calculados todos los costes, basta con sumarlos para obtener el coste total del presente proyecto. De nuevo, insistir en la separación de los costes relacionados con la investigación del proyecto y aquellos imputables directamente al dispositivo.

| COSTES TOTALES | |
|---|---|
| Costes directos Fijos | 735.4 € |
| Costes directos Variables | 12011,6 € |
| Costes indirectos | 400 € |
| **Costes totales investigación** | **12411,6 €** |
| **Costes totales imputables al dispositivo** | **735.4 €** |

Tabla 11. Costes totales.

Como ya se dijo al principio de este capítulo, los costes de investigación se pueden repartir entre todos los dispositivos, por lo que en función de la producción, los costes de los mismos variarán.

En el caso de producir 200 dispositivos, los costes de investigación se reparten de la siguiente forma:

$$\frac{12411{,}6\ €}{200\ dispostivos} = \mathbf{62{,}058}\ €$$

Por lo que sumando este coste a los costes fijos imputables a cada dispositivo, el coste total de cada uno de ellos es de:

| COSTES TOTALES DEL DISPOSITIVO | 797.45 € |
|---|---|

# 7. CONCLUSIONES

A lo largo del presente proyecto se ha ido consiguiendo, de manera satisfactoria, el desarrollo de los diferentes procesos de visión artificial, teniendo que haber realizado previamente un estudio en la materia de programación de C++, así como la utilización de las librerías OpenCV.

En primer lugar, se consiguió realizar la correcta detección de todo elemento en movimiento en un vídeo grabado a priori. Después, se consiguió detectar el elemento en movimiento buscado a través de una matriz de tamaño realizada con ayuda de un programa secundario de visión artificial con el fin de no considerar un animal o cualquier otro objeto en movimiento en la escena como el robot buscado.

Posteriormente se creó una interfaz interactiva con el usuario para:

- Elegir si se quiere seleccionar alguna zona del habitáculo como no apta para aspirar con el robot.
- Elegir con cuantos puntos se quiere delimitar esa zona.
- Indicar esos puntos sobre la imagen.

Diseñada la zona no permitida para el paso del robot, se consiguió desarrollar el programa de tal forma que se detectará cuando el robot entraba dentro de esta zona. Recordar que todos estos objetivos se consiguieron desde videos grabados previamente, procurando que los mismos tuvieran diferentes grados de complejidad, como obstáculos y otros elementos en movimiento, similares al robot.

Por último, antes de comenzar a probar el programa en tiempo real, se comprobó que la comunicación entre el programa anteriormente mencionado y el microcontrolador Arduino fuera correcta, viendo que al entrar el robot en una zona no permitida para su paso, un led conectado al microcontrolador se encendía.

Con todos estos pasos comprobados, se procedió a probar todos estos programas en tiempo real. Se vio como efectivamente, todos los pasos anteriormente mencionados funcionaban correctamente, a saber:

- Detección del movimiento.
- Detección del elemento buscado.
- Detección de que el robot se encuentra dentro de la zona permitida o de la zona prohibida.

Para finalizar el proyecto, se consiguió emitir correctamente la señal infrarroja de forma que el robot la recibiera, haciéndole alejarse de la zona prohibida en caso de entrar en ella.

Por otro lado, se quiere transmitir que el éxito del proyecto no depende únicamente de cumplir todos y cada uno de los objetivos marcados, también depende de la capacidad de subsanar los fallos que se han ido encontrando a lo largo de su desarrollo, sobre todo relacionados con la instalación de los programas y las librerías, habiendo sido necesario aplicar diferentes conocimientos aprendidos a lo largo de la carrera, consiguiendo afianzar y ampliar muchos de ellos al ponerlos en práctica, además de aprender muchos nuevos conceptos, como la programación en C++ y el uso de las librerías OpenCV.

Respecto a las dificultades que se han ido encontrando a lo largo del proyecto destacan tres puntos.

- Por un lado, se encuentra la forma en cómo encontrar el objeto adecuado. En primer lugar se trató de encontrar filtrando por color, puesto que el robot buscado es negro. Sin embargo, en función de la luz y de otros objetos de la habitación, en la imagen había más zonas en negro que la del robot, por lo que la opción se desechó. Tras este intento, surgió la idea de encontrar el robot por comparación de imágenes. Es decir, a partir de una foto del robot que se desea buscar, realizar un programa que encontrará por comparación de puntos clave la imagen de la foto en el video. Esta idea se consiguió desarrollar de forma exitosa encontrando el robot dentro de una captura del video, sin embargo, al tratarse de un programa muy pesado, al tratar de aplicar este programa sobre el video, el programa se ralentizaba tanto que se tuvo que desechar la opción. Finalmente se realizó un matriz de tamaño cuya aplicación ya se ha explicado anteriormente.
- Por otro lado, se encontraron bastantes problemas a la hora de instalar las librerías en los programas a usar tanto en Visual Studio como el Arduino. En cuanto a Visual Studio, no era suficiente con la librería por defecto de OpenCV, fue necesaria la instalación de librerías OpenCV adicionales. En cuanto a Arduino, para usar el sensor infrarrojo fue necesario el uso de una librería adicional que era incompatible con la que viene por defecto junto al programa.
- Por último, hubo problemas a la hora de emitir la señal infrarroja. En un primer momento se intentó mandar la señal con el led conectado directamente al Arduino, sin embargo, la intensidad que recibía el emisor era insuficiente, haciendo que la distancia alcanzada por la señal infrarroja fuese muy reducida. Por este motivo, se decidió incluir un transistor, con el fin de incrementar está intensidad y a su vez la distancia de emisión.

Hay que tener en cuenta que se trata de un proyecto diseñado para la investigación, el cual tiene inmensas posibilidades de ampliación y mejora.

Una posible ampliación podría ser la creación de un algoritmo mediante la emisión de señales infrarrojas que sustituya a los algoritmos que vienen de fábrica en los robots aspiradora, los cuales no terminan de funcionar correctamente. Dentro de estos algoritmos podrían incluirse las diferentes formas en las que puede recorrer la habitación para realizar la limpieza o ir hacia la base de carga ante un determinado nivel de carga, sin que este se quede en el camino, como pasa con muchos modelos.

También se podría ampliar el proyecto creando una aplicación para dispositivos móviles y tabletas, que sustituya el ordenador a la hora de ejecutar el programa, siendo posible la emisión de los infrarrojos desde el propio emisor del dispositivo.

# 8. BIBLIOGRAFÍA


1. Aspiradora Robot. *"¿Cuál es el mejor robot aspirador?"*. (2017). Recuperado de: https://www.aspiradorarobot.es/blog/mejor-robot-aspirador/
2. Aspirame. *Comparativa de los LG Hombot Square*. Recuperado de: https://robotsaspirador.es/comparativa-lg-hombot-square/
3. Aspirame. *Paredes virtuales de Roomba ¿Cómo funcionan?* Recuperado de: http://aspirame.es/consejos/paredes-virtuales/
4. Aspirame. *¿Qué robot aspirador comprar? La guía definitiva*. Recuperado de: https://robotsaspirador.es/que-robot-aspirador-comprar/
5. Bradley, Tony. *Pros and Cons Of Using A Robot Vacuum.* (2015). Recuperado de: https://www.forbes.com/sites/tonybradley/2015/12/22/pros-and-cons-of-using-a-robot-vacuum/#2523df2a4ac4
6. BricoyDeco, *Experiencia Personal con un Robot Aspirador*. (2014). Recuperado de: http://www.bricoydeco.com/robot-aspirador-ventajas-e-inconvenientes/#Desventajas_de_tener_un_robot_aspirador
7. Cock, P. *Roomba 620 infrared signals*. (2013). Recuperado de: http://astrobeano.blogspot.fr/2013/10/roomba-620-infrared-signals.html
8. Fernandez Garrido, I. *Robot aspirador: 6 ventajas y 4 desventajas*. (2017). Recuperado de: https://www.mediatrends.es/a/103656/robot-aspirador-ventajas-y-desventajas/
9. García Arenas, F. Luz Pueras, M. (1998) *Teorema de la Curva de Jordan.* Divulgaciones Matemáticas v.6, No. 1, 46-60.
10. Hsin Yi, Cohen. *The Truth About Robot Vacuum Cleaners.* (2017). Recuperado de: http://www.cleaningexpert.co.uk/truth-about-robot-vacuum-cleaners.html
11. iRobot*. Perfil de la compañía, Historia*. Recuperado de: https://www.irobot.es/explorar-irobot/acerca-de-iRobot/History
12. iRobot. *Vacuum Cleaning Robot, Roomba 866*. Recuperado de: http://www.witt-ltd.com/iRobot/Vacuum_Cleaning_Robots/Roomba_866
13. iRobot Corporation. (2013 - 2014). *iRobot Roomba, Manual de Usuario Serie 800.* Bedford.
14. Lady Ada. *IR Sensor.* (2017). Recuperado de: https://cdn-learn.adafruit.com/downloads/pdf/ir-sensor.pdf
15. Layton, Julia. *How Robotic Vacuums Work.* Recuperado de: http://electronics.howstuffworks.com/gadgets/home/robotic-vacuum2.htm
16. OpenCV. Recuperado de: http://opencv.org
17. OpenCV. OpenCV modules. Recuperado de: http://docs.opencv.org/3.1.0/index.html
18. Orbe, Antonio. *"Mi robot aspirador y yo"*. (2015). Recuperado de: http://red.computerworld.es/actualidad/mi-robot-aspirador-y-yo

19. Prestazion, *"Lo que debes saber antes de comprar un robot aspirador"*. (2015). Recuperado de: http://hogar.prestazion.com/lo-que-debes-saber-antes-de-comprar-un-robot-aspirador.html
20. Prinz, P. Kirck-Prinz, U. (2002). *A Complete Guide to Programming in C++.* Sudbury, Jones and Bartlett Publishers, Inc.
21. Visual Studio. (2017). Recuperado de: https://www.visualstudio.com/en-us/news/releasenotes/vs2017-relnotes
22. Robotsaldetalle. *Cosas que hacen los robots aspiradores (y no nos gustan)*. (2015). Recuperado de: http://robotsaldetalle.es/noticias/cosas-que-hacen-los-robots-aspiradores-y-no-nos-gustan/
23. Robotsaldetalle. *5 aspiradores robot con WiFi.* Recuperado de: http://robotsaldetalle.es/noticias/5-aspiradores-robot-con-wifi/
24. Sodnpoo. *Arduino powered robot vacuum virtual Wall.* (2014). Recuperado de: https://www.sodnpoo.com/posts.xml/arduino_powered_robot_vacuum_virtual_wall.xml
25. Universidad Carlos III de Madrid. *Curso: "Introducción a la visión artificial por computador: desarrollo de aplicaciones con OpenCv".* Plataforma edX.
26. University of Pittsburgh. *Bipolar Transistor Basics.* Recuperado de: http://www.pitt.edu/~qiw4/Academic/ME2082/Transistor%20Basics.pdf
27. Woodford, Chris. *Roomba robot vacuum cleaners.* (2016). Recuperado de: http://www.explainthatstuff.com/how-roomba-works.html
28. Zahumenszky, Carlos. *"He probado 4 robots aspiradora y ahora solo quiero barrer con escoba".* (2015). Recuperado de: http://es.gizmodo.com/he-probado-4-robots-aspiradora-y-ahora-solo-quiero-barr-1694536566

# 9. ANEXOS

A lo largo de la presente memoria se han ido indicando una serie de Anexo con el fin de no introducirlos el contenido de los mismos a lo largo de la redacción y así no perder el hilo de la explicación y no cargar de manera innecesaria algunos de los capítulos.

A modo resumen y de índice, los Anexos que se podrán ver a continuación son:

Anexo 1. DataSheet Arduino Uno.

Anexo 2. DataSheet Transistor 2N2222.

Anexo 3. Programa Arduino, Pared Virtual.

Anexo 4. Programa Arduino, mando.

Anexo 5. Programa de tratamiento de Visión Artificial.

Anexo 6. Programa matriz de tamaño.

Anexo 7. Programa fotografía.