ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE DISEÑO

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Trabajo Fin de Grado

SISTEMA DE DETECCIÓN DE OBSTÁCULOS PARA DRONES BASADO EN SENSOR LÁSER

Grado en Ingeniería Aeroespacial

Autor: Lucía Morcillo Martínez

Tutor: Rafael Masot Peris

Cotutor: Miguel Alcañiz Fillol

Junio de 2018

A mi familia, en especial a mis padres,
que siempre me han apoyado y animado
aunque no compartieran mis decisiones.

A mis amigos,
que se han convertido en mi familia durante estos cuatro años,
porque lo mejor que me ha dado esta carrera ha sido conocerlos.

A cada uno de mis profesores, desde el primero hasta el último,
en especial a mis tutores Rafael Masot y Miguel Alcañiz,
que me han dado algo muy valioso:
educación y conocimientos.

Resumen

Los vehículos aéreos no tripulados o UAV son una tecnología en continuo desarrollo y muy actual. Es de especial interés el estudio de la evitación de obstáculos en tiempo real ya que es un aspecto crucial cuando se trata de navegación autónoma. El diseño de este tipo de sistemas tiene que pasar primero por la detección de éstos; conocer el entorno en el que nos encontramos es algo de fundamental para poder llevar a cabo la misión con seguridad. Para ello ha surgido el desafío de diseñar nuevos sistemas de sensores electrónicos capaces de realizar mediciones fiables, precisas y eficientes computacionalmente.

En este Trabajo Fin de Grado se ha desarrollado un sistema de detección de obstáculos para drones basado en sensores láser, como su propio nombre indica, siendo éste un primer paso hacia un sistema más complejo integrable en un UAV real.

Se ha utilizado para el desarrollo de este sistema una plataforma de pruebas que consiste en la estructura de un cuadricóptero equipada con un servomotor en su parte superior que lleva acoplado dos sensores láser VL53L0X. Al moverse el motor los sensores van tomando medidas de la distancia a los posibles obstáculos en distintas posiciones, realizando un barrido en el plano horizontal de 360º alrededor del dron. Una vez conocida la distancia y el radial en el que se encuentran los obstáculos se conoce perfectamente la posición de éstos y se puede mostrar gráficamente, teniendo en cuenta que trabajaremos siempre en dos dimensiones.

Para la realización de este sistema se ha utilizado la placa Arduino UNO con su correspondiente software asociado y el programa Processing para el procesado y representación gráfica de los datos.

Tras todo el proceso realizado en este proyecto, desde el diseño inicial, pasando por la caracterización, hasta la validación, queda demostrada la viabilidad de la prueba de concepto desarrollada.

Abstract

The Unmanned Aerial Vehicles or UAVs are a technology in continuous development and which are a current issue. The study about the collision avoidance in real time is especially interesting since it is a key aspect when it deals with autonomous navigation. The design of this kind of systems has as its first stage the detection of the different obstacles.

Moreover, knowing the environment where we are flying is something fundamental in order to carry out the mission safely. For this purpose, it has emerged a new challenge for designing electronic sensors with the capacity of making reliable, precise and computationally efficient measurements.

In this Bachelor's Final Thesis, we have developed an obstacle detection system for drones based on laser sensors, as its name suggests, which is the first step towards a more complex system that could be integrated into a real UAV.

For the development of this system, we have used a test platform which consists of the structure of a quadricopter with a servomotor on it and with two laser sensors VL53L0X that spin with the engine. As the engine moves, the sensors take the distance measurements of the obstacles in the different positions, performing a sweep in the horizontal plane of 360º around the drone. Once we know all the distance and the radials where the obstacles are, we can accurately know the position of these and therefore the information can be displayed. Furthermore, we must keep in mind that in this project we always work in two dimensions.

To develop this system, we have employed the board Arduino UNO and its corresponding associated software. In addition, we have used Processing in the post-processing of the data and its graphic representation.

Finally**,** after all the work carried out in this project, from the initial design and characterization, to the validation, the viability of the conceptual test developed here can be demonstrated.

Resum

Els vehicles aeris no tripulats o UAV són una tecnologia en desenvolupament continu i molt actual. És de especial interès l'estudi de l'evitació d'obstacles en temps real ja que és un aspecte necessari quan es tracta de navegació autònoma. El disseny de este tipus de sistemes ha de passar primer per la detecció d'estos; conèixer l'entorn en el que ens trobem és una cosa fonamental per a poder dur a terme la missió amb seguretat. Per aconseguir-ho, ha sorgit desafiament de dissenyar nous sistemes de sensors electrònics capaços de realitzar mesuraments fiables, precisos i eficients computacionalment.

En aquest Treball de Fi de Grau s'ha desenvolupat un sistema de detecció d'obstacle per a drons basat en sensor làser, com el seu nom indica, sent este un primer pas cap a un sistema més complex integrable en un UAV real.

S'ha utilitzat per al desenvolupament d'aquest sistema una plataforma de proves que consisteix en l'estructura d'un quadricòpter equipat amb un servomotor en la seua part superior que porta integrat dos sensors làser VL53L0X. Al moure's el motor, els sensors van prenent mesures de la distancia als possibles obstacles en distintes posicions, analitzant tot el pla horitzontal amb una visió de 360º al voltant del dron. Una vegada coneguda la distància i el radial en que es troben els obstacles, es coneix perfectament la posició d'aquests i es pot mostrar gràficament, tenint en compte que treballarem sempre en dos dimensions.

Per a la realització d'aquest sistema, s'ha utilitzat la placa Arduino UN amb el seu corresponent software associat i el programa Processing per al postprocessat i representació gràfica de les dades.

Amb tot el procés realitzat en este projecte, des del disseny inicial, passant per la caracterització, fins a la validació, queda demostrada la viabilitat de la prova de concepte desenvolupada.

Índice general

Índice de figuras

Índice de tablas

1. Introducción

El presente Trabajo Fin de Grado tiene como objetivo el desarrollo de un sistema de detección de obstáculos para drones basado en sensores laser. Dichos sensores son capaces de medir de forma precisa la distancia entre el dron y los objetos que se encuentren en su entorno. Además, se utilizará un servomotor que hará girar estos sensores para poder detectar dichos obstáculos no en una sola dirección sino en los 360º alrededor del dron.

El trabajo se estructura de la siguiente forma. En el *Capítulo 1*, en el que nos encontramos, se hace una introducción a este trabajo explicando por qué se ha elegido la temática, qué pretendemos conseguir con él y en qué dirección están avanzando las investigaciones sobre esta temática en la actualidad. También se da una breve explicación sobre el marco teórico que hay detrás de los dispositivos de medición de distancia.

En el *Capítulo 2* se explica de forma detallada el diseño y la implementación del sistema, tanto de forma física como la programación de la parte electrónica. Así pues, se va a exponer de forma detallada cómo se realiza el montaje del prototipo y el hardware y el software empleados en su desarrollo.

El *Capítulo 3* muestra cómo se ha realizado la calibración tanto de sus elementos de forma individual como integrados ya todos en el sistema final. Se explica tanto el montaje de los bancos de pruebas como los programas realizados para su implementación. Mediante los estudios estadísticos de los datos obtenidos se ha podido ir mejorando el sistema hasta obtener la mejor versión posible de éste según nuestras limitaciones y necesidades.

En el *Capítulo 4 se* hace una pequeña validación del sistema comprobando su funcionamiento en un entorno de trabajo más realista. En este caso se han empleado obstáculos con diferentes formas, tamaños y colores (y por tanto diferentes reflectancias) para poder confirmar el correcto funcionamiento del sistema independientemente de estas variables.

Finalmente, en el *Capítulo 5*, se discuten los resultados obtenidos al aplicar el sistema final en el entorno real, valorando si se cumple o no los objetivos de este Trabajo Fin de Grado y las posibles mejoras en futuras investigaciones. Se incluye, como Capítulo 6, el presupuesto aproximado para el desarrollo del sistema. Y, como en cualquier proyecto, podemos consultar la bibliografía empleada en el *Capítulo 7*.

Además, esta memoria consta de un *Anexo I* final en el que se puede consultar cómo se ha diseñado la pieza necesaria para la unión de los sensores y el motor. Dicha pieza fue impresa mediante una impresora 3D una vez finalizado su diseño.

1.1. Justificación

La aparición en el mercado de los drones o Vehículos Aéreos no Tripulados, comúnmente denominados por sus siglas en inglés UAV (Unmanned Aerial Vehicle), ha tenido un gran éxito en muchos campos de interés. En el ámbito militar, se ha demostrado que los UAV son una plataforma móvil efectiva en futuros escenarios de combate. En aplicaciones civiles, numerosas plataformas de UAV han proliferado y se han aplicado a la vigilancia, el monitoreo de desastres y el rescate, la entrega de paquetes y la fotografía aérea. Varias compañías están desarrollando sus propios sistemas UAV, como Amazon Prime Air, Google's Project Wing y DHL's Parcelcopter. Para aumentar la seguridad de vuelo, el UAV debe ser capaz de detectar adecuadamente y evitar otras aeronaves o intrusos durante su vuelo.

Como consecuencia ha habido un gran auge en las tecnologías asociadas a este sector por lo que se ha iniciado un desafío en el diseño de nuevos sensores electrónicos capaces de ayudar al piloto en el aire o incluso de permitir en vuelo autónomo sin necesidad de que haya una persona expresamente controlando el dron. Un problema de diseño recurrente en este ámbito consiste en equilibrar la complejidad del subsistema de detección de obstáculos con algoritmos avanzados de procesamiento de datos que nos permita tomar decisiones y cambiar la dirección de movimiento para lograr evitar dichos obstáculos. Además, hay que tener en cuenta que todo este procesamiento debe llevarse a cabo en tiempo real y en el menor tiempo posible, ya que el dron se encuentra en movimiento y en un entorno cambiante por lo que necesita conocer los posibles obstáculos de forma prácticamente instantánea.

En la pasada década, el interés por los UAVs y su autonomía ha sufrido un constante desarrollo. La evasión de colisiones es un requerimiento muy importante en vuelos autónomos. Aunque existen múltiples soluciones para la detección de obstáculos y su posterior evitación cada una tiene sus inconvenientes por lo que aún no se ha encontrado la solución definitiva. Las distintas soluciones que se han ido planteando para detectar los diferentes obstáculos del entorno son:

- Fusión de sensores ultrasónico e infrarrojo.
- LIDAR (Laser Imaging Detection and Ranging).
- Reconocimiento de imágenes con cámara.
- Sensor Láser.

En este proyecto hemos decidido emplear sensores láser de bajo coste, más concretamente el módulo VL53L0X, para poder detectar los obstáculos que es el primer paso a la hora de elaborar un sistema de evitación de colisiones. Se ha tomado esta decisión ya que otras opciones quedaban fuera de presupuesto, como ocurría con el LIDAR, o llevaban consigo un postprocesado de muy alto nivel [1], como pasa con el reconocimiento de imágenes. Por otro lado la fusión sensorial sí que nos la planteamos ya que es asequible y se puede llevar a cabo con sensores de bajo coste [2] y ya hay una amplia investigación sobre ellos que se ha llevado a cabo en estos últimos años [3].

Aunque esta última no nos parecía una mala opción, requería de un mayor número de sensores lo que aumentaba tanto el presupuesto como la complejidad del sistema, además es algo muy investigado y desarrollado en la actualidad. Por todo esto se prefirió apostar por una tecnología menos usada en este campo como son los sensores láser para lograr un proyecto más innovador y poder sacar conclusiones sobre el uso de esta tecnología en los sistemas de evasión de obstáculos.

Las últimas investigaciones y desarrollos se centran en la capacidad de formación de enjambre de los vehículos aéreos no tripulados donde varios vehículos pueden cooperar y trabajar juntos de forma autónoma. Para operar de manera segura y cumplir tareas de la misión, uno de los criterios importantes que se requieren para los UAV es la capacidad de evitar colisiones con otros miembros de enjambres y obstáculos ambientales.

Este tipo de vuelo en formación en UAV ya se está empezando a usar en la actualidad, de momento de forma experimental, y es uno de los principales focos de investigación hoy en día. El gobierno de Estados Unidos está apostando por el desarrollo de esta tecnología sobre todo con fines militares. Ya se hicieron las primeras pruebas en enero de 2017 en California con el lanzamiento de 103 mini-drones en formación de enjambre desde tres cazas. [4]

Cuando trabajamos con enjambres de drones hay que tener en cuenta que no sólo deben evitarse entre sí, sino que además deben ser capaces de evitar obstáculos externos al sistema. Todo ello debe llevarse a cabo sin separarse ni romper la formación y con un movimiento coordinado que no haga que al evitar un obstáculo externo los UAVs choquen entre ellos. Todos estos concepto se han investigado ya en múltiples estudios [3] [5] [6] y conllevan algoritmos de control muy elaborados en los que no vamos a entrar ya que quedan fuera de los objetivos de este trabajo.

El sistema de detección de obstáculos desarrollado en este Trabajo Fin de Grado está dirigido, además de para aeronaves no tripuladas que vuelan de forma independiente, a lograr un vuelo de proximidad más seguro en aplicaciones de enjambrado UAV sin colisiones entre los miembros del grupo. Esta es una primera prueba de concepto de lo que podría ser un sistema de detección y evasión de obstáculos definitivo y comerciable.

Se ha decidido trabajar en esta dirección ya que pensamos que es una línea de investigación que está muy de actualidad y que realmente puede interesar a mucha gente. Podríamos encontrar posibles clientes interesados en el producto tanto en el ámbito militar como en el civil, como se ha comentado anteriormente. Además, todos los sistemas relacionados con los UAVs están evolucionando en muy poco tiempo de forma exponencial ya que es una tecnología muy nueva. Esto es algo que algún día llegará a su máximo y dejará de existir un nicho de mercado y desarrollo como el actual por lo que se debe que aprovechar este momento ya que es idóneo para la investigación en este campo.

1.2. Objetivos

En este Trabajo Fin de Grado se tiene como objetivo principal el diseño y desarrollo del sistema electrónico necesario para detectar los obstáculos que se puedan encontrar en el entorno de un dron basado en sensores láser.

Es, por lo tanto, la finalidad de este trabajo proporcionar mediciones de distancia y posición de los distintos objetos que se encuentren alrededor del dron. Estas mediciones deben llevarse a cabo de forma eficiente y precisa a través de los sensores láser VL53L0X ya que deben hacerse en el menor tiempo posible y un error puede desencadenar el fallo de la misión. De esta forma se puede conocer el entorno en el que está volando el UAV con la finalidad de poder evitar todos los posibles obstáculos.

Se desea conseguir al finalizar este proyecto que, tras medir la distancia a los objetos del entorno, determinando su posición (según el radial en el que se encuentra y la distancia al dron), se postprocese la señal para intentar minimizar el error y se muestre por pantalla para poder tener una idea de la disposición de los obstáculos de la forma más clara posible. Queda fuera de los objetivos de este proyecto, por lo tanto, el desarrollo de un algoritmo que una vez finalizado el procesado de datos de las mediciones redirija el dron hasta ambientes seguros y libres de obstáculos.

Esta es una primera prueba de concepto con la que se pretende llegar al máximo desarrollo posible de este sistema de detección de obstáculos, dentro de nuestras limitaciones. En proyectos futuros este sistema puede ser perfeccionado con elementos de mayor calidad y precisión, si se cuenta con un presupuesto mayor, y con algoritmos de procesado de datos más complejos, siempre teniendo en cuenta que es una aplicación en tiempo real lo que supone muchas restricciones.

Así durante el desarrollo de este trabajo se han ido marcando diferentes objetivos conforme se avanzaba en el diseño:

- Analizar los diferentes dispositivos electrónicos utilizados en la actualidad para medir distancias y realizar un estudio comparativo entre ellos, mostrando ventajas e inconvenientes, con la finalidad de encontrar el óptimo para nuestras necesidades.
- Investigar acerca de la nueva tecnología basada en sensores de bajo coste y analizar su utilización en diferentes aplicaciones aeronáuticas.
- Analizar de forma detallada las posibilidades de hardware y software disponibles para poder elegir el más adecuado y, una vez elegido, estudiarlo en profundidad para poder sacarle el máximo rendimiento.
- Caracterizar correctamente los sensores láser empleados, tanto individualmente como trabajando conjuntamente en el sistema final, para conseguir la máxima fiabilidad en la medida.
- Proponer diferentes líneas de desarrollo futuras para conseguir un sistema de detección de obstáculos que sea asequible pero seguro.

1.3. Marco teórico

Como se ha mencionado en los apartados anteriores, el objetivo perseguido en este Trabajo Fin de Grado es desarrollar un sistema capaz de detectar la ubicación de los obstáculos que se encuentran alrededor de un dron mediante la medición de distancias empleando sensores láser. Se trata de un nuevo planteamiento de la problemática actual de diseñar sensores fiables, precisas y económicos capaces de proporcionar medidas rápidas de la distancia a tiempo real.

Para poder entender por qué se ha elegido el uso de sensores laser en este trabajo primero vamos a exponer las teorías y modelos fundamentales de medición de distancia en vehículos aéreos.

Entre los dispositivos más empleados para la detección y evasión de obstáculos se encuentran, tal y como se ha mencionado en el apartado 1.1., el sensor ultrasónico y el infrarrojo que suelen trabajar conjuntamente, el LIDAR, la visión directa con cámara y el sensor laser que hemos utilizado en este trabajo.

SENSOR ULTRASÓNICO

El sensor de ultrasonidos es un dispositivo capaz de medir la distancia a un objeto mediante el uso de ondas sonoras y la evaluación del eco recibido. Básicamente, está constituido por un emisor que genera una onda sonora a una frecuencia específica, y un receptor que percibe el eco producido al reflejar esa misma onda contra el objeto cuya distancia se desea medir. Al evaluar el tiempo transcurrido entre la generación de la onda y la recepción de la misma, y conociendo la velocidad del sonido, es posible calcular la distancia entre el sensor y el objeto con unas sencillas fórmulas:

$$Distancia = tiempo * velocidad \rightarrow L = \frac{T}{2} * a$$

La principal ventaja de este tipo de dispositivos es su capacidad para funcionar correctamente en condiciones adversas, ya que la detección del blanco no depende ni de su color, ni de la luz solar, ni del polvo ambiental. Estos son sensores de alta frecuencia y potencia de penetración que permite detectar con precisión los obstáculos que se encuentren paralelos a su superficie. Además, son sencillos de utilizar y no suponen ningún peligro para personas, equipos u objetos cercanos. Podemos encontrar sensores de este estilo por una amplia variedad de precios en el mercado, que va desde los 5 € hasta más de 300 €, pero por un precio muy asequible se puede conseguir un sensor ultrasónico de buenas características.

Por otro lado, este tipo de sensores tienen ciertos inconvenientes que se deben tener en cuenta a la hora de su utilización. Para empezar, este tipo de sensores emiten las ondas con un haz cónico por lo que existe un cambio de sensibilidad en función del ángulo del haz, ya que, en función de la distancia el área que cubre la onda es mayor. Puesto que al alejarnos esta área aumenta no podemos conocer exactamente la posición del obstáculo y, si hay más de un objeto dentro de la superficie cubierta por el haz solo detectaremos el más cercano.

Otra desventaja que se debe considerar son los errores en la lectura causados por la inclinación de un objeto, ver Figura 1, ya que puede ocasionar que el reflejo emitido no llegue hasta el receptor por lo que este blanco seria invisible a ojos del sensor ultrasónico. En nuestro caso esto es difícil de solucionar ya que los obstáculos pueden encontrarse en cualquier sitio y con cualquier orientación.

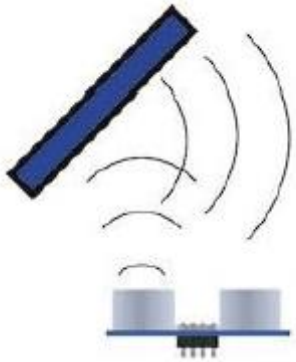


Figura 1: Error de medición del sensor ultrasónico.

Por último, existe otro tipo de inconveniente a estudiar y es el hecho de que existen materiales que absorben el sonido como puede ser el cuerpo o cortinas, por este motivo los sensores ultrasónicos no son capaces de detectar gente o dar una medida correcta hasta ellos. Esto se solucionará utilizando conjuntamente este tipo de sensores con los infrarrojos ya que estos sí que son capaces de detectar este tipo de obstáculos, la utilización conjunta de éstos se explicará más adelante.

Al igual que ocurre con otro tipo de sensores, éstos tienen límites tanto de distancia máximo como mínima y de tamaño del blanco. Puede ocurrir que el blanco sea tan pequeño que no se detecte el eco reflejado.

En conclusión, los sensores ultrasónicos son fáciles de utilizar y se pueden encontrar a precios asequibles, con una relación calidad precio razonable. Sin embargo, cuando se trabaja con aplicaciones críticas los distintos errores de estos hacen que no sean una buena opción para emplear por sí solos, pero sus características mejoran al emplearlos junto con sensores infrarrojos filtrando los resultados de ambos.

SENSOR INFRARROJO

El sensor infrarrojo es un sensor de medición de distancia, que se basa en un sistema de emisión/recepción de radiación lumínica en el espectro de los infrarrojos (cuya longitud de onda se encuentra entre las ondas de radio y el espectro visible). Es un sensor muy utilizado para la medición de distancias pequeñas, en un rango entre 50 y 80 cm, y la detección de obstáculos debido, entre otras cosas, a sus reducidos precios. Además, se debe tener en cuenta que el tipo de detección que realizan es direccional, es decir, sólo son capaces de detectar objetos que están enfrente del sensor.

Este tipo de sensores está formado por un emisor LED de luz infrarroja y un receptor fotodiodo que detecta el reflejo de dicha luz en el blanco. Normalmente este tipo de detectores vienen acompañados de un condensador para reducir el ruido. Además, este

tipo de dispositivos proporcionan una señal analógica en su salida en función de la posición en la que el rayo de luz impacta, se proporciona una placa de medición estándar que permite obtener la lectura como un valor digital.

Una de las técnicas más habituales para la medición de la distancia es mediante la triangulación del haz de luz colimada, en este caso se mide el ángulo con el que llega el reflejo ya que dicho ángulo es diferente en función de la distancia al objeto. Si bien también se puede "estimar" la distancia de un objeto a partir de la cantidad de energía recibida tras rebotar la luz sobre un objeto.

Uno de los inconvenientes de emplear este tipo de técnica de medición es que el ángulo de incidencia del eco detectado varía muy poco a grandes distancias por lo que es muy poco sensible a obstáculos lejanos. Por otro lado, este tipo de sensor presenta el inconveniente de ser sensible a la luz ambiente como consecuencia de que los rayos de sol también emiten en el espectro de luz infrarroja. Por este motivo, son sensores que se utilizan habitualmente en entornos con iluminación artificial de forma predominante (interiores). Además, la cantidad de luz infrarroja reflejada es muy dependiente de las características del objeto, como su color, material y forma. Así, el receptor no es capaz de detectar correctamente el eco cuando la superficie es oscura, ya que debida a su baja reflectancia absorbe gran parte de la luz, por lo que no disponen de una precisión suficiente para proporcionar una estimación de distancia adecuada. Por último, como cualquier sensor óptico, este tipo de dispositivos fallan ante condiciones de iluminación pobres como con la presencia de humo o niebla.

Como se puede observar todos los inconvenientes presentados por este sensor no afectan a la medición mediante sensores ultrasónicos, quitando el hecho de que siempre cuanto mayor sea la distancia a medir, menor será la precisión de la medida. Al igual que estos sensores pueden funcionar perfectamente con materiales que absorben sonido, como las ropas, por lo que complementan perfectamente a los sensores ultrasónicos. Por ello ambos tipos de dispositivos suelen trabajar conjuntamente para que al procesar las señales recibidas de distancias por ambos sensores se pueda detectar el mayor número de obstáculos posible minimizando el error y compensando las carencias que presentan individualmente. Este procesado de señal se suele hacer mediante un filtro Kalman Extendido.

En definitiva, este tipo de sensores son baratos y sencillos de utilizar. Aunque individualmente no son una buena opción ya que dependen del color, material, forma y posición del objeto, trabajando junto con los sensores ultrasónicos son una tecnología buena y asequible que ya está siendo muy empleada en la actualidad en campos como el tratado en este trabajo.

LIDAR

El sistema LIDAR (Laser Imaging Detection and Ranging) es un dispositivo que permite determinar la distancia a un objeto o superficie utilizando un haz de láser pulsado. La distancia se determina midiendo el intervalo de tiempo entre la salida del impulso y su retorno de nuevo al sensor LIDAR.

Su principio de funcionamiento es muy similar a los anteriores sistemas de medición de distancia expuestos. El LIDAR envía un pequeño haz láser sobre un objeto y mide el tiempo que tarda la señal reflejada en volver hasta el receptor. El equipo requerido para medir este tiempo necesita funcionar con una gran rapidez que sólo ha sido posible gracias a los avances en la tecnología informática. El cálculo básico consiste en medir la distancia que ha viajado un fotón de luz, hecho que resulta muy sencillo siendo conocida y constante la velocidad de la luz.

$$Distancia = tiempo * velocidad \rightarrow L = \frac{T}{2} * c$$

Lo que diferencia este tipo de dispositivos de un sensor láser convencional es que el sistema LIDAR envía un rayo láser directamente hacia un espejo que oscila a una frecuencia muy elevada en un solo eje (véase Figura 2). A continuación, dicho espejo redirige este haz hacia una lente difusora que duplica el ángulo de orientación del propio haz en el otro eje (x, y). Por lo que la principal ventaja de este tipo de sistemas con respecto a un sensor láser convencional es que no mide distancia en una sola dirección, sino que puede hacer un mapeo en 3D, lo que para nuestro objetivo perseguido sería una solución ideal.

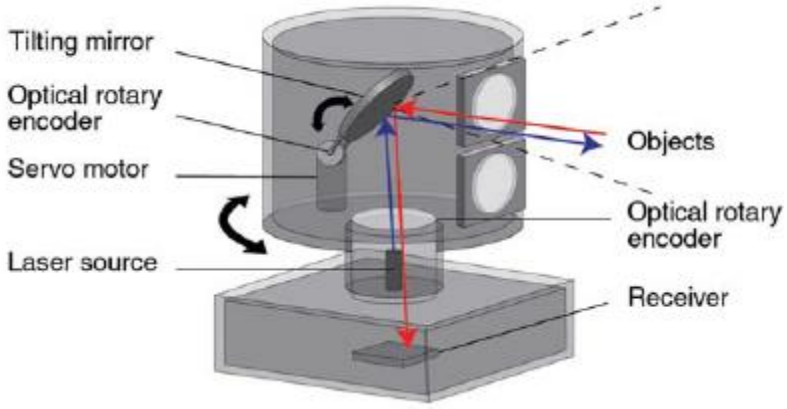


Figura 2: Esquema de funcionamiento de un sistema LIDAR.

Cuando dicho haz alcanza a un objeto, su señal es reflejada y capturada por la lente receptora, que la redirige a su vez a una matriz de fotodetectores. El diodo láser sincroniza su pulsación con la del reloj para conseguir un escaneo formado por múltiples líneas de visión.

Por último, se digitaliza y procesa la señal reflejada. El array de fotodetectores segmenta cada señal en múltiples mediciones individuales para construir una matriz 3D formada, en la mayoría de las ocasiones, por más de 20000 puntos por segundo. En la industria aeronáutica este tipo de dispositivos son combinados con otros sistemas como el GPS y

la IMU. Pese a su gran potencial, el LIDAR no es una herramienta que funcione en soledad, sino que necesita otros dispositivos para obtener medidas precisas y fiables. Tras todo el procedimiento de medición y procesado de señal se consigue una recreación en tres dimensiones del entorno.

Actualmente podemos encontrar en el mercado una amplia variedad de sistemas LIDAR, todos con unos precios muy elevados. En comparación con las primeras versiones de LIDAR, que comenzaron teniendo un precio de alrededor de los cien mil euros por unidad, el precio ha caído de forma muy acusada. Aun así, se trata de un sistema caro cuyo precio mínimo en el mercado está en torno a los 200€. Esto se sale del presupuesto con el que contábamos, lo que es el motivo por el que descartamos este sistema, aunque sus características eran muy llamativas para nuestro proyecto. Además, suelen ser sistemas muy complejos mecánicamente, con el sobrepeso que eso conlleva, lo que para un sistema embarcado es una variable fundamental.

En conclusión, el LIDAR es una herramienta capaz de generar un mapa virtual del entorno por lo que podría detectar todos los obstáculos cercanos sin problema alguno. Además, es capaz de medir y procesar gran cantidad de información en apenas unos segundos y no se ve afectada por el color, tamaño o forma del objetivo. Si se dispusiera de un presupuesto mayor ésta sería la opción óptima y se podría estudiar en futuras ampliaciones si se nos presenta la oportunidad de contar con uno de estos sistemas.

RECONOCIMIENTO DE IMÁGENES CON CÁMARA

Este tipo de sistemas de medida de distancia son menos empleados para la detección de obstáculos, en el ámbito aeronáutico, ya que conllevan un procesado de señal muy complejo lo que no es adecuado para sistemas embarcados de tiempo real.

En contraposición, el sistema de detección y evasión de obstáculos basado en la visión es cada vez más atractivo ya que las cámaras son livianas y de bajo costo, y lo más importante, pueden proporcionar información más rica que otros sensores. El estudio de la problemática asociada a la evasión de obstáculos en aeronaves no tripuladas, como tema central en la visión por computadora, ha estado activo durante las últimas décadas y ha logrado un gran progreso. Aun así, esta todavía es una tecnología en desarrollo y demasiado compleja computacionalmente hablando.

La detección de obstáculos mediante visión artificial se lleva a cabo buscando objetos salientes en la imagen, es decir, se busca algo diferente en un fondo más o menos monótono. La detección de objetos salientes en la visión artificial se interpreta como un proceso de cálculo de un mapa de prominencia en una escena donde resalta las regiones visuales distintas y suprime el fondo. La mayoría de los métodos de detección de objetos más importantes se basan en la suposición sobre las propiedades de los objetos y el fondo. La suposición más utilizada es el contraste anterior que supone que los contrastes de apariencia entre los objetos y los fondos son muy altos. Muchas investigaciones actuales están apostando por conocer cómo es el fondo a priori, esto se hace asumiendo que normalmente el fondo coincide con lo que aparece en el límite de

la imagen. Sin embargo, esos métodos carecen de la capacidad de utilizar la información contextual entre fotogramas consecutivos en la secuencia de imágenes.

Por otro lado, este tipo de sistemas no sólo se marcan como objetivo el detectar un objeto, sino que además se proponen seguir su movimiento. Para realizar esta tarea normalmente se emplea un filtro Kalman para predecir la futura ubicación del objeto, y con la siguiente medida del detector de obstáculos se refina la ubicación del objeto calculada por el filtro.

En conclusión, este es un sistema de detección y seguimiento de obstáculos que puede ser muy bueno si se consigue optimizar el procesado de la señal para poderlo emplear en sistemas de tiempo real. EL hardware es muy económico ya que sólo es necesario una cámara convencional, pero el software es muy complicado y costoso. La complejidad del procesado nos hace descartar esta opción para el presente Trabajo Fin de Grado.

SENSOR LÁSER

Un láser (del acrónimo inglés LASER, light amplification by stimulated emission of radiation; amplificación de luz por emisión estimulada de radiación) es un dispositivo que utiliza un efecto de la mecánica cuántica, la emisión inducida o estimulada, para generar un haz de luz coherente (ondas luminosas con fase coherente y que por tanto conservan una relación de fase constante) tanto espacial (mantiene un tamaño pequeño al transmitirse) como temporal (concentra la emisión en un rango espectral muy estrecho). Este efecto supone una estimulación eléctrica o térmica de los átomos, moléculas o iones que conforman un material. Una de las características principales de un láser es su direccionalidad, es decir, es un rayo recto y concentrado.

Existen varios dispositivos empleados para medir distancias basados en tecnología láser, estos son, el telémetro láser, el distanciómetro láser y los sensores láser. La diferencia entre ellos es el rango de distancias en el que pueden trabajar, por los principios de funcionamiento empleados y sus aplicaciones. Habitualmente, los telémetros se utilizan para largas distancias (hasta 1 km) donde no se busca la precisión, los distanciómetros se emplean para distancias de hasta 200 m con una elevada precisión, y los sensores para distancias cortas donde la precisión milimétrica adquiere importancia. Esta última opción es la que nos interesa en nuestro trabajo.

El método utilizado depende de la precisión y la distancia máxima requerida por el sistema. Entre los principales métodos se encuentran la triangulación y el método basado en el principio del Time-Of-Flight (ToF). El método de triangulación es muy similar al explicado para los sensores infrarrojos, la luz reflejada por el objeto llega a la lente del receptor y según el ángulo con la que sea recibida el blanco se encuentra a una distancia u a otra. Por otro lado, los sensores láser basados en el principio del ToF emiten un estrecho haz que viaja hacia el objeto cuya distancia se quiere conocer y se mide el tiempo transcurrido entre que se empite la luz y vuelve al receptor, tras incidir en el objeto. EL cálculo de la distancia una vez tenemos el tiempo es directo:

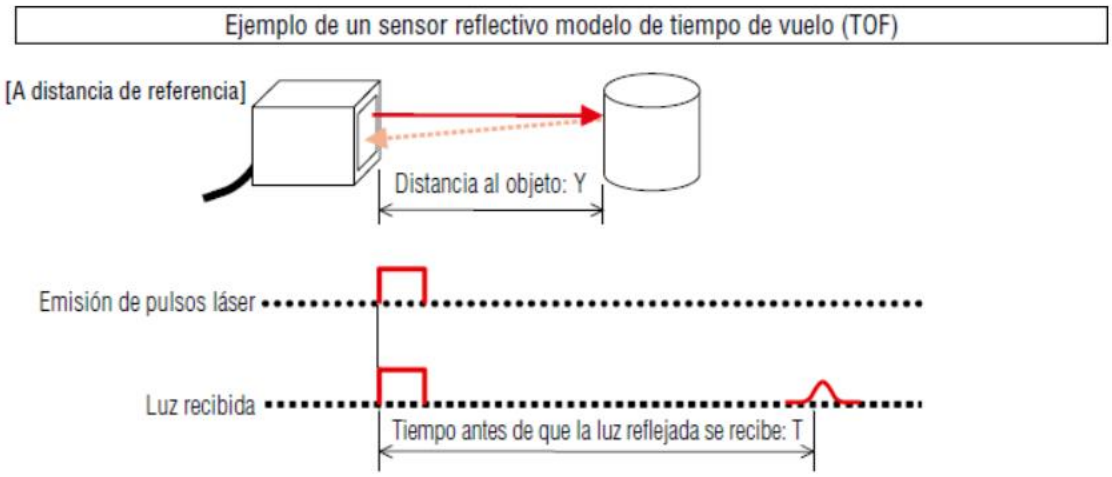


Figura 3: Principio de funcionamiento de un sensor láser de tipo ToF.

$$Y = \frac{T}{2} * c$$

Una de las principales ventajas es que, debido a la alta velocidad de la luz, el rayo es devuelto en muy poco tiempo y como ya hemos comentado anteriormente en varias ocasiones el tiempo es algo muy valioso en aplicaciones como la tratada en este proyecto, es decir, en aplicaciones de tiempo real. Además, al tratarse de una medida directa de la distancia tiene una alta precisión ya que eliminamos los errores de cálculo. En definitiva, son sensores muy precisos y direccionales, que se muestran invariantes ante la mayoría de las perturbaciones.

Pero este tipo de sensor tiene varios inconvenientes. Por un lado, es sensible a las condiciones medioambientales del entorno donde se hace la medida, es decir, se ve afectado por la lluvia, nieve o niebla ya que la luz láser puede rebotar de manera incorrecta y ofrecer mediciones de la distancia poco fiables. Además, tampoco pueden detectar objetos o superficies con una baja reflectancia ya que podrían absorber gran parte de la luz láser y hacer que el eco no llegue con suficiente intensidad al receptor.

En este trabajo vamos a usar uno de estos sensores ya que por precio y características nos parece el más adecuado. Nos proporciona una buena precisión y al ser direccionales podemos conocer exactamente el punto en el que está el obstáculo. Más concretamente vamos a emplear el sensor láser VL53L0X. La exhaustiva investigación del mismo en este proyecto surge como una idea de encontrar un sistema basado en la tecnología LIDAR, pero de bajo coste.

2. Diseño del sistema

2.1. Prototipo

El prototipo sobre el que vamos a estudiar este sistema de detección de obstáculos consiste en la estructura de un cuadricóptero sobre la que se montará el servo motor con los sensores acoplados a él mediante una pieza, como se muestra en la Figura 46 (ver Anexo I donde se desarrolla la impresión 3D de la pieza en cuestión). Para la realización de este trabajo se ha elegido la placa Arduino UNO como hardware para implementar la programación adecuada del sistema. El resto de elementos del hardware, es decir, tanto los sensores laser como el motor se conectarán a la placa para poder comunicarse con el ordenador. Dichas conexiones se harán siguiendo el esquema mostrado en la Figura 4.

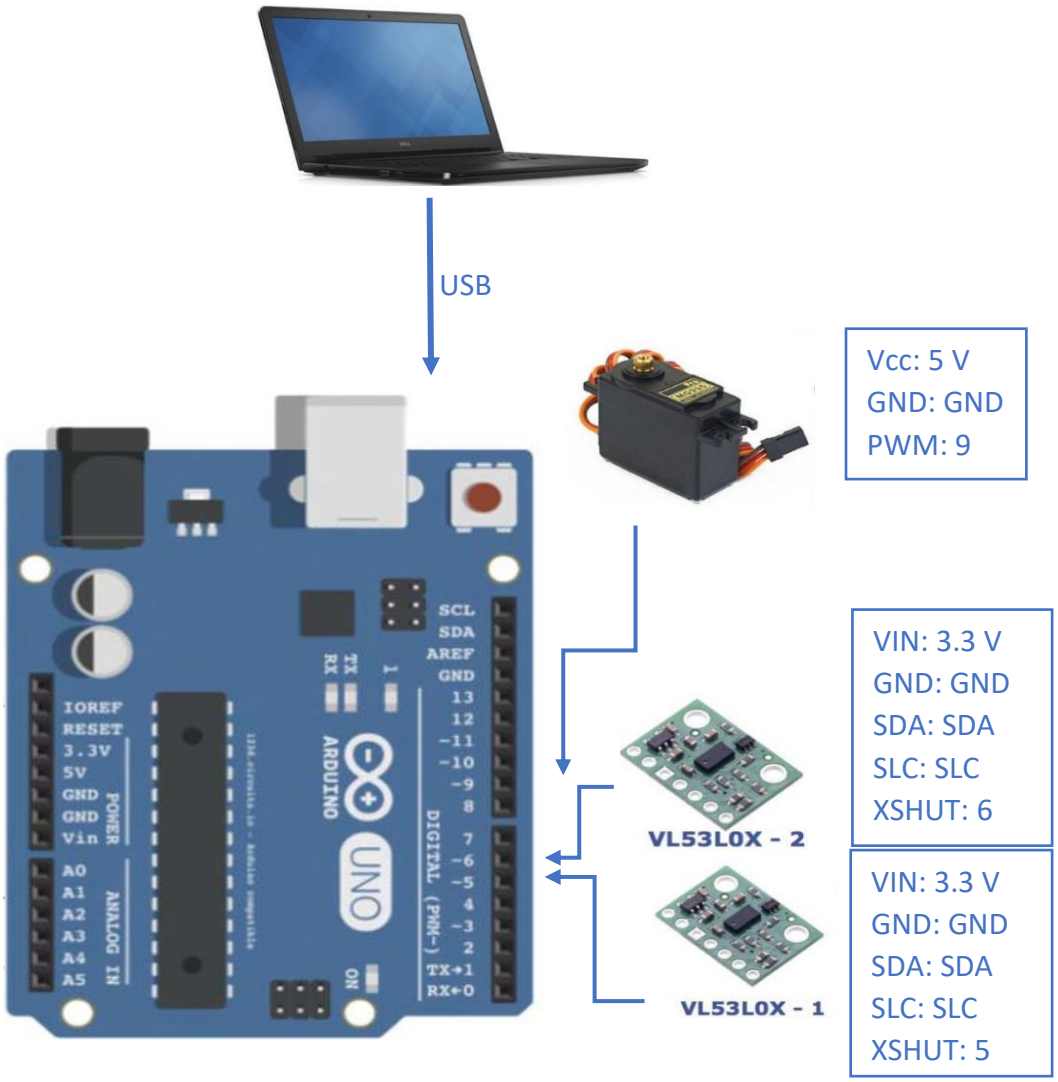


Figura 4: Esquema de conexiones de Arduino.

A su vez dicha placa está ubicada en el centro del cuadricóptero en un compartimento por debajo de donde se encuentra el motor y los sensores. La placa de Arduino debe estar conectada al ordenador mediante un cable USB, que si se quisiera en un futuro podría cambiarse por un módulo inalámbrico cuando este sistema se pruebe en drones reales. Por otro lado, el motor consta de cuatro tornillos que lo unen a una placa de madera que está pegada a la estructura. Por último, los sensores están fijados al motor mediante una pieza impresa en la impresora 3D tal y como se desarrolla en el Anexo I.

Así el montaje final del prototipo queda como sigue:

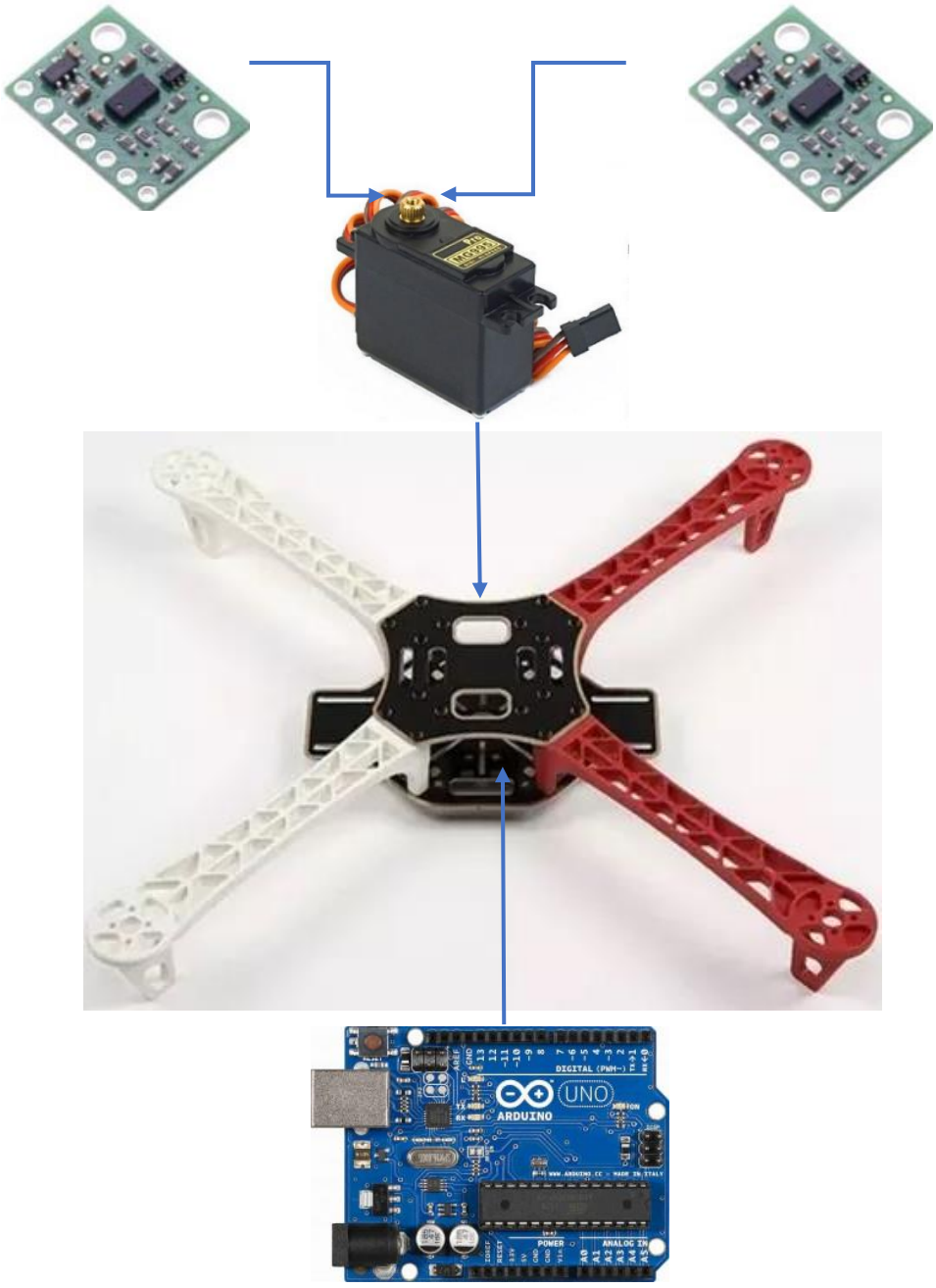


Figura 5: Esquema montaje físico del prototipo final.

Una vez montado el sistema tal y como se muestra en la Figura 6 el prototipo final queda como se puede ver en la siguiente imagen:

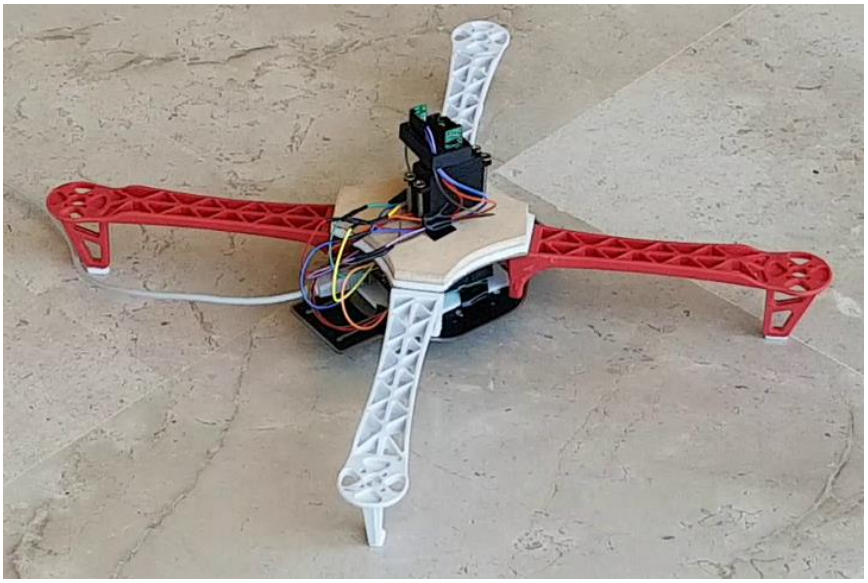


Figura 6: Montaje del prototipo final.

2.2. Hardware

En este apartado se va a hablar sobre el hardware que compone el sistema final. En concreto estos componentes son la placa de Arduino UNO, los sensores láser VL53L0X con los que se toman las medidas y, por último, el servomotor MG995 que dotará de movimiento a los sensores para poder barrer los 360º.

Se comenzará con una descripción detallada con el elemento principal de este montaje, la placa de Arduino UNO. Esta placa es la que nos permite controlar el resto de elementos desde nuestro ordenador y nos devuelve los datos necesarios para poder trabajar con ellos. Por este motivo se va a explicar en profundidad los elementos básicos que componen esta placa y se estudiarán también sus limitaciones para intentar sacar el máximo rendimiento de esta.

Seguidamente, se detallará también el funcionamiento y características del sensor láser VL53L0X, que es el elemento principal de este sistema. Se eligió este módulo por su precisión y tamaño, ya que tenía características optimas dentro del rango de precios en los que podíamos trabajar. Además, dispone de una librería completa, sencilla y gratuita para trabajar con Arduino lo que nos facilitará mucho el trabajo a la hora de desarrollar este sistema.

Por último, se va a describir el servomotor MG955 que vamos a utilizar para realizar el barrido con los sensores y medir la distancia a los obstáculos en los distintos radiales. Con este elemento podemos girar solamente un ángulo determinado pudiendo así hacer

las paradas deseadas lo que nos da una gran libertad a la hora de diseñar nuestro sistema.

2.2.1 Placa Arduino UNO [7]

Arduino es una plataforma electrónica de código abierto (open-source) basada en hardware y software fácil de usar. Las placas Arduino pueden leer entradas (luz en un sensor, un dedo presionando un botón o un mensaje de Twitter) y convertirlo en una salida (activar un motor, encender un LED y publicar algo en línea). Se le puede decir a la placa qué hacer enviando un conjunto de instrucciones al microcontrolador de esta. Para hacerlo, se utiliza el lenguaje de programación Arduino, basado en el conocido lenguaje C, y el software Arduino (IDE) .

Con los años, Arduino ha sido la base de miles de proyectos, desde objetos cotidianos hasta complejos instrumentos científicos. Una comunidad mundial de fabricantes (estudiantes, aficionados, artistas, programadores y profesionales) se ha reunido en torno a esta plataforma de código abierto, sus contribuciones se han añadido a una increíble cantidad de conocimiento accesible que puede ser de gran ayuda para principiantes y expertos por igual. Este es uno de los motivos por los que se ha elegido esta plataforma de desarrollo, ya que en internet podemos encontrar una gran comunidad de usuarios y foros muy activos que ayudarán al desarrollo de cualquier aplicación.

Arduino nació en el *Ivrea Interaction Design Institute* como una herramienta fácil para el prototipado rápido, dirigido a estudiantes sin experiencia en electrónica y programación. Tan pronto como llegó a una comunidad más amplia, la placa Arduino comenzó a cambiar para adaptarse a las nuevas necesidades y desafíos, diferenciando su oferta de simples placas de 8 bits a productos para aplicaciones IoT (Internet of Things, Internet de las cosas), impresión 3D y entornos integrados. Todas las placas Arduino son completamente de código abierto, lo que permite a los usuarios construirlos de forma independiente y eventualmente adaptarlas a sus necesidades particulares. El software también es de código abierto y está creciendo a través de las contribuciones de los usuarios en todo el mundo.

Arduino Uno es una placa de microcontrolador basada en ATmega328P. Tiene 14 pines digitales de entrada / salida (de los cuales 6 se pueden usar como salidas PWM), 6 entradas analógicas, un cristal de cuarzo de 16 MHz, conexión USB, conector de alimentación, encabezado ICSP y botón de reinicio. Contiene todo lo necesario para soportar el microcontrolador; simplemente conectándolo a un ordenador con un cable USB o con un adaptador de CA a CC o batería para comenzar.

Figura 7: Placa Arduino UNO.

Las especificaciones técnicas de esta placa Arduino UNO Rev3 quedan recogidos en la Tabla 1, cuyo conocimiento es necesario para el correcto funcionamiento de la misma.

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13

Length	68.6 mm
Width	53.4 mm
Weight	25 g

Tabla 1: Aspectos técnicos de la placa Arduino UNO

En el análisis a realizar de este hardware es necesario profundizar en las características del microcontrolador utilizado y los diferentes protocolos de comunicación que podemos utilizar para comunicarnos con ella.

MICROCONTROLADOR [8]

La placa con la que vamos a trabajar está basada en el microcontrolador ATmega328P, que podemos ver en la Figura 8, que es un chip microcontrolador creado por Atmel y pertenece a la serie megaAVR.

Figura 8: Microcontrolador ATmega328P.

Un microcontrolador se define como un circuito integrado programable, capaz de ejecutar las ordenes grabadas en su memoria. Está compuesto de tres bloques funcionales principales: CPU, memoria y los periféricos de entrada/salida. Dichos bloques se conectan entre sí mediante grupos de líneas eléctricas denominados buses.

En cuanto al ATmega328p, es un microcontrolador de 8 bits de alto rendimiento basado en una arquitectura tipo RISC. Tiene una frecuencia de 16 MHz, 32 KB de memoria flash una memoria con la capacidad de leer-mientras-escribe, 1 KB de memoria EEPROM, 2 KB de SRAM, 23 líneas de E/S de propósito general, 32 registros de proceso general, tres temporizadores flexibles/contadores con modo de comparación, interrupciones internas y externas, programador de modo USART, una interfaz serial orientada a byte de 2 cables, SPI puerto serial, 6-canales 10-bit Conversor A/D , "watchdog timer" programable con oscilador interno, y cinco modos de ahorro de energía seleccionables por software. En la Figura 9 aparece representado el mapa de los pines del ATmega328p y su conexión con la placa Arduino UNO.

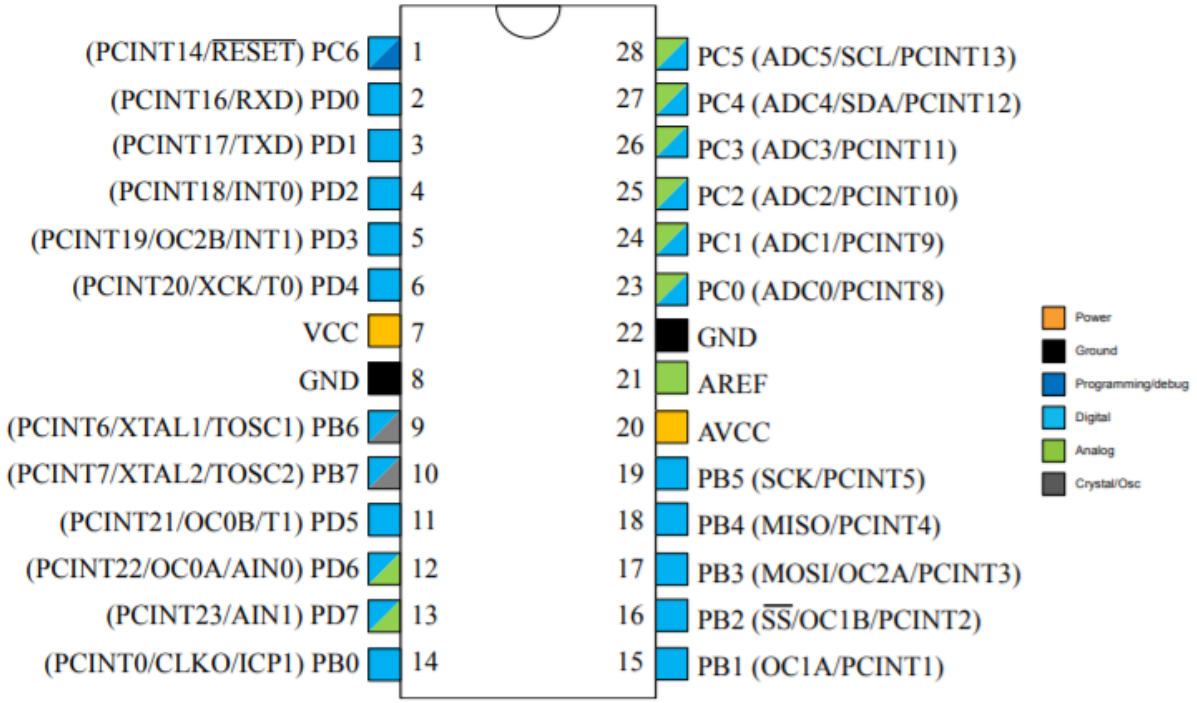


Figura 9: Mapa de pines de ATmega328P.

COMUNICACIÓN [8]

Existen diferentes protocoles de comunicación entre la placa Arduino y otro dispositivo, o incluso entre dos placas Arduino. Los puertos serie son la forma principal de comunicar una placa Arduino con un ordenador. Los pines asignados a esta tarea son los pines 0 (Rx) y 1 (Tx) de recepción y transmisión respectivamente.

En nuestro caso solamente se van a desarrollar el protocolo de comunicación I^2C (Inter-Integrated Circuit) ya que es los más usados, es con el que nosotros trabajaremos. Este es un protocolo asíncrono que se encuentra dentro de la comunicación serie.

El estándar I^2C, también denominado TWI, fue desarrollado por Philips en el año 1982 como método de comunicación interna entre sus dispositivos electrónicos. De forma progresiva, fue adoptado por otros fabricantes, hasta convertirse en uno de los protocolos de comunicación más utilizado en el mercado hoy en día.

El protocolo TWI permite al diseñador de sistemas interconectar hasta 128 dispositivos diferentes utilizando solo dos líneas de bus bidireccionales: una para reloj (SCL) y otra para datos (SDA). El único hardware externo necesario para implementar el bus es una sola resistencia de extracción para cada una de las líneas de bus TWI. Todos los dispositivos conectados al bus tienen direcciones individuales, y los mecanismos para resolver la contención del bus son inherentes al protocolo TWI.

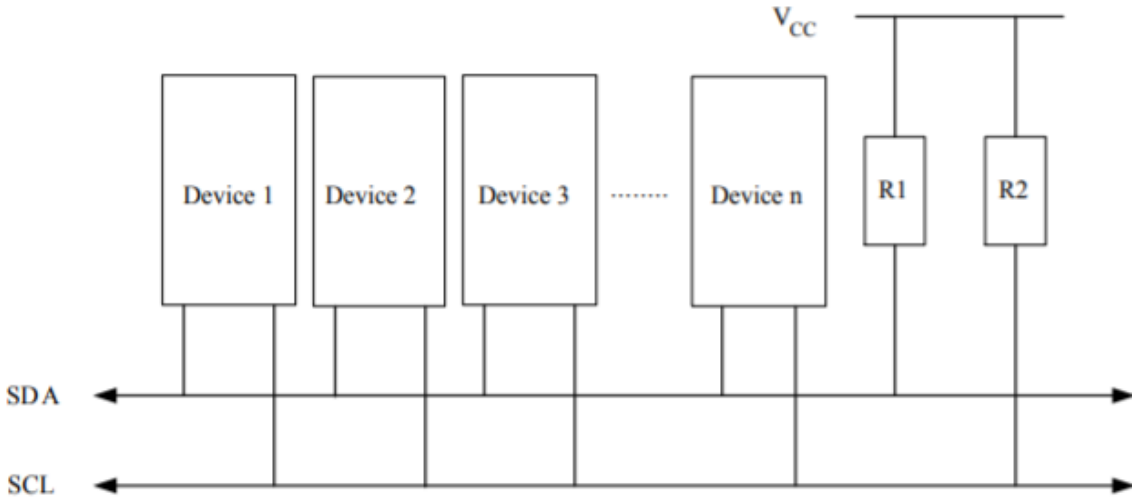


Figura 10: Interconexión de un Bus TWI.

El problema principal es que esta dirección de acceso, en la mayoría de las ocasiones, viene preestablecida por el fabricante. Este hecho ocasiona que, cuando se quieren conectar varios dispositivos del mismo tipo, todos presentan la misma dirección de acceso; por lo que será necesario cambiar dicha dirección.

2.2.2 Sensor Láser: VL53L0X [9]

El sensor VL53L0X es un dispositivo de nueva generación que utiliza la tecnología Time of Flight (ToF), es decir, en el cálculo de la distancia a partir del tiempo que tarda la luz en viajar hasta el objetivo y volver de nuevo al sensor. En la actualidad este es el medidor de distancias basado en laser con el encapsulamiento más pequeño lo que nos hizo decantarnos por él, entre otros motivos, ya que el tamaño y el peso en aplicaciones para drones es un parámetro muy importante. Es capaz de proporcionar una medición precisa de la distancia a un objetivo sin importar cuál sea la reflectancia de este, a diferencia de otras tecnologías tradicionales. Es papaz de medir distancias absolutas de hasta 2 m, estableciendo un nuevo punto de referencia en niveles de cualidades técnicas de medida. Este es el otro motivo principal por el que se eligió este sensor, ya que cuanto más lejos sea capaz de detectar los obstáculos más tiempo de maniobra tendremos para esquivarlos.

Figura 11: Sensor láser VL53L0X.

El VL53L0X integra, también, una matriz de SPADs (Single-Photon Avalanche Diodos), un fotodetector de estado sólido de vanguardia utilizado en aplicaciones en las que los detectores tradicionales ya no pueden distinguir la diferencia entre señal y ruido. Así, dicha matriz, constituye el receptor integrado de este dispositivo cuya tecnología utilizada se denomina FlightSenseTM, tecnología de segunda generación patentada por el propio fabricante (ST Microelectronics).

El emisor láser es del tipo VCSEL (Vertical-Cavity Surface-Emitting Lasers), un láser semiconductor en el que la luz se propaga perpendicularmente al plano de la región activa con una longitud de onda de 940nm no visible por el ojo humano. En este emisor se ha combinado con filtros internos de infrarrojos que permiten detectar objetos a distancias más largas, mayor inmunidad a la luz ambiental y una mejor robustez ante el cross-talk (diafonía) debida al cristal protector.

Las especificaciones técnicas nos las proporciona el fabricante mediante la Tabla 2 que se muestra a continuación.

Feature	**Detail**
Package	Optical LGA 12
Size	4.40 x 2.40 x 1.00 mm
Overating voltage	2.6 to 3.5 V
Operating temperature	-20 to 70ºC
Infrared emitted	940 nm
I^2C	Up to 400 kHz (FAST mode) serial bus Address:0x52

Tabla 2: Especificaciones Técnicas VL53L0X.

Estos láseres consumen muy poca potencia y su fabricación resulta barata debido, entre otros aspectos, a su pequeño tamaño y a la estructura vertical que presentan. Este es, una vez más, otro incentivo para usar este dispositivo. Sin embargo, para el correcto funcionamiento de estos, es necesaria la existencia de espejos de alta reflectividad que generan el haz circular de luz.

En este trabajo se va a utilizar el sensor montado sobre una placa de reducidas dimensiones integrado por Polulu tal y como se puede ver en la Figura 12. La placa tiene un regulador lineal de 2.8 V e indicadores de nivel integrados que le permiten trabajar en un rango de voltaje de entrada de 2.6 V a 5.5 V, y el espaciado de pin de 0.1 lo que hace que sea fácil de usar con placas de prueba sin soldadura estándar y perfboards de 0.1.

Figura 12:Sensor VL53L0X integrado por Polulu.

PROCESOS DE CALIBRACIÓN [10]

El fabricante nos recomienda la calibración que debemos llevar a cabo a nivel de consumidor, solamente una vez al comenzar la utilización del sensor, para facilitarnos dicho proceso nos da un flujograma de los pasos a seguir (Figura 13). Este flujograma ya tiene en cuenta todos los parámetros (cristal protector, temperatura y voltaje).

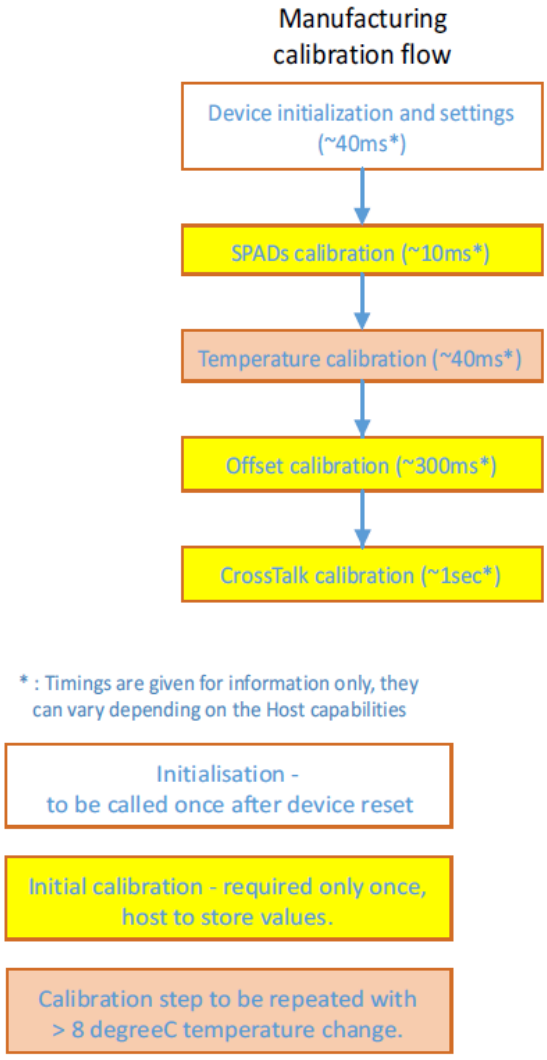


Figura 13: Flujograma de calibración del sensor VL53L0X.

Calibración SPAD

Para optimizar la dinámica del sistema, es necesario calibrar el SPAD de referencia. Este paso se realiza durante el proceso de producción una sola vez, y los datos derivados de esta calibración (número y tipo de SPADs) son almacenados en la memoria NVM del sensor.

Únicamente, en el caso de utilizar una carcasa de protección encima del VL53L0X, se han de calibrar de nuevo los SPADs de referencia a nivel usuario. El procedimiento de calibración no requiere condiciones específicas, ni de iluminación, ni de objetivo.

EL procedimiento a seguir para realizar esta calibración es:

- Llamar a *VL53L0X_PerformRefSpadManagement()*
    - Esta función devuelve el número y tipo de SPADs de referencia que debe ser usado.
- Al finalizar, almacena ambos valores en la memoria NVM del dispositivo. Durante el proceso de inicialización, el host tiene que guardar estos dos valores para aplicarlos posteriormente a la corrección de la medida.

Las funciones *VL53L0X_SetReferenceSpads()* y *VL53L0X_GetReferenceSpads()* pueden ser usadas para fijar/obtener el número y tipo de SPADs de referencia.

Calibración Ref (temperatura)

La calibración de la temperatura es, en realidad, la calibración de dos parámetros, que son dependientes de la temperatura, utilizados para ajustar la sensibilidad del dispositivo cuando la temperatura varía. Dicha calibración es realizada durante el proceso de producción y sus parámetros son guardados en la memoria del Host. En un principio no es necesario volver a calibrar. Para realizar esta calibración la función utilizada es:

VL53L0XSetRefCalibration()

Únicamente es necesario repetir el proceso de calibración cuando la temperatura varíe más de 8ºC en comparación con la temperatura de calibración inicial (23 ºC). No se requiere ninguna configuración específica para realizarla, el único requisito existente es llevarla a cabo justo después de la calibración SPAD y antes de realizar la primera medida y las calibraciones de offset y cross-talk. La función destinada para ello es:

VL53L0XPerformRefCalibration()

Calibración del Offset de medida

El offset de medida puede ser caracterizado por el offset medio, que es la diferencia entre el valor central de la medida y el valor real de la distancia.

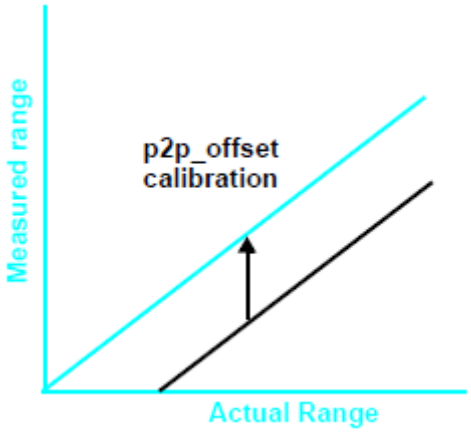


Figura 14: Offset de medida.

Esta calibración debe llevarse a cabo en el proceso de fabricación para optimizar las características y será almacenado el valor obtenido en la memoria NVM del sensor. Para hacerla se deben tener en cuenta el suministro de voltaje, la temperatura y el cristal protector del módulo láser. En algún caso el valor almacenado en memoria puede no ser correcto. Esto ocurre cuando el usuario está usando la carcasa protectora. En este caso la medida puede verse afectada por el offset por lo que el usuario debe realizar de nuevo la calibración del offset.

El procedimiento que se debe seguir en este tipo de calibración debe cumplir lo siguiente:

- Se recomienda usar un objetivo blanco (con un 88% o más de reflectancia) a 100 mm, en un ambiente oscuro. La distancia al blanco puede ser cambiada si las necesidades del usuario así lo demandan, pero debe hacerse siempre en la parte lineal de la curva de medición.
- Tanto la calibración de SPADs como la de temperatura deben haber sido realizadas antes de llevar a cabo esta.
- Debe llamarse a la siguiente función para calcular el offset:

  VL53L0X_PerformOffsetCalibration().
- La salida de esta función es el valor del offset en la calibración, en micrómetros.
- EL valor de la calibración debe ser almacenado en la memoria Host.

Calibración Cross-talk

Cross-talk (diafonía) es definido como la señal devuelta por el cristal protector. La magnitud del cross-talk depende del tipo de cristal y del aire que se encuentre entre el emisor y este. La corrección de este fenómeno es básicamente la aplicar una ganancia ponderada a los datos de medida, basándonos en los resultados de la calibración.

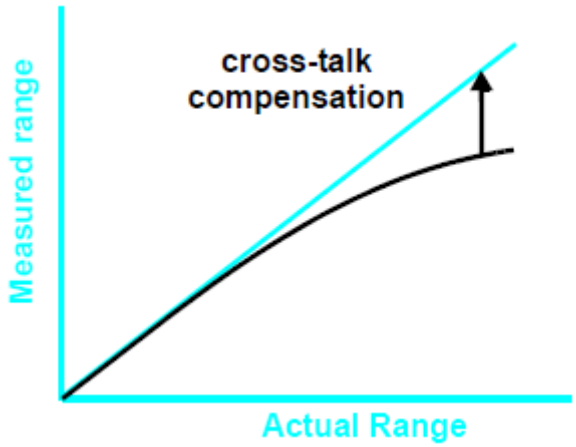


Figura 15: Compensación de cross-talk.

La distancia a la que se efectúa esta calibración depende de la calidad del cristal protector, la calibración de un cross-talk bajo no puede ser caracterizada igual que la de uno alto. El punto de partida al partir del cual la distancia de calibración se toma como válida es en aquel en el que la señal medida empieza a desviarse de la curva ideal ya que si la calibración se hace en la parte lineal de la curva de medidas el factor de corrección será muy bajo y la corrección no tendrá efecto. EL rango en el que son válidas las distancias termina cuando la señal empieza a ser demasiado baja (la distancia medida empieza a disminuir). Para comprender mejor esto se representa con un ejemplo en la Figura 16.

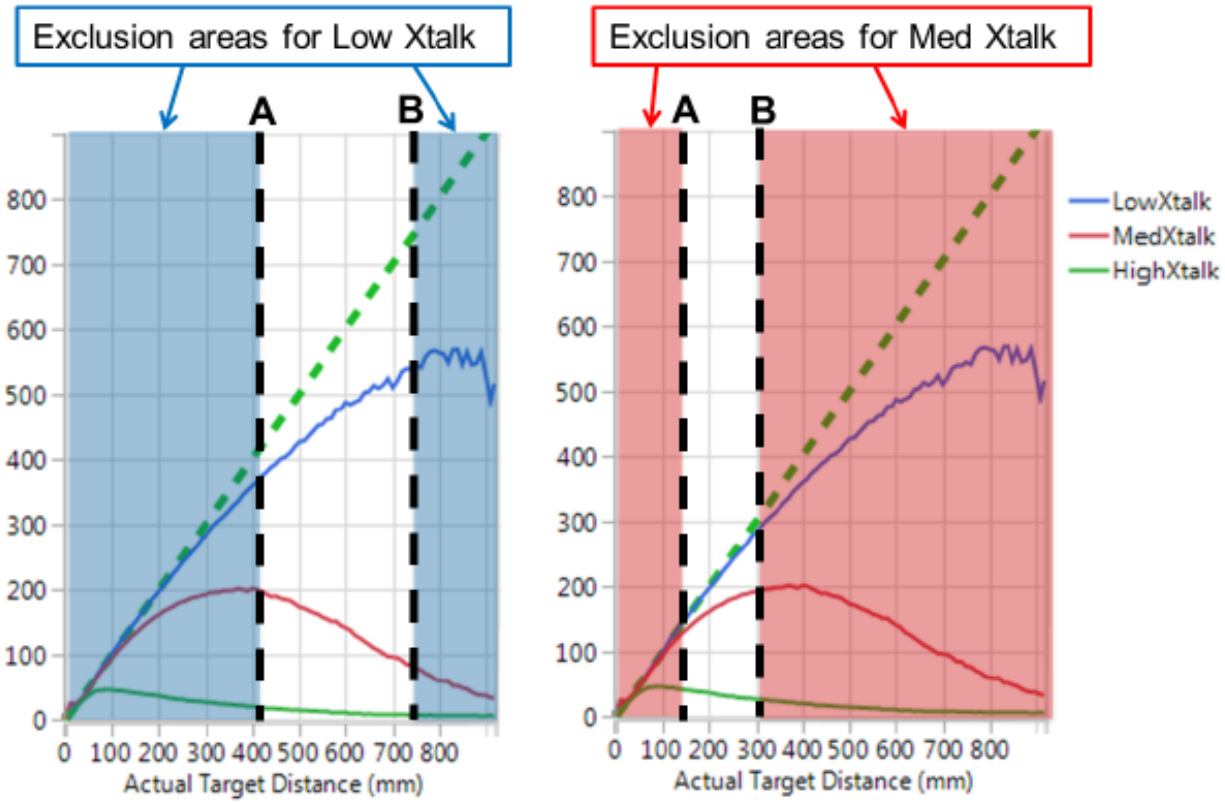


Figura 16: Distancias válidas para la calibración de cross-talk.

El procedimiento de calibración debe seguir las siguientes indicaciones:

- Elección de la distancia de calibración, basándonos en el tipo de cristal protector usado (ver Figura 16).
- Usar un objetivo gris, con un 17% de reflectividad.
- Llamar a la función de calibración: *VL53L0X_PerformXTalkCalibration()*.
    - La entrada de esta función será la distancia de calibración en milímetros.
    - La salida es el factor de cross-talk. Este debe ser almacenado en la memoria Host.
    - La función aplica y habilita la corrección necesaria.

MODOS DE MEDICIÓN

La API del sensor nos permite tres modos de medición:

1. Single ranging (medición única): La medida de distancia se realiza una sola vez después de llamar a la correspondiente función API. Tras la medición el sistema vuelve de forma automática a la posición de standby de software.
2. Continuous ranging (medición continua): La medición se realiza de forma continuada después de llamar a la correspondiente función API. Tan pronto como se termina una medida otra empieza sin esperas. Es el usuario el encargado de parar la medición y volver a la posición de standby de software. Siempre antes de terminar se espera a completar la última medida en curso.
3. Timed ranging (medición temporizada): La medición se realiza de forma continuada después de llamar a la correspondiente función API. A diferencia del caso anterior, cuando una medida termina se espera un tiempo determinado, definido por el usuario, antes de comenzar la siguiente medida. Este retardo o periodo entre medidas puede ser definido a través de la API. Al igual que en el modo continuo el usuario debe parar la medición y volver a la posición de standby de software para finalizar. Si la petición de parada se lleva a cabo durante una medida esta debe completarse antes de terminar. Por el contrario, si ducha petición ocurre durante un periodo entre medidas el proceso parará inmediatamente.

PERFILES DE MEDICIÓN [10]

La Api nos proporciona código para realizar cuatro perfiles de medición distintos, aunque el usuario puede crear su propio perfil de medición según las características requeridas en su caso. Los perfiles proporcionados por la API pueden verse resumidos en la Tabla 3.

Range Profile	Range timing budget	Typical performance	Typical application
Default mode	30ms	1.2m, accuracy as per *Table 12*	standard
High accuracy	200ms	1.2m, accuracy < +/- 3%	precise measurement
Long range	33ms	2m, accuracy as per *Table 12*	long ranging, only for dark conditions (no IR)
High speed	20ms	1.2m, accuracy +/- 5%	high speed where accuracy is not priority

Tabla 3: Perfiles de medición proporcionados por la API.

- Default mode (Modo estándar): Es el perfil por defecto por lo que no hay que escribir ningún código específico para establecerlo como perfil de medición.
- High accurancy (Alta precisión): Para trabajar en este modo se debe configurar con el siguiente código AP antes de empezar la medición.

```
if (Status == VL53L0_ERROR_NONE) {
   Status = VL53L0_SetLimitCheckValue(pMyDevice,
   VL53L0_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE,
  (FixPoint1616_t)(0.25*65536));
  }
if (Status == VL53L0_ERROR_NONE) {
  Status = VL53L0_SetLimitCheckValue(pMyDevice,
  VL53L0_CHECKENABLE_SIGMA_FINAL_RANGE,
  (FixPoint1616_t)(18*65536));
  }
if (Status == VL53L0_ERROR_NONE) {
   Status =
   VL53L0_SetMeasurementTimingBudgetMicroSeconds(pMyDevice,
   200000);
   }
```

- Long range (Largo alcance): Para trabajar en este modo se debe configurar con el siguiente código AP antes de empezar la medición.

```
if (Status == VL53L0_ERROR_NONE) {
    Status = VL53L0_SetLimitCheckValue(pMyDevice,
    VL53L0_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE,
   (FixPoint1616_t)(0.1*65536));
    }
if (Status == VL53L0_ERROR_NONE) {
```

```
    Status = VL53L0_SetLimitCheckValue(pMyDevice,
    VL53L0_CHECKENABLE_SIGMA_FINAL_RANGE,
    (FixPoint1616_t)(60*65536));
    }
if (Status == VL53L0_ERROR_NONE) {
    Status =
    VL53L0_SetMeasurementTimingBudgetMicroSeconds(pMyDevice,
    33000);
    }
if (Status == VL53L0_ERROR_NONE) {
    Status = VL53L0_SetVcselPulsePeriod(pMyDevice,
    VL53L0_VCSEL_PERIOD_PRE_RANGE, 18);
    }
if (Status == VL53L0_ERROR_NONE) {
    Status = VL53L0_SetVcselPulsePeriod(pMyDevice,
    VL53L0_VCSEL_PERIOD_FINAL_RANGE, 14);
    }
```

- High speed (Alta velocidad): Para trabajar en este modo se debe configurar con el siguiente código AP antes de empezar la medición.

```
if (Status == VL53L0_ERROR_NONE) {
    Status = VL53L0_SetLimitCheckValue(pMyDevice,
    VL53L0_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE,
    (FixPoint1616_t)(0.25*65536));
    }
if (Status == VL53L0_ERROR_NONE) {
    Status = VL53L0_SetLimitCheckValue(pMyDevice,
    VL53L0_CHECKENABLE_SIGMA_FINAL_RANGE,
    (FixPoint1616_t)(32*65536));
    }
if (Status == VL53L0_ERROR_NONE) {
    Status =
    VL53L0_SetMeasurementTimingBudgetMicroSeconds(pMyDevice,
    20000);
    }
```

FASES DE MEDICIÓN

A su vez cada uno de los perfiles de medición comentados en el apartado anterior conta de tres fases consecutivas. El flujograma de la Figura 17 muestra esquemáticamente cómo se deben llevar a cabo estas fases.

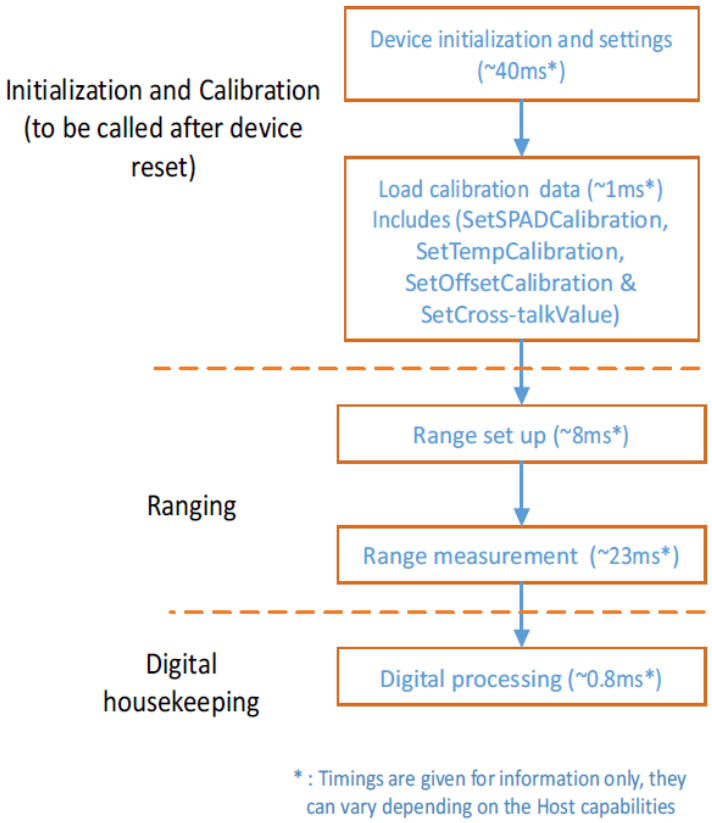


Figura 17: Esquema de fases de medición típicas.

Inicialización y cargar datos de calibración [10]

La fase de inicialización y calibración debe llevarse a cabo antes de la primera medida o después de un reset. Puede ser que el usuario necesite repetir la fase de calibración de temperatura de forma periódica, dependiendo del caso de uso. Se puede ver más claramente como llevar a cabo esta fase en el flujograma que se muestra en la Figura 18. Para realizar la inicialización del sistema se debe llamar a las funciones API en el orden que se muestra en la Figura 18. Se debe tener en cuenta que el nombre completo de las funciones no es el que se muestra en el flujograma, sino que hay que añadirle *"VL53L0X_"* delante, por ejemplo la primera función a llamar es *VL53L0X_DataInit()* y lo mismo ocurre con todas las demás.

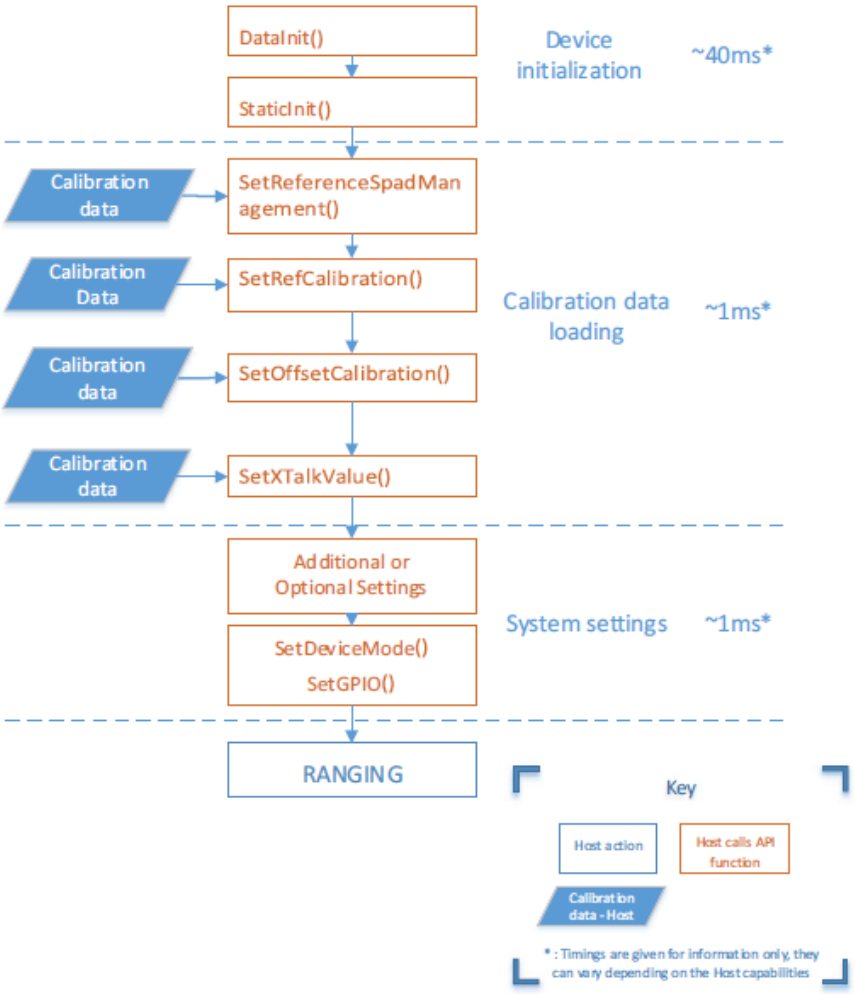


Figura 18: Flujograma de la fase de inicialización y calibración.

Medición [10]

La fase de medición consiste en la configuración previa de una serie de parámetros y en la medición propiamente dicha.

Durante la operación de medición, muchos pulsos infrarrojos VCSEL son emitidos, después se refleja en el objetivo y es detectado por el receptor. El fotodetector usado dentro del VL53L0X usa una avanzada tecnología de ultra-rápido SPAD (Single Photon Avalanche Diodes), protegida por numerosas patentes.

El promedio de tiempo para una secuencia de medición completa es de 33 ms (inicialización + medición + procesado digital), siendo el tiempo real de la medida de la distancia de 23 ms. El promedio de tiempo mínimo para una secuencia de medición es de 20ms, y el máximo 5 segundos. Cuanto mayor es este promedio de tiempo, mayor es la precisión y la capacidad de tomar medida en un rango mayor.

El flujo a seguir para llevar a cabo esta fase se esquematiza en la Figura 19.

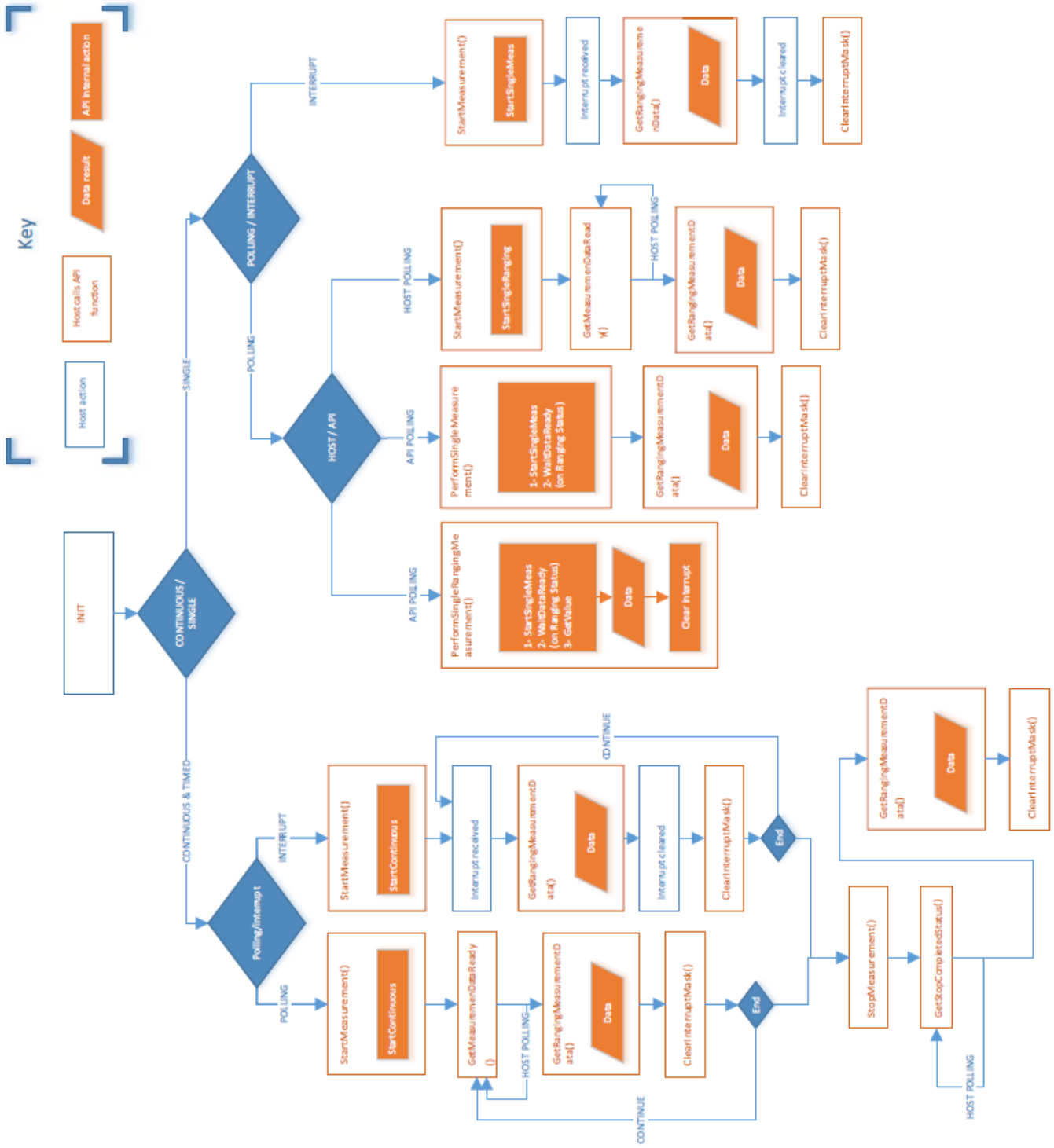


Figura 19: Flujograma de la fase de medición.

Procesado digital (Digital housekeeping)

El procesado digital es la última operación dentro de la secuencia de medición que calcula, valida o rechaza la medida. Parte de este procesado es llevado a cabo internamente mientras otras partes se están ejecutando en el Host por la API.

Al final del procesado digital la distancia medida es calculada por el VL53L0X directamente. Si la distancia no puede ser medida (señal débil, ningún objetivo…), se devuelve el correspondiente código de error.

Las siguientes funciones son llevadas a cabo dentro del mismo sensor:

- Comprobación del valor de la señal (señal débil).
- Corrección del offset.
- Corrección del cross-talk (en caso de que haga cristal protector).
- Valor final de la medida calculada.

Mientras tanto la API lleva a cabo lo siguiente:

- Comprobación RIT (Return Ignore Threshold), verificación de la señal en función del cross-talk.
- Comprobación de sigma (condición de precisión).
- Cálculo del estado final de la medida.

Si el usuario quiere mejorar la precisión de la medida, puede llevarse a cabo algún procesado extra (ajeno a la API) por parte del host, por ejemplo, un promediado continuo, histéresis o cualquier tipo de filtrado.

OBTENCIÓN DE DATOS [10]

El usuario puede obtener los datos finales usando dos tipos de mecanismos:

- Modo Polling: El usuario debe comprobar el estado del proceso de medición haciendo una consulta contante te una función API. Dentro de este modo, la función *VL53L0X_GetMeasurementDataReady()* permite al Host conocer en qué estado se encuentra la medición que se esté llevando a cabo en ese momento, y la función *VL53L0X_GetRangingMeasurementData()* devuelve los datos de dicha medición. Entre estos datos, se encuentra también, entre otros, un valor que indica el estado de la medida tomada, si es válida o no, y los posibles fallos del dispositivo.
- Modo interrupción: un pin de interrupción (GPIO1) envía una interrupción al Host cuando una nueva medida está disponible. En este modo, la función *VL53L0X_SetGPIOConfig()* permite configurar las características de interrupción del sistema.

2.2.3 Servomotor MG995

Un servomotor es un dispositivo similar a un motor de corriente continua pero que tiene la capacidad de ubicarse en cualquier posición, dentro de su rango de operación, y mantenerse estable en dicha posición. En otras palabras, un servomotor es un motor especial al que se ha añadido un sistema de control (tarjeta electrónica), un potenciómetro y un conjunto de engranajes. Con anterioridad los servomotores no permitían que el motor girara 360 grados, solo aproximadamente 180; sin embargo, hoy en día existen servomotores en los que puede ser controlada su posición y velocidad en los 360 grados.

En este trabajo fin de grado se emplea un servomotor para poder realizar un barrido de 360º alrededor del dron y así conocer todos los posibles obstáculos que hay en su entorno con la finalidad de poder evitarlos. Con la finalidad de realizar dicho barrido en el mínimo tiempo posible ya que esta es una variable fundamental en nuestro caso se ha elegido colocar dos sensores en vez de uno apuntando a direcciones opuestas y así el motor sólo debe girar 180º en vez de 360º. Con ello cada uno de los sensores laser mide a distancia a los obstáculos en un ángulo suplementario al otro.

Para realizar esta función se ha elegido el servomotor MG995 ya que en nuestro caso cumplía con los requerimientos para esta primera prueba de concepto. En ampliaciones posteriores de este proyecto se podría elegir un servomotor más ligero, ya que se encuentra embarcado en un dron y el peso es fundamental, y que gire lo más rápido posible ya que así el dron podría volar más rápido sin peligro a colisionar ya que le dará tiempo a realizar todo el barrido en un tiempo mucho menor. En nuestro caso el principal cuello de botella de la temporización es que hay que darle un tiempo de espera para que el motor pueda terminar el giro antes de medir.

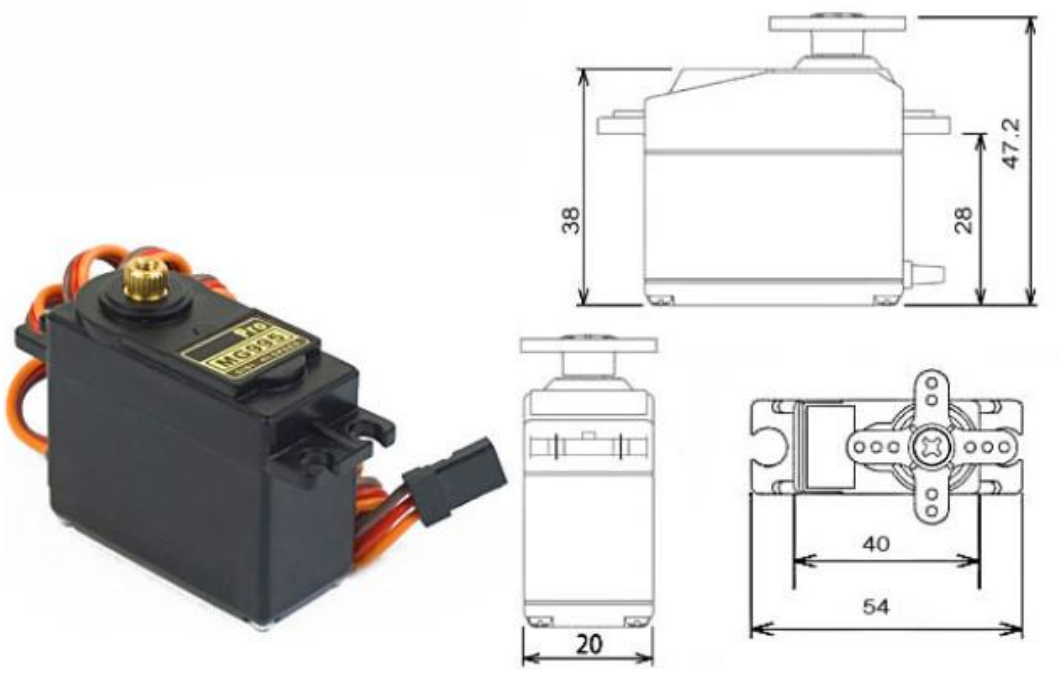


Figura 20: Motor MG995 y vistas.

La unidad viene con 30 cm de cable con tres pines 'S' tipo hembra. Para la programación de este se puede usar cualquier código destinado a servos, en nuestro caso Arduino tiene una librería con funciones muy sencillas. Con el pack te vienen numerosas piezas para acoplarle, pero puesto que nuestras necesidades eran tan específicas se ha hecho una pieza específica, cuyo desarrollo se puede observar en la sección X. Las características de este motor son las siguientes:

Weight	55g	
Dimensions	40.7 x 19.7 x 42.9 mm	
Stall torque	4.8 V	8.5 kgf*cm
	6 V	10 kgf*cm
Operating speed	4.8 V	0.2 s/ 60º
	6 V	0.16 s/ 60º
Operating voltage	4.8 V to 7.2 V	
Dead band width	5 µ	
Temperature range	0ºC – 55ºC	

Tabla 4: Características servomotor MG995

Las conexiones con la placa Arduino deben hacerse siguiendo la Figura 21 tal y cuya equivalencia en pines de Arduino se muestra en la Figura 4.

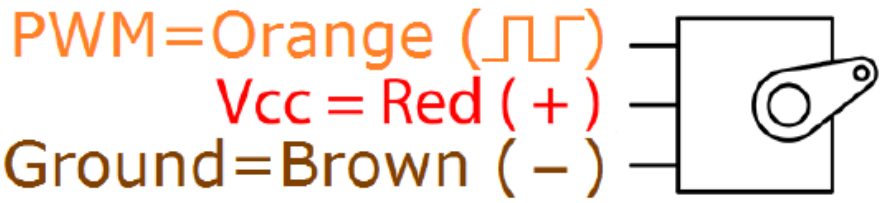


Figura 21: Conexiones servomotor MG995.

Este motor servo, como cualquier otro funciona mediante la modulación de una señal cuadrada que, dependiendo del ciclo de trabajo de esta el motor gira un determinado ángulo, es decir, hace uso de la modulación por ancho de pulsos (PWM) para controlar la dirección o posición de los motores de corriente continua. La mayoría de los servos, y este no es una excepción, trabajan en la frecuencia de los 50 Hz, así las señales PWM tendrán un periodo de veinte milisegundos. La electrónica dentro del servomotor responderá al ancho de la señal modulada. En la Figura 22 se puede ver un ejemplo de cómo respondería un servomotor ante distintas entradas de señal cuadrada dependiendo de su ancho de puso.

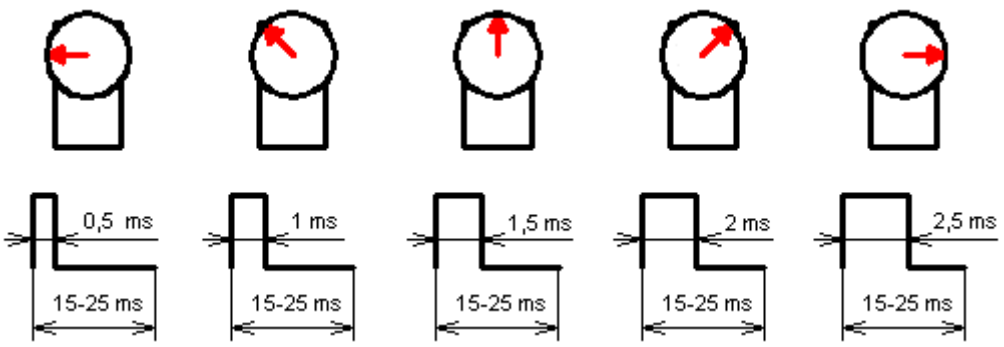


Figura 22:Ancho de pulso para lograr diferentes posiciones de un servomotor.

En nuestro caso el esquema de señal que nos proporciona el fabricante es muy similar a los mostrados en la Figura 22 pero especificando la amplitud de la señal y confirmándonos que trabaja a 50 Hz de frecuencia. Dicho esquema se puede ver en la Figura 23.

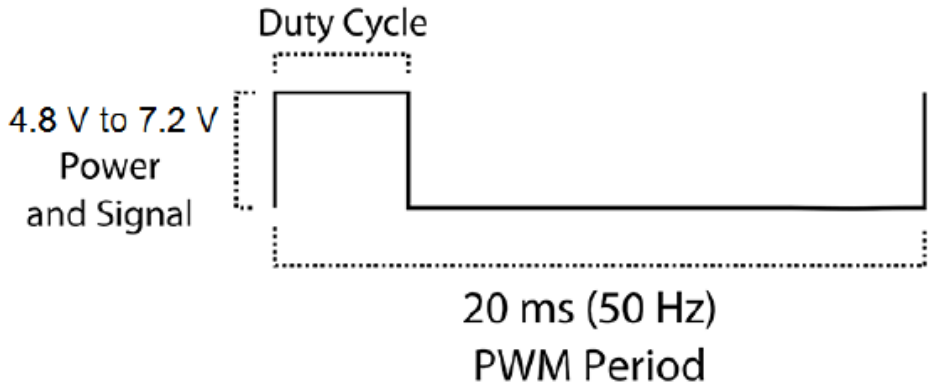


Figura 23: Señal PWM del servomotor MG559.

Puesto que trabajamos con Arduino la programación de este tipo de motores es muy sencilla, como podremos ver en apartado 2.3.2., ya que disponemos de una librería que se encarga de que toda la modulación de la señal sea transparente para el usuario.

2.3. Software

Una vez ya se ha explicado detalladamente el hardware del sistema diseñado y los esquemas de conexiones y montaje de todos los elementos de este prototipo vamos a proceder a explicar el software empleado. Los dos software principales sobre los que gira este proyecto son el IDE de Arduino con el que nos comunicaremos con la placa, tanto para darle las instrucciones de lo que tiene que hacer como para recibir datos desde los sensores. Por otro lado, también se emplea Processing para tomar dichos datos, guardarlos para su posterior análisis y graficar los resultados obtenidos para hacerlos más visibles y que se puedan entender de un solo vistazo.

Aunque estos sean los dos elementos principales sin los cuales el proyecto no podrá llevarse a cabo hay otros elementos también importantes que han contribuido a la realización de este Trabajo Fin de Grado. Se han utilizado varias aplicaciones de Microsoft Office, principalmente Excel para el estudio de los datos adquiridos durante la caracterización, tanto de los sensores individualmente como del sistema completo; Word para la redacción de este TFG y PowertPoint para la realización de la presentación que nos servirá de guía a la hora de defenderlo ante el tribunal. Por otro lado, también se ha empleado Inventor, un software de Autodesk en formato CAD, que nos ha permitido el diseño de la pieza que posteriormente imprimiremos con la impresora 3D (ver Anexo I).

En esta sección vamos a desarrollar en profundidad únicamente los dos Software principales, Arduino IDE y Processing, ya que son los menos conocidos por el usuario medio y en los que más se ha tenido que investigar para poder llevar a cabo este proyecto.

2.3.1 Arduino IDE [7]

El entorno de desarrollo integrado de Arduino, comúnmente denominado por sus siglas en inglés IDE (Integrated Development Environment), contiene un editor de texto para escribir el código, un área de mensajes, una consola de texto, una barra de herramientas con botones que realizan las funciones más comunes y una serie de menús. Este entorno conecta con el hardware de Arduino para subir los programas deseados y comunicarnos con él.

Los programas escritos usando Arduino IDE se llaman sketches. Estos sketches están escritos con el editor de textos y guardados en archivos con extensión .ino. El área de mensajes nos da la respuesta del programa al guardar o exportar un archivo, y también nos muestra los errores. La consola muestra la salida de texto que proporciona Arduino IDE, incluyendo los mensajes de error completos y otra información. La barra que se encuentra en la esquina inferior derecha muestra la configuración de la placa y el puerto serie al que está conectada. Y, por último, la barra de herramientas permite verificar y subir los programas, crear, abrir y salvar los sketches y abrir el monitor serie.

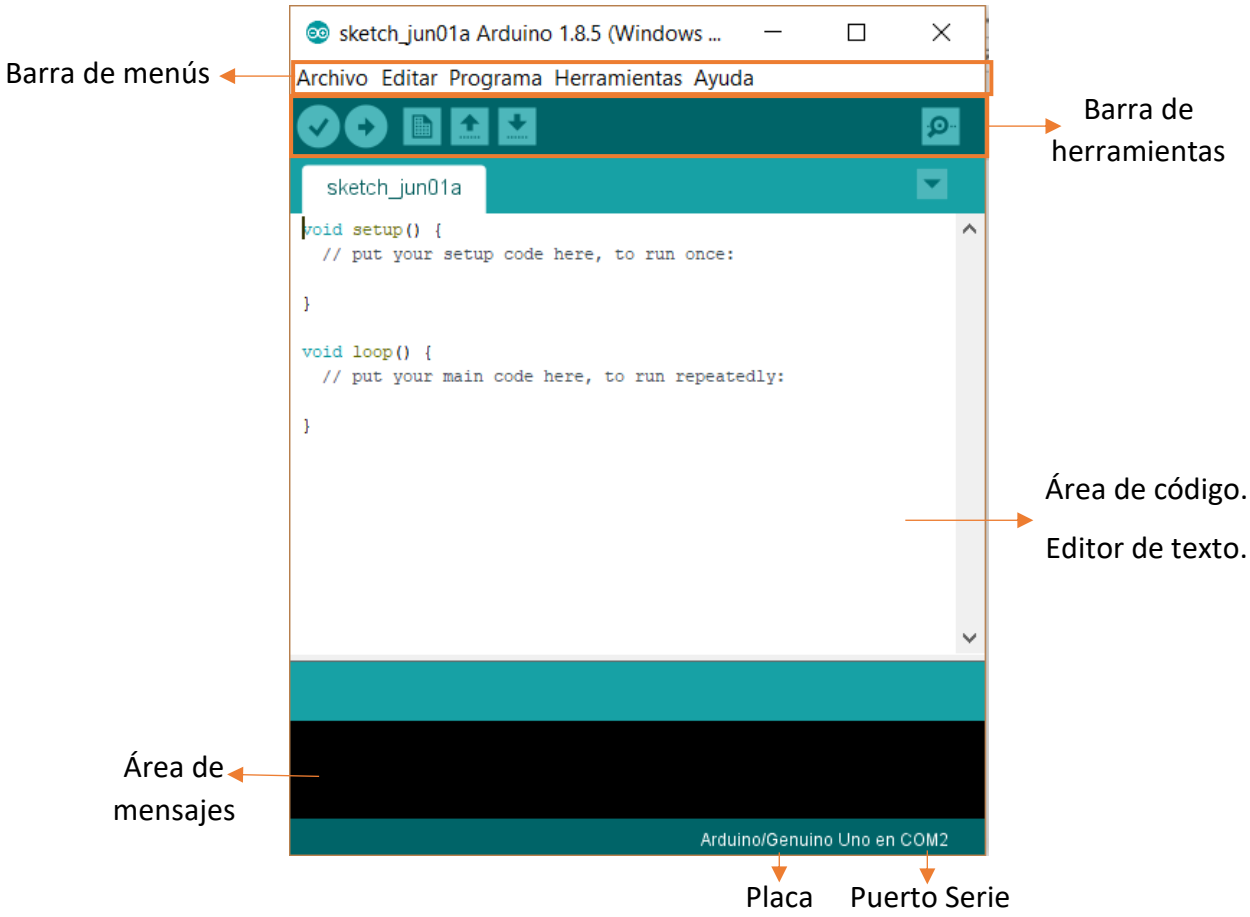


Figura 24: Interfaz de Arduino IDE.

Elementos de la barra de herramientas:

- Verificar*:* Comprueba su código para detectar errores al compilarlo. En caso de error lo muestra en el área de mensajes y sino el programa se compilará sin problemas.
- Cargar: Carga el código en la placa, en caso de no haber sido compilado previamente lo compilará antes de subirlo.
- Nuevo: Crea un nuevo sketch.
- Abrir: Presenta un menú de todos los sketches en su cuaderno. Al hacer clic en uno de ellos, se abrirá dentro de la ventana actual sobrescribiendo su contenido.
- Guardar: Guarda el sketch actual.
- Serial Monitor: Abre el monitor en serie que nos permite visualizar la comunicación e intercambio de datos con la placa mediante el puerto serie.

Además, dentro de los cinco menús se encuentran comandos adicionales. Los menús son contextuales, lo que significa que solo están disponibles elementos relevantes para que el trabajo que se esté llevando a cabo actualmente. Los menús más remarcables son los siguientes:

- Archivo: dentro de este menú se encuentran diferentes opciones tales como: nuevo proyecto, abrir uno ya existente, guardar el actual, etc. Una opción interesante es la de Ejemplos, opción que proporciona, como su propio nombre

indica, ejemplos sobre el uso de la placa Arduino; desde los más básicos hasta algunas más complejos. Esto sirve de gran ayuda para los usuarios que empiezan a trabajar con este software.

- Editar: en este menú aparecen las opciones básicas de edición como pueden ser: copiar, cortar, pegar, comentar / descomentar, etc.
- Herramientas: son varias las opciones que proporciona el IDE de Arduino dentro de este menú. Las más destacables son Placa y Puerto que permite la selección del modelo de la placa y el puerto en el que se encuentra conectada, debemos asegurarnos de que estos datos están bien configurados siempre para evitar posibles errores; y la opción de Quemar bootloader, que permitiría volver a cargar el bootloader en caso de que fuese necesario.

Todo sketch de Arduino sigue la siguiente estructura básica, esta se divide en tres elementos principales:

- Inicialización y declaración de variables: En la inicialización de un programa se deben incluir todas las librerías usadas en el programa. Por otro lado, se entiende por declarar una variable, crear un valor que Arduino puede guardar en su memoria para ser posteriormente utilizado o alterado. Esta sección solo se ejecuta una vez, pero las variables definidas aquí pueden ser utilizadas en cualquier momento. En cambio, aquellas variables que se declaren dentro de una función sólo podrán ser empleadas en dicha función. Los tipos de variables más utilizados son: *int, long* y *float*. En nuestro caso también usamos otro tipo de variables específicos que forman parte de las librerías empleadas como *VL53L0X* o *Servo.*
- Configuración de Arduino - void setup(): En esta sección del código se ha de especificar toda la configuración del programa, así como su estado inicial. Se concreta qué pines funcionan como entradas o salida y si son analógicas o digitales, si se establece o no una conexión por el puerto serie y a qué velocidad, perfil de medición de los sensores, etc.
- Bucle principal - void loop(): En este último apartado del programa se definen todas las instrucciones, comandos y funciones necesarias que hacen que, finalmente, Arduino realice las acciones que nosotros deseamos. Se trata del cuerpo principal del programa, un bucle infinito que se ejecuta continuamente mientras Arduino reciba alimentación.

Arduino utiliza un subconjunto del conocido lenguaje de programación C, un lenguaje de programación basado en wiring. Esto facilita su uso, puesto que es un lenguaje estándar en la mayoría de las plataformas y dispositivos. Los programas de Arduino están compuestos por un solo fichero de extensión .ino, que ya incluye las librerías necesarias para su correcto funcionamiento.

CÓDIGO FINAL

Tras ir probando y modificando las distintas opciones de las que partimos para llevar a cabo este proyecto se ha llegado a un código final con el que se pretende minimizar los tiempos y optimizar el programa.

Los cambios que se han realizado con respecto a otras versiones anteriores, como las que se pueden ver en los apartados de caracterización (ver apartado 3.2.2), es que las medidas se toman en su totalidad cuando el motor gira en una dirección, en concreto en sentido horario, para después hacer el giro en la otra dirección sin paradas. Con esto se consigue una reducción de tiempos de 500 ms aproximadamente, ya que al volver sin paradas se pierde mucho menos tiempo en el giro, y al ir con Paradas más pequeñas también tendremos que esperar menos a que el motor termine su giro.

Por lo demás el código final queda como se muestra a continuación, gran parte del cual ya está explicado en el apartado 3.1 y 3.2.2. y también se ha ido explicando sobre el código con comentarios.

```
// Iniacilización

#include <Wire.h>
#include <VL53L0X.h>
#include <Servo.h>

#define XSHUT_pin2 6
#define XSHUT_pin1 5

//ADDRESS_DEFAULT 0b0101001 OR 41
//#define Sensor1_newAddress 41 no requiere cambio de dirección
#define Sensor2_newAddress 42

int Dist_1;
int Dist_2;
int Angulo;
unsigned long inicio, fin, transcurrido;

VL53L0X Sensor1;
VL53L0X Sensor2;

Servo Servo1;  //creamos un objeto servo

void setup() {

  pinMode (XSHUT_pin1, OUTPUT);
  pinMode (XSHUT_pin2, OUTPUT);

  Servo1.attach(9);  // asignamos el pin 9 al servo.

  Serial.begin(9600); //Se inicia la cunicación serie con el sensor

  Wire.begin();
  Wire.setClock(50000);

  //Cambiar la dirección de los sensores
  pinMode(XSHUT_pin2, INPUT);
  delay(10);
  Sensor2.setAddress(Sensor2_newAddress);
  pinMode(XSHUT_pin1, INPUT);
  delay(10);

  Sensor1.init();
  Sensor2.init();
```

```
  Sensor1.setTimeout(500);
  Sensor2.setTimeout(500);

  // Ya que es Long Range configuramos el sensor de la siguiente manera:

  Sensor1.setSignalRateLimit(0.1);
  Sensor2.setSignalRateLimit(0.1);
  Sensor1.setVcselPulsePeriod(VL53L0X::VcselPeriodPreRange, 18);
  Sensor1.setVcselPulsePeriod(VL53L0X::VcselPeriodFinalRange, 14);
  Sensor2.setVcselPulsePeriod(VL53L0X::VcselPeriodPreRange, 18);
  Sensor2.setVcselPulsePeriod(VL53L0X::VcselPeriodFinalRange, 14);
}
void loop() {
  inicio = millis();
  //Barremos los 180° con 12 paradas, es decir con medidas cada 15°
  //Incrementamos el angulo de 15° en 15° para recorrer de 0° a 180°
  for (Angulo = 0; Angulo < 180 ; Angulo += 15)
  {
    Servo1.write(Angulo); //Decirle al Servo que que varíe el ángulo
    delay(100);
    Dist_1 = Sensor1.readRangeSingleMillimeters();
    Dist_2 = Sensor2.readRangeSingleMillimeters();

    //Imprimimos en el puerto serie los datos obtenidos con el formato:("Angulo,Dist_sensor1,Dist_sensor2")

    Serial.print(Angulo);
    Serial.print(',');
    Serial.print(Dist_1);
    if (Sensor1.timeoutOccurred()) Serial.print(" TIMEOUT");
    Serial.print(',');
    Serial.print(Dist_2);
    if (Sensor2.timeoutOccurred()) Serial.print(" TIMEOUT");
    Serial.println();


  }
  Angulo = 0;
  Servo1.write(Angulo);
  delay(850);
  fin = millis();
  transcurrido = fin - inicio; //Calcula el tiempo que tarda en barrer los 360°
  //  Serial.println(transcurrido);
}
```

2.3.2 Processing [11].

Processing es un software flexible y un lenguaje que nos permite aprender a codificar en el contexto de las artes visuales basado en lenguaje Java. Desde 2001 Processing ha promovido la alfabetización del software dentro de las artes visuales y la alfabetización visual dentro de la tecnología. Hay decenas de miles de estudiantes, artistas, diseñadores, investigadores y aficionados que usan Processing para aprender y crear prototipos. Además, es un software gratuito y open source que nos permite crear programas interactivos con salida 2D, 3D, PDF o txt. Dispone también de numerosas librerías con las que se puede ampliar su software central y está muy bien documentado y tiene una amplia comunidad de foros que facilita mucho el desarrollo de aplicaciones para principiantes. Por todo esto se ha elegido este software para desarrollar nuestro proyecto.

El entorno de desarrollo que nos proporciona hace que sea más fácil la escritura de programas. Éstos se escriben en el editor de texto y se inician presionando el botón Ejecutar. En Processing a los programas se les llama scketch, al igual que en Arduino. Estos sketches se almacenan en el Sketchbook, que es una carpeta en su ordenador.

Los sketches pueden dibujar gráficos bidimensionales y tridimensionales, aunque lo predeterminado es para dibujar gráficos bidimensionales. Las opciones que nos brinda este software se amplían con librerías y herramientas. Las librerías hacen posible que los sketches tengan funcionalidades más allá del código central. Hay cientos de librerías aportadas por la comunidad de Processing que se pueden agregar a los scketches para permitir cosas nuevas como reproducir sonidos o trabajar con geometría 3D avanzada. Las herramientas amplían el PDE (Processing Development Environment) para facilitar la creación de scketches al proporcionar interfaces para tareas como la selección de colores.

Processing ofrece diferentes modos de programación para que sea posible implementar sketches en diferentes plataformas y programas de diferentes maneras. El modo Java es el predeterminado.

El entorno de desarrollo de Processing (PDE) consiste en un editor de texto simple para escribir código, un área de mensaje, una consola de texto, pestañas para administrar archivos, una barra de herramientas con botones para las acciones más comunes y una serie de menús. Las opciones de menú cambian de modo a modo. El modo Java predeterminado está documentado aquí.

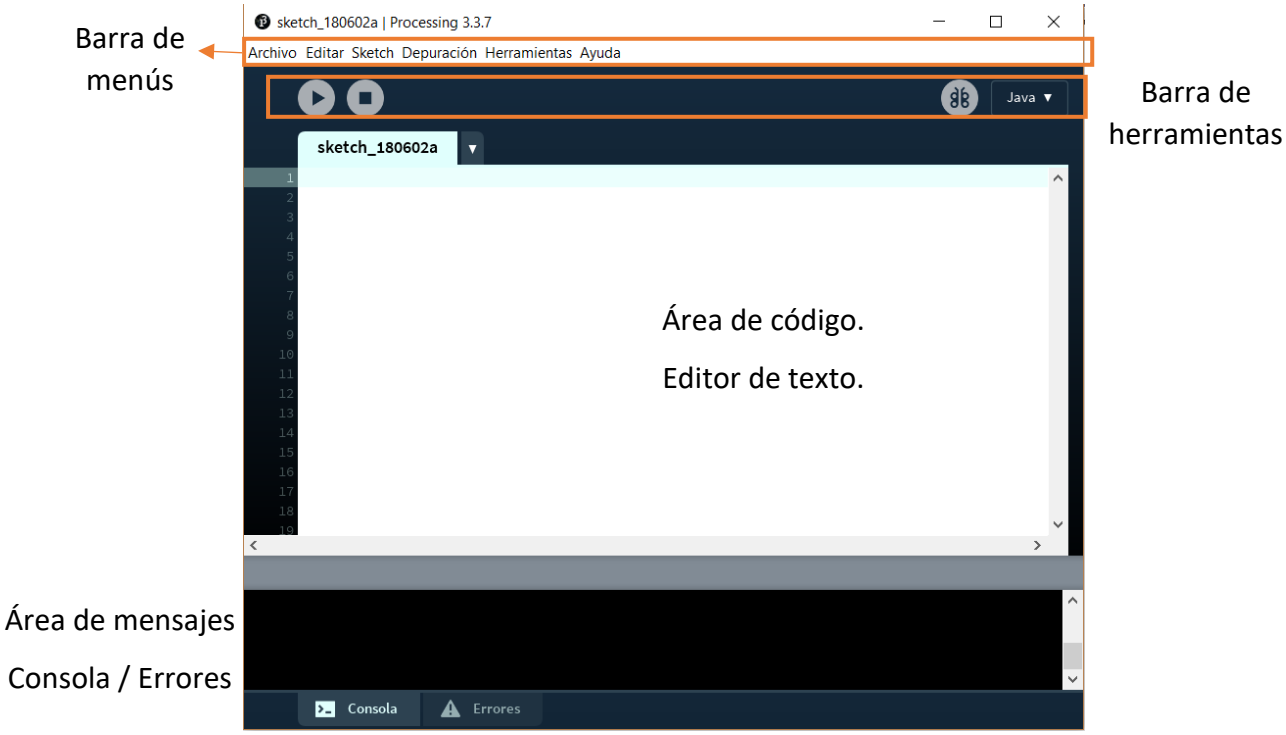


Figura 25: Interfaz del entorno de desarrollo de Processing.

Los diferentes sketches están escritos en el editor de texto. Este tiene características para cortar / pegar y para buscar / reemplazar texto. El área de mensajes muestra comentarios al guardar y exportar y también muestra los errores que se produzcan. La consola muestra la salida de texto, incluidos mensajes de error completos y salida de texto de los sketches con las funciones print () y println (). (Tenga en cuenta que la

consola funciona bien para mensajes ocasionales, pero no para salida de alta velocidad en tiempo real ya que no nos sería de utilidad al no poder ver los valores claramente).

Los botones en la barra de herramientas pueden ejecutar y detener programas y estos son los siguientes:

Ejecutar: Ejecuta el boceto. En modo Java, compila el código y abre una nueva ventana de visualización.

Detener: Termina un boceto en ejecución.

Se pueden encontrar comandos adicionales dentro de los seis menús: Archivo, Edición, Boceto, Depuración, Herramientas, Ayuda. Los menús son contextuales, lo que significa que solo están disponibles los elementos relevantes para el trabajo que se está llevando a cabo actualmente. Un resumen de lo más importante que estos nos ofrecen son:

- Archivo: dentro de este menú podemos encontrar diferentes opciones tales como: nuevo proyecto, abrir uno ya existente, guardar el actual, abrir Sketchbook etc. Una opción interesante es, al igual que en Arduino IDE, la de Ejemplos, donde podemos encontrar ejemplos sobre el uso de este software.
- Editar: en este menú aparecen las opciones básicas de edición como pueden ser: copiar, cortar, pegar, comentar / descomentar, autoformato, etc.
- Sketch: en este menú tenemos todas las opciones referentes a las acciones que podemos llevar a cabo con el sketch actual, como pueden ser: ejecutar, presentar, Tweak (que es una opción interesante ya que nos permite cambiar algunos valores de color y variable mientras se está ejecutando el código), importar biblioteca, etc.
- Depuración: Este menú nos permite depurar el código, es decir, podemos analizar lo que ocurre mientras se está ejecutando para detectar posibles errores. Cuando se quieren realizar aplicaciones complejas tener esta posibilidad es fundamental. Podemos analizar el código desde un punto, saltando línea a línea y ver cómo varían las variables y así saber exactamente qué está haciendo cada línea del texto.
- Herramientas: hay varias las opciones que proporciona Processing dentro de este menú, tales como: crear fuente (nos permite agregar una fuente al sketch actual y configurarla), selector de color, movie maker, etc.

Además, el entorno de desarrollo de Processing (PDE) es altamente configurable. Las preferencias más comunes se pueden modificar en la ventana Preferencias, ubicada en el menú Archivo en Windows y Linux y en el menú Processing en Mac Os X.

Como última nota a tener en cuenta sobre este programa hay que decir que Processing usa un sistema de coordenadas cartesianas con el origen en la esquina superior izquierda. Si su boceto tiene, por ejemplo, 320 píxeles de ancho y 240 píxeles de alto, la coordenada (0, 0) es el píxel superior izquierdo y las coordenadas (320, 240) se

encuentran en la esquina inferior derecha. El último píxel visible en la esquina inferior derecha de la pantalla está en la posición (319, 239) porque los píxeles se dibujan a la derecha y debajo de la coordenada. En nuestro caso desplazaremos este origen de coordenadas al centro de la pantalla para poder trabajar más fácilmente con radiales y distancias.

CÓDIGO FINAL

El código final que se ha realizado es el que se ha ido desarrollando para las caracterizaciones, un primer boceto muy sencillo de lo que se quería obtener se realizó para la caracterización de los sensores, dicho código se puede ver en el apartado 3.1.. A partir de esa base se evolucionó hasta conseguir dibujar los objetos que se encuentran en el entorno del dron con el código realizado para la caracterización del sistema completo, apartado 3.2.2., que posteriormente se perfeccionó añadiéndole un pequeño filtrado para reducir el error de medida, apartado 3.3.. Aunque no hemos cambiado nada más del código se muestra aquí, aunque sin explicaciones, en caso de no comprender algo mirar los apartados 3.3.2. y 3.3. ya que en estos sí que se ha explicado el código paro por paso.

```
import processing.serial.*; //Importamos la librería Serial

Serial port; //Nombre del puerto serie

int xPos = 1;         // horizontal position of the graph
float x1 = 0;
float x2=0;
float y1=0;
float y2 = height / 2;
int iteraciones=0;
PrintWriter output;  //Para crear el archivo de texto donde guardar los datos


String  dist1, dist2, angulo, data;//Valor de la distancia
int anguloi, indice;
int index1=0;
int index2=0;
float pixsDistance, pixsDistance2, pixsDistanceold, pixsDistance2old;
int[] dist1i={8000,8000,8000,8000,8000,8000,8000,8000,8000,8000,8000,8000};
int[] dist2i={8000,8000,8000,8000,8000,8000,8000,8000,8000,8000,8000,8000};
int[] dist1old={8000,8000,8000,8000,8000,8000,8000,8000,8000,8000,8000,8000};
int[] dist2old={8000,8000,8000,8000,8000,8000,8000,8000,8000,8000,8000,8000};

void setup()
{
  size (960, 1080);//Creamos una ventana de 1920 píxeles de anchura por 1080píxeles de altura

  //println(Serial.list()); //Visualiza los puertos serie disponibles en la consola de abajo
  port = new Serial(this, Serial.list()[0], 9600); //Abre el puerto serie COM2

  //Creamos el archivo de texto, que es guardado en la carpeta del programa
  output = createWriter("distancia_datos.txt");

  background(0);//Fondo de color negro
}
```

```
void draw()
{
  fill(98, 245, 31); // color verde
  // Llamamos a las funciones para dibujar
  drawRadar();
  drawLine();
  drawObject();
  drawObjectOld();

  data=port.readStringUntil('\n'); //Lemos toda un linea de datos
  if (data!=null && data.length()>2 ) {

    data=data.substring(0, data.length()-2); //Eliminamos el \n del string
    index1=data.indexOf(','); //Buscamos la primera , (primer separador entre datos)
    index2=data.indexOf(',', index1+1); //Buscamos la segunda , (segundo separador entre datos)
    if (index1>0 && index2>0) {
      angulo=data.substring(0, index1); //Obtenemos el primer dato que nos da el string de datos
      dist1=data.substring(index1+1, index2); //Obtenemos el segundo dato
      dist2=data.substring(index2+1, data.length()); //Obtenemos el tercer dato

      //Los convertimos a numeros para poder operar con ellos
      anguloi=int(angulo);
      switch(anguloi) {
      case 0:
        indice=0;
        break;
      case 15:
        indice=1;
        break;
      case 30:
        indice=2;
        break;
      case 45:
        indice=3;
        break;
      case 60:
        indice=4;
        break;
      case 75:
        indice=5;
        break;
      case 90:
        indice=6;
        break;
      case 105:
        indice=7;
        break;
      case 120:
        indice=8;
        break;
      case 135:
        indice=9;
        break;
      case 150:
        indice=10;
        break;
      case 165:
        indice=11;
        break;
      }
      // Guardamos los datos antiguos para hacer un filtrado
      dist1old[indice]=dist1i[indice];
      dist2old[indice]=dist2i[indice];

      //Actualizamos los datos
      dist1i[indice]=int(dist1);
      dist2i[indice]=int(dist2);
```

```
    }

    if (angulo != null && dist1!=null && dist2!= null) {
      // Lo escribes en el .txt
      iteraciones=iteraciones+1;
      output.print(angulo+","+dist1+","+dist2+"\n");
      //println(angulo);
      //println(dist1);
      //println(dist2);
    }
  }
}
void drawRadar() {
  pushMatrix();
  translate(480, 500); // mueve el centro de coordenadas
  noFill();
  strokeWeight(2);
  stroke(98, 245, 31);
  // dibuja los arcos
  arc(0, 0, 900, 900, 0, TWO_PI);
  arc(0, 0, 700, 700, 0, TWO_PI);
  arc(0, 0, 500, 500, 0, TWO_PI);
  arc(0, 0, 300, 300, 0, TWO_PI);
  // dibuja los radiales
  line(-480, 0, 480, 0);
  line(0, 0, -480*cos(radians(30)), -480*sin(radians(30)));
  line(0, 0, -480*cos(radians(60)), -480*sin(radians(60)));
  line(0, 0, -480*cos(radians(90)), -480*sin(radians(90)));
  line(0, 0, -480*cos(radians(120)), -480*sin(radians(120)));
  line(0, 0, -480*cos(radians(150)), -480*sin(radians(150)));
  line(0, 0, -480*cos(radians(180)), -480*sin(radians(180)));
  line(0, 0, -480*cos(radians(210)), -480*sin(radians(210)));
  line(0, 0, -480*cos(radians(240)), -480*sin(radians(240)));
  line(0, 0, -480*cos(radians(270)), -480*sin(radians(270)));
  line(0, 0, -480*cos(radians(300)), -480*sin(radians(300)));
  line(0, 0, -480*cos(radians(330)), -480*sin(radians(330)));
  line(-480*cos(radians(30)), 0, 480, 0);
  popMatrix();
}
void drawLine() {
  pushMatrix();
  strokeWeight(9);
  stroke(30, 250, 60);
  translate(480, 500);
  line(0, 0, 480*cos(radians(anguloi)), -480*sin(radians(anguloi)));
  line(0, 0, 480*cos(radians(anguloi+180)), -480*sin(radians(anguloi+180)));
  popMatrix();
}
void drawObject() {
  pushMatrix();
  translate(480, 500);
  strokeWeight(9);
  stroke(255, 10, 10); // color rojo
  pixsDistance = dist1i[indice]*0.2375; // convierte la distancia obtenida a pixeles en el dibujo
  if (10< dist1i[indice] && dist1i[indice]<2000) {
    // dibuja el objeto de acuerdo con el angulo y la distancia en la que se encuentra
    line(pixsDistance*cos(radians(anguloi)), -pixsDistance*sin(radians(anguloi)), ...
    480*cos(radians(anguloi)), -480*sin(radians(anguloi)));
  }
  pixsDistance2 = dist2i[indice]*0.2375;
  //println(pixsDistance2+","+dist2i+","+dist2);
  if (10<dist2i[indice] &&  dist2i[indice]<2000) {
    line(pixsDistance2*cos(radians(anguloi+180)), -pixsDistance2*sin(radians(anguloi+180)), ...
    480*cos(radians(anguloi+180)), -480*sin(radians(anguloi+180)));
  }
  popMatrix();
}
```

```
void drawObjectOld() {
  pushMatrix();
  translate(480, 500);
  strokeWeight(9);
  stroke(300, 10, 100); // color rojo flojito
  pixsDistanceold = dist1old[indice]*0.2375; // convierte la distancia obtenida a pixeles en el dibujo
  if (dist1i[indice]>2000 && 10< dist1old[indice] && dist1old[indice]<2000) {
    // dibuja el objeto donde se encontraba anteriormente aunque en esta medida no esté
    line(pixsDistanceold*cos(radians(anguloi)), -pixsDistanceold*sin(radians(anguloi)), ...
    480*cos(radians(anguloi)), -480*sin(radians(anguloi)));
  }
  pixsDistance2old = dist2old[indice]*0.2375;
  if (dist2i[indice]>=2000 && 10< dist2old[indice] && dist2old[indice] < 2000) {
    line(pixsDistance2old*cos(radians(anguloi+180)), -pixsDistance2old*sin(radians(anguloi+180)), ...
    480*cos(radians(anguloi+180)), -480*sin(radians(anguloi+180)));
  }
  popMatrix();
}
void keyPressed() //Cuando se pulsa una tecla
{
  //Pulsar la tecla E para salir del programa
  if (key=='e' || key=='E')
  {
    output.flush(); // Escribe los datos restantes en el archivo
    output.close(); // Final del archivo
    exit();//Salimos del programa
  }
}
```

3. Caracterización

En este apartado se muestra la caracterización realizada tanto de los sensores láser VL53L0X individualmente como del sistema en su totalidad. Dicha caracterización es necesaria para conocer cómo funciona nuestro sistema de detección de obstáculos y detectar errores con la finalidad de poder corregirlos. Para poder validar este sistema debe haberse comprobado el buen funcionamiento de sus elementos, tanto por separado como ya integrados trabajando conjuntamente.

3.1. Caracterización sensores VL53L0X

En este proyecto se está trabajando con dos sensores laser VL53L0X, para caracterizarlos se ha tenido en cuenta los distintos perfiles en los que se puede usar dicho sensor según se describe en su *Datasheet*. Debido a nuestras necesidades se han hecho pruebas tanto para "High speed" como para "Long Range" y en el mejor de los casos, es decir, midiendo distancia a un objeto blanco, y en el peor, midiendo distancia a uno negro.

Range Profile	Range timing budget	Typical performance	Typical application
Default mode	30ms	1.2m, accuracy as per *Table 12*	standard
High accuracy	200ms	1.2m, accuracy < +/- 3%	precise measurement
Long range	33ms	2m, accuracy as per *Table 12*	long ranging, only for dark conditions (no IR)
High speed	20ms	1.2m, accuracy +/- 5%	high speed where accuracy is not priority

Tabla 5: Perfiles de medición proporcionados por la API.

Para tomar las medidas necesarias y poder pasar los datos desde Arduino hasta Excel nos hemos ayudado de Processing. Para empezar, se ha hecho un programa de Arduino que nos envía los datos medidos del sensor mediante el puerto serie haciendo una espera para darle tiempo a Processing a medir los datos ya que si no podrían llegar a solaparse una medida con la siguiente. Posteriormente mediante Processing se leerá el valor que nos envía Arduino desde el puerto serie de la placa y lo almacenaremos en un fichero de texto. Además, se va a graficar los datos que nos están llegando para poder detectar un error lo antes posible.

Se le añade una opción de salida rápida del programa, antes de que éste acabe, pulsando la tela "E" para poder parar la recepción de datos en cuanto se detecte un error sin necesidad de acabar el programa. Se ha configurado para que tome 150 datos cada vez que se inicie una medida. La gráfica que nos muestra este programa es la mostrada en la Figura 26.

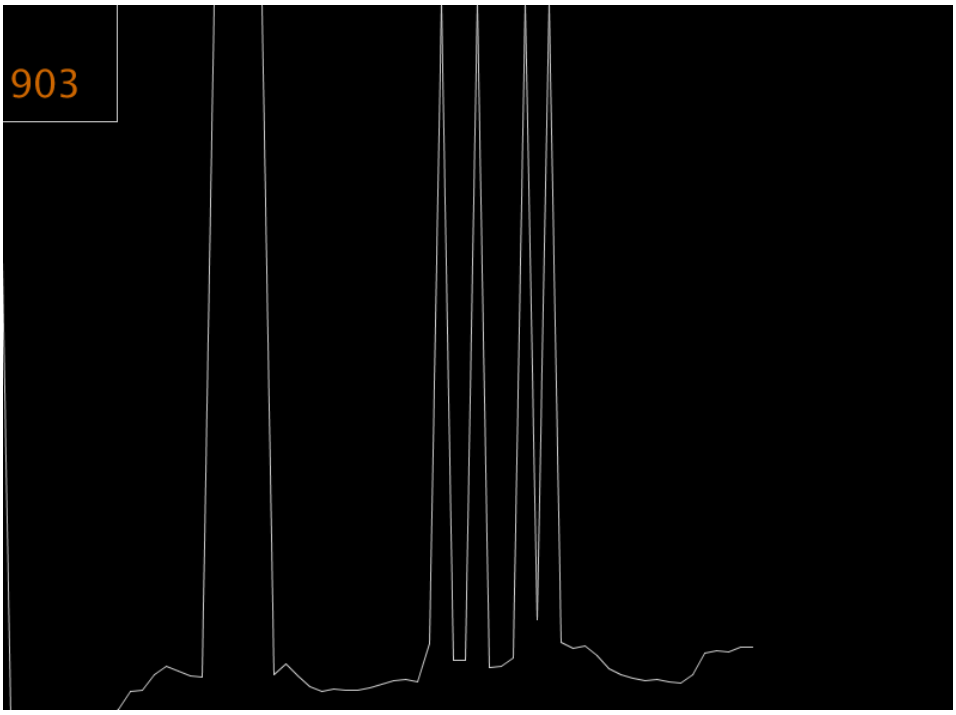


Figura 26: Ejemplo ejecución programa en Processing (lectura de datos).

Por último, se abrirá el documento de texto en formato .txt creado por Processing con Excel y una vez tenemos los datos tabulados ya podemos trabajar con ellos. A continuación, se mostrará cómo se han programado estos procesos:

ARDUINO:

```
// Estoy usando Long Range que llega mas lejos pero tarda más en medir
int Dist;

#include <Wire.h>
#include <VL53L0X.h>

#define XSHUT_pin1 5

VL53L0X Sensor1;

void setup() {
  pinMode (XSHUT_pin1,OUTPUT);

  Serial.begin(9600); //Se inicia la comunicación serie con el sensor a 9600 baudios.

  Wire.begin();
  Wire.setClock(50000);

  //Cambiar la dirección de los sensores

  pinMode(XSHUT_pin1, INPUT);
  delay(10);

  Sensor1.init();

  Sensor1.setTimeout(500);

// Ya que es Long Range configuramos el sensor de la siguiente manera:

  Sensor1.setSignalRateLimit(0.1);

  Sensor1.setVcselPulsePeriod(VL53L0X::VcselPeriodPreRange, 18);
  Sensor1.setVcselPulsePeriod(VL53L0X::VcselPeriodFinalRange, 14);


}

void loop() {
  Dist =Sensor1.readRangeSingleMillimeters();
  Serial.println(Dist);  //Enviamos los datos
  delay(200);            // Esperamos a que Processing los recoja y los guarde

}
```

PROCESSING:

```
import processing.serial.*; //Importamos la librería Serial

Serial port; //Nombre del puerto serie

int xPos = 1;         // horizontal position of the graph
float x1 = 0;
float x2=0;
float y1=0;
float y2 = height / 2;
int iteraciones=0;
PrintWriter output;  //Para crear el archivo de texto donde guardar los datos


String  valor;//Valor de la distancia

void setup()
{
  size(800, 600); //Creamos una ventana de 800 píxeles de anchura por 600 píxeles de altura

  println(Serial.list()); //Visualiza los puertos serie disponibles en la consola de abajo
  port = new Serial(this, Serial.list()[0], 9600); //Abre el puerto serie COM2

  output = createWriter("distancia_datos.txt"); //Creamos el archivo de texto, que es guardado en la carpeta del programa


  background(0);//Fondo de color negro
  textSize(32);
}


void draw()
{
  valor = port.readStringUntil('\n');

  if (valor != null) {
    // Lo escribes en el .txt
    iteraciones=iteraciones+1;
    output.print(valor);
    println(valor);
    // eliminar cualquier espacio en blanco:
    valor = trim(valor);
    // Convertir en float para poder usar la función map
    float inByte = float(valor);

    fill(0);
    rect(0, 0, 100, 100);
    fill(204, 102, 0);
    text(valor, 10, 80);
    inByte = map(inByte, 0, 8191, 0, height);

    // draw the line:
    stroke(204);
    line(xPos - 10, y1, xPos, y2);

    y1 = y2;
    y2 = height - inByte;

    // cuando llega al borde de la gráfica vuelve a empezar
    if (xPos >= width) {
      xPos = 0;
      background(0);
    } else {
      // incrementar la posición horizaontal:
      xPos+=10;
    }
  }
  if (iteraciones >150) {
    output.flush(); // Escribe los datos restantes en el archivo
```

```
    output.close(); // Final del archivo
    exit();//Salimos del programa
  }
}
void keyPressed() //Cuando se pulsa una tecla
{
  //Pulsar la tecla E para salir del programa
  if (key=='e' || key=='E')
  {
    output.flush(); // Escribe los datos restantes en el archivo
    output.close(); // Final del archivo
    exit();//Salimos del programa
  }
}
```

3.1.1. Estudio de los datos obtenidos:

Ahora vamos a analizar las medidas obtenidas a distintas distancias, en distintos entornos y tipos de medida de los sensores. Estas medidas se harán desde una distancia de 2 cm hasta 2.2 m, este margen superior depende de la situación. Analizaremos los datos devuelto por un obstáculo blanco y por uno negro, e intentaremos deducir cuál de todas las opciones que nos ofrece este sensor láser es la mejor y cómo detectar los errores y tratarlos.

La caracterización de un sensor se basa principalmente en hallar la ecuación característica de éste. Esta ecuación relaciona las medidas del sensor con la variable física real a medir. Además, una vez realizadas dichas medidas, se calculará mediante una tabla de Excel el promedio de éstas, su desviación típica, su varianza y el error cometido.

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

$$s^2 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n}$$

$$\sigma = \sqrt{s^2}$$

$$\varepsilon = \bar{x} - Valor\ real$$

Además, se efectuarán estos cálculos también para los datos una vez eliminado los valores medidos espurios, es decir los datos no válidos que podemos identificar cómo tal, en este caso hemos tomado como medida no válida todos los datos inferiores a 10 mm y los superiores a 8190 (este es el valor que el sensor devuelve cuando no detecta ningún objeto). Para la medida del error (última columna de las tablas), la media tomada será la que no tiene en cuenta los espurios ya que aspiramos a identificar estos datos y poder eliminarlos con un filtro. Por último, el porcentaje de medidas no válidas (%MNV) que se muestra en la penúltima columna es el porcentaje de datos espurios con respecto al total de valores medidos, es decir, que si hay un %MNV de un 18% quiere decir que cada 100 medidas 18 son datos no válidos que podemos identificar como tal.

OBJETO NEGRO:

High speed:

- Sensor 1:

Dist real(mm)	MEDIA DIST MEDIDA	Media sin espurios	Desv tipica	Desv tipica sin e	Varianza	Varianza s.e.	% MNV	Error
20	18,664	18,664	3,06368953	3,063689532	9,38619355	9,38619355	0,00%	-1,336
30	27,75471698	27,75471698	2,57965993	2,579659925	6,65464533	6,65464533	0,00%	-2,24528302
40	36,98571429	36,98571429	2,31661411	2,316614108	5,36670092	5,36670092	0,00%	-3,01428571
50	48,64516129	48,64516129	2,36572252	2,365722525	5,59664306	5,59664306	0,00%	-1,35483871
60	57,89130435	57,89130435	2,19117371	2,191173712	4,80124224	4,80124224	0,00%	-2,10869565
70	68,96835443	68,96835443	2,60675479	2,606754788	6,79517052	6,79517052	0,00%	-1,03164557
80	81,0472973	81,0472973	2,76354908	2,763549082	7,63720353	7,63720353	0,00%	1,0472973
90	93,84821429	93,84821429	2,65496245	2,65496245	7,04882561	7,04882561	0,00%	3,84821429
100	104,92	104,92	2,65973562	2,659735616	7,07419355	7,07419355	0,00%	4,92
200	206,8305085	208,5299145	19,0945936	4,901746082	364,603506	24,0271146	0,85%	8,52991453
300	309,5092025	309,5092025	8,76748208	8,767482076	76,868742	76,868742	0,00%	9,50920245
400	410,1340782	412,2640449	31,2762952	12,92538122	978,206641	167,06548	0,55%	12,2640449
500	526,6935484	505,1783784	294,10178	19,95335777	86495,8569	398,136486	0,54%	5,17837838
600	591,2542373	593,7319149	45,3471126	24,70182099	2056,36062	610,17996	0,43%	-6,26808511
700	736,5369128	690,8095238	618,261113	33,05461096	382246,804	1092,60731	1,35%	-9,19047619
800	1306,589552	751,3951613	1962,91707	47,8686921	3853043,43	2291,41168	7,52%	-48,6048387
900	2271,258278	864,1065574	2958,02973	672,1422125	8749939,87	451775,154	20,00%	-35,8934426
1000	3952,166667	943,6458333	3595,08989	76,63611615	12924671,3	5873,0943	40,70%	-56,3541667
1200	5896,629808	1083,348485	3337,49844	113,0245028	11138895,8	12774,5382	66,82%	-116,651515
1400	7773,084746	1154,6	1669,0705	82,60024213	2785796,33	6822,8	95,73%	-245,4

Tabla 6: Caracterización sensor 1 Objeto negro configurado en High Speed.

- Sensor 2:

Dist real(mm)	MEDIA DIST MEDIDA	Media sin espurios	Desv tipica	Desv tipica sin e	Varianza	Varianza s.e.	%MNV	Error
20	14,28037383	14,28037383	3,080046534	3,080046534	9,48668665	9,48668665	0,00%	-5,71962617
30	25,96835443	25,96835443	2,626229493	2,626229493	6,89708135	6,89708135	0,00%	-4,03164557
40	35,64705882	35,64705882	2,792757417	2,792757417	7,79949399	7,79949399	0,00%	-4,35294118
50	46,31216931	46,31216931	2,381870812	2,381870812	5,67330857	5,67330857	0,00%	-3,68783069
60	55,78017241	55,78017241	2,643122901	2,643122901	6,98609867	6,98609867	0,00%	-4,21982759
70	67,88	67,88	2,643122901	2,643122901	6,89825503	6,89825503	0,00%	-2,12
80	78,94977169	78,94977169	2,496279331	2,496279331	6,2314105	6,2314105	0,00%	-1,05022831
90	92,609375	92,609375	2,654348526	2,654348526	7,0455661	7,0455661	0,00%	2,609375
100	104,7132867	104,7132867	2,590944082	2,590944082	6,71299123	6,71299123	0,00%	4,71328671
200	207,1166667	208,2569832	16,10901236	5,058188806	259,500279	25,585274	0,56%	8,25698324
300	308,4054054	310,4761905	26,59006971	8,53705327	707,031807	72,8812785	0,68%	10,4761905
400	395,2606635	399,5288462	53,67666264	39,54095455	2881,18411	1563,48709	1,43%	-0,47115385
500	494,0542986	496,2909091	37,98452726	18,40728735	1442,82431	338,828227	0,45%	-3,70909091
600	554,6428571	556,6236559	41,60913386	25,1991695	1731,32002	634,998143	0,36%	-43,3763441
700	692,7027027	647,3957704	584,7276154	34,13632004	341906,384	1165,28835	0,60%	-52,6042296
800	999,0769231	684,1021898	1510,925158	41,73415245	2282894,83	1741,73948	4,23%	-115,89781
900	1661,037879	823,8632479	2434,988933	689,2934652	5929171,11	475125,481	11,45%	-76,1367521
1000	3676,269737	890,6382979	3558,626731	84,56210431	12663824,2	7150,74949	38,41%	-109,361702
1200	5013,80102	950,0697674	3604,278535	177,5269849	12990823,8	31515,8304	56,41%	-249,930233
1400	7890,228571	1185	1424,068608	98,61034428	2027971,4	9724	95,68%	-215

Tabla 7. Caracterización sensor 2 Objeto negro configurado en High Speed.

Inicialmente este tipo de medida nos ha parecido interesante ya que es la que puede darnos la distancia en el menor tiempo posible cosa que nos puede ser muy útil ya que debemos tomar muchas medidas a distintos ángulos y queremos saber la localización de los obstáculos prácticamente en tiempo real, dentro de lo que sea posible. Pero como podemos ver a partir de un metro de distancia la medida empieza a ser muy pobre y

además conforme aumentamos la distancia el %MNV crece de forma muy notable y conseguir medir una distancia a más allá del metro es prácticamente imposible independientemente del filtro que se le pueda meter. Además, teniendo en cuenta los datos que nos da el fabricante, no podríamos pasar del 1.20 metros ni en el mejor de los casos por esto este tipo de medida programada en *High speed* quedará descartada ya que en un UAV en vuelo necesitamos detectar los posibles objetos interferentes lo antes posible para evitar colisiones y más teniendo en cuenta que son un vehículo en movimiento. Así la ventaja que inicialmente podría parecer que este tipo de medida ofrecía queda enmascarada por el gran porcentaje de medidas no válidas midiendo a largas distancias.

Long range:

- Sensor 1:

Dist real(mm)	MEDIA DIST MEDIDA	Media sin espurios	Desv tipica	Desv tipica s.e.	Varianza	Varianza s.e.	% MNV	Error
20	18,11	18,11	2,20	2,20	4,85	4,85	0,00%	-1,89
30	25,38	25,38	1,90	1,90	3,59	3,59	0,00%	-4,62
40	37,24	37,24	2,04	2,04	4,16	4,16	0,00%	-2,76
50	45,96	45,96	1,52	1,52	2,31	2,31	0,00%	-4,04
60	56,52	56,52	2,01	2,01	4,02	4,02	0,00%	-3,48
70	69,88	69,88	1,92	1,92	3,69	3,69	0,00%	-0,12
80	80,94	80,94	1,96	1,96	3,86	3,86	0,00%	0,94
90	92,18	92,18	1,73	1,73	2,98	2,98	0,00%	2,18
100	104,91	104,91	1,90	1,90	3,60	3,60	0,00%	4,91
200	204,79	207,45	23,26	3,78	540,96	14,29	1,34%	7,45
300	308,05	310,11	26,12	6,75	682,29	45,59	0,67%	10,11
400	402,27	397,84	48,01	61,35	2304,71	3763,89	1,34%	-2,16
500	507,70	507,70	15,99	15,99	255,80	255,80	0,00%	7,70
600	598,74	606,78	73,64	24,87	5422,37	618,65	1,34%	6,78
700	833,92	683,80	1054,76	28,40	1112526,42	806,33	2,01%	-16,20
800	1422,15	833,64	2092,09	631,93	4376837,47	399332,74	8,05%	33,64
900	2587,11	881,86	3101,70	50,99	9620520,44	2599,84	23,49%	-18,14
1000	3388,32	987,38	3407,16	59,38	11608744,27	3525,49	33,56%	-12,62
1200	5553,59	1127,80	3427,99	98,11	11751087,25	9624,82	63,09%	-72,20
1400	6467,45	1238,09	3023,50	167,30	9141579,81	27987,90	76,51%	-161,91
1600	6902,21	1289,61	2698,97	183,65	2698,97	33728,40	81,21%	-310,39
1800	7080,17	1440,37	2562,05	374,77	2562,05	140455,58	87,92%	-359,63
2000	7568,13	1927,25	2036,76	167,62	2036,76	28096,92	97,32%	-72,75

Tabla 8: Caracterización sensor 1 Objeto negro configurado en Long Range.

- Sensor 2:

Dist real(mm)	MEDIA DIST MEDIDA	Media sin espurios	Desv tipica	Desv tipica sin e	Varianza	Varianza s.e.	% MNV	Error
20	16,89	16,89	1,97	1,97	3,87	3,87	0,00%	-3,11
30	23,95	23,95	2,01	2,01	4,03	4,03	0,00%	-6,05
40	36,37	36,37	2,11	2,11	4,44	4,44	0,00%	-3,63
50	46,75	46,75	1,96	1,96	3,84	3,84	0,00%	-3,25
60	56,34	56,34	2,06	2,06	4,23	4,23	0,00%	-3,66
70	69,05	69,05	1,93	1,93	3,74	3,74	0,00%	-0,95
80	80,55	80,55	2,04	2,04	4,16	4,16	0,00%	0,55
90	93,13	93,13	1,96	1,96	3,83	3,83	0,00%	3,13
100	105,00	105,70	8,92	2,26	79,54	5,09	0,67%	5,70
200	208,96	208,96	3,82	3,82	14,62	14,62	0,00%	8,96
300	303,24	305,22	25,09	6,38	629,39	40,73	0,67%	5,22
400	410,53	410,53	10,90	10,90	118,72	118,72	0,00%	10,53
500	512,72	512,72	15,66	15,66	245,24	245,24	0,00%	12,72
600	607,89	611,95	53,55	19,95	2867,65	398,17	0,67%	11,95
700	849,91	704,89	1055,56	64,17	1114210,25	4118,15	2,68%	4,89
800	1255,61	812,99	1758,26	40,83	3091466,87	1667,05	6,04%	12,99
900	2467,87	916,07	2990,25	52,25	8941610,51	2730,05	21,48%	16,07
1000	3050,37	984,89	3269,47	55,50	10689407,54	3079,86	28,86%	-15,11
1200	5520,75	1286,47	3421,74	927,74	11708309,14	860701,48	61,74%	86,47
1400	6218,82	1321,64	3121,37	140,41	9742958,26	19716,14	71,81%	-78,36
1600	6692,99	1519,28	2836,13	183,63	8043644,95	33719,78	80,54%	-80,72
1800	7159,62	1510,25	2501,03	198,69	6255161,93	39478,57	91,95%	-289,75
2000	7751,33	1779,00	1751,55	#¡DIV/0!	3067942,55	#¡DIV/0!	99,33%	-221,00

Tabla 9:Caracterización sensor 2 Objeto negro configurado en Long Range.

En este caso también se tiene un %MNV considerable a partir del metro de distancia pero a partir de aquí este no crece de forma tan drástica como en el caso anterior, además en este caso la mejora al detectar un objeto blanco es muy considerable y, como veremos en los siguientes apartados, se puede detectar objetos claros hasta a 2 metros de distancia sin problema por lo que optaremos por este tipo de medida de los sensores ya que aunque estemos perdiendo 13 ms en cada medida la distancia a la que podemos detectar mejora de forma muy considerable.

OBJETO BLANCO:

Long Range:

- Sensor 1:

Dist real(mm)	MEDIA DIST MEDIDA	Media sin espurios	Desv tipica	Desv tipica s.e.	Varianza	Varianza s.e.	%MNV	Error
20,00	24,24	24,24	2,25	2,25	5,04	5,04	0,00%	4,24
30,00	34,65	34,65	2,35	2,35	5,50	5,50	0,00%	4,65
40,00	35,34	35,34	1,68	1,68	2,83	2,83	0,00%	-4,66
50,00	46,65	46,65	1,70	1,70	2,90	2,90	0,00%	-3,35
60,00	57,05	57,05	1,93	1,93	3,72	3,72	0,00%	-2,95
70,00	69,26	69,26	3,76	3,76	14,14	14,14	0,00%	-0,74
80,00	79,73	79,73	1,76	1,76	3,11	3,11	0,00%	-0,27
90,00	90,35	90,35	1,80	1,80	3,25	3,25	0,00%	0,35
100,00	100,99	100,99	1,86	1,86	3,48	3,48	0,00%	0,99
200,00	208,13	209,52	17,20	1,75	295,72	295,72	0,67%	9,52
300,00	313,67	313,67	2,23	2,23	4,99	4,99	0,00%	13,67
400,00	407,09	409,76	32,85	2,91	1079,15	8,50	0,67%	9,76
500,00	507,02	513,86	59,13	3,70	3496,95	13,69	1,34%	13,86
600,00	613,49	617,56	50,02	4,10	2501,92	16,77	0,67%	17,56
700,00	715,46	720,21	58,48	5,34	3419,54	28,56	0,67%	20,21
800,00	810,43	821,34	94,32	5,24	8896,31	27,44	1,34%	21,34
900,00	909,49	909,49	9,13	9,13	83,33	83,33	0,00%	9,49
1000,00	1014,52	1014,52	10,24	10,24	104,84	104,84	0,00%	14,52
1200,00	1220,25	1220,25	11,06	11,06	122,31	122,31	0,00%	20,25
1400,00	1402,70	1402,70	20,35	20,35	414,02	414,02	0,00%	2,70
1600,00	1595,82	1595,82	26,17	26,17	685,12	685,12	0,00%	-4,18
1800,00	1930,23	1802,48	897,73	29,68	805919,14	880,81	2,01%	2,48
2000,00	2848,41	2026,62	2162,71	546,13	4677321,87	298262,76	13,42%	26,62
2200,00	3781,53	2123,29	2713,31	60,92	7362028,40	3710,91	27,52%	-76,71

Tabla 10: Caracterización sensor 1 Objeto blanco configurado en Long Range.

- Sensor 2:

Dist real(mm)	MEDIA DIST MEDIDA	Media sin espurios	Desv tipica	Desv tipica s.e.	Varianza	Varianza s.e.	% MNV	Error
20	23,94	23,94	2,54	2,54	6,47	6,47	0,00%	3,94
30	35,73	35,73	2,32	2,32	5,39	5,39	0,00%	5,73
40	38,07	38,07	1,86	1,86	3,45	3,45	0,00%	-1,93
50	48,89	48,89	2,02	2,02	4,07	4,07	0,00%	-1,11
60	59,75	59,75	1,91	1,91	3,67	3,67	0,00%	-0,25
70	69,21	69,21	1,85	1,85	3,43	3,43	0,00%	-0,79
80	80,54	80,54	2,06	2,06	4,22	4,22	0,00%	0,54
90	92,75	92,75	1,91	1,91	3,67	3,67	0,00%	2,75
100	101,49	102,79	11,39	1,76	129,71	3,11	1,34%	2,79
200	211,24	211,24	1,96	1,96	3,85	3,85	0,00%	11,24
300	312,05	314,10	25,18	2,27	633,84	5,13	0,67%	14,10
400	416,73	419,50	34,06	3,04	1160,14	9,25	0,67%	19,50
500	519,86	523,33	42,62	3,47	1816,89	12,02	0,67%	23,33
600	615,03	619,05	49,49	4,97	2449,00	24,67	0,67%	19,05
700	716,30	716,30	6,38	6,38	40,75	40,75	0,00%	16,30
800	825,17	825,17	6,89	6,89	47,54	47,54	0,00%	25,17
900	915,83	921,93	75,13	8,03	5644,64	64,47	0,67%	21,93
1000	1021,56	1021,56	9,00	9,00	81,01	81,01	0,00%	21,56
1200	1210,01	1210,01	10,99	10,99	120,70	120,70	0,00%	10,01
1400	1405,61	1360,08	558,20	24,54	311583,33	602,03	0,67%	-39,92
1600	1580,29	1580,29	20,60	20,60	424,18	424,18	0,00%	-19,71
1800	1943,42	1772,28	1037,76	27,45	1076942,33	753,25	2,68%	-27,72
2000	2336,52	1962,89	1484,16	33,36	2202738,08	1112,78	6,04%	-37,11
2200	3837,56	2200,39	2723,68	581,28	7418440,77	337888,80	28%	0,39

Tabla 11:Caracterización sensor 2 Objeto blanco configurado en Long Range.

Podemos observar en las tablas que sobre fondo blanco las medidas son mucho más exactas y con un %MNV mucho menor. En esta tabla se observa también que en el corto alcance los errores dados son menores y que a partir de unos 40 cm se disparan, pero también hay que tener en cuenta que un error de 4 mm en una medida de 2 cm es mucho más notable que un error de 28 mm en una medida de 1.8 m, por ello este error es más crítico a cortas distancias ya que puede variarnos más la medida tomada. Se han graficado los resultados para poder analizarlos de forma más inmediata y obtener la ecuación de caracterización de los sensores para este caso, que sería el óptimo.

Figura 27: Recta de caracterización sensor 1.

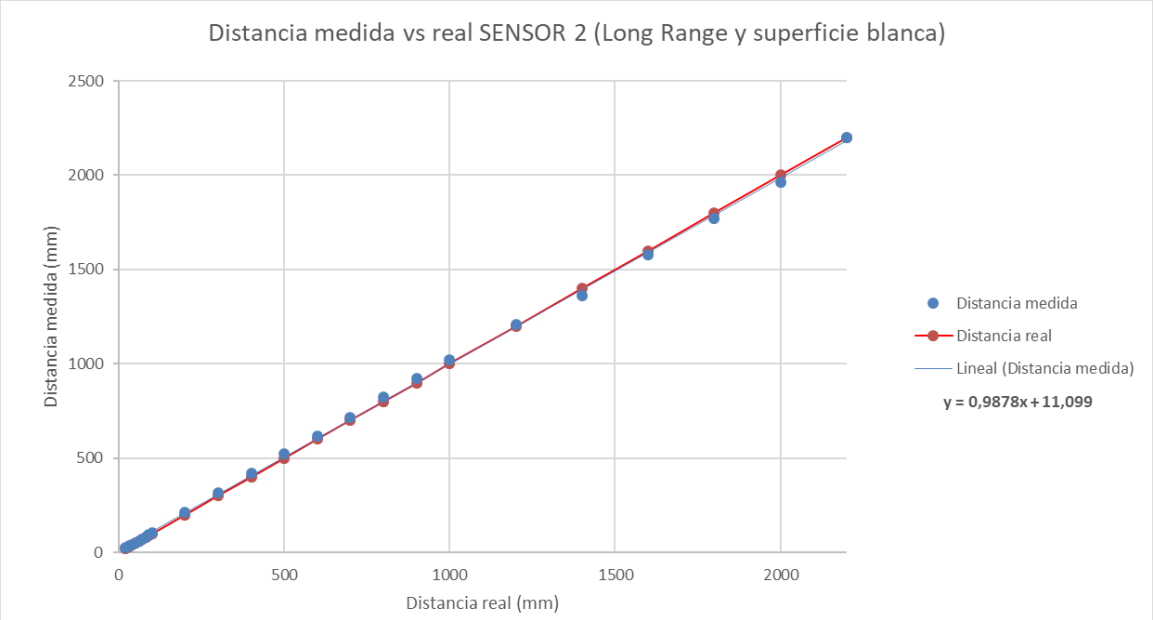


Figura 28: Recta de caracterización sensor 2.

Como se puede observar en este caso las medidas son casi perfectas, tal y como muestran las gráficas las líneas de tendencia de las medidas tomadas están prácticamente sobre la línea de las distancias reales. El %MNV es muy manejable hasta los 2 metros y aunque a partir de ahí se dispara se podría llegar incluso más lejos con un buen filtro. Además, la dispersión de los datos no es demasiado grande lo que ayuda a la hora de tratarlos y representarlos mediante una interfaz gráfica en Matlab, podemos ver que las desviaciones típicas son bastante bajas hasta el metro y medio y manejables hasta los dos metros.

3.1.2. Conclusiones:

Los datos obtenidos en el laboratorio son bastante parecidos a los que nos proporciona el fabricante, que podemos observar en la siguiente tabla.

Target reflectance level (Full FOV)	Conditions	Indoor (2)	Outdoor overcast (2)
White Target (88%)	Typical	200cm+ (1)	80cm
	Minimum	120cm	60cm
Grey Target (17%)	Typical	80cm	50cm
	Minimum	70cm	40cm

Tabla 12: Capacidades máximas de medida del VL53L0X.

3.2. Caracterización del sistema completo

En este apartado se va a tratar sobre la caracterización del sistema de detección de obstáculos ya conformado en su totalidad. Para ello se deben tomar medidas de ambos sensores con obstáculos a distintas distancias. Estas medidas serán recogidas en tablas para su posterior estudio. Para la realización de éstas se ha realizado un programa en Arduino que toma las medidas de los sensores y las envía, junto con el ángulo al que se encuentra el motor en el momento de la medición, por el puerto serie. Este programa servirá de base para el que se usará definitivamente en este proyecto, una vez se hayan ajustado correctamente las velocidades de medidas y los ángulos a los que éstas se deben tomar en este proceso de caracterización.

Por otro lado, volvemos a hacer uso del software Processing para recoger los datos que Arduino nos está enviando por el puerto serie. Con dicho software procederemos a tomar los datos de medidas de distancias y ángulos para posteriormente graficarlos y guardarlos en un archivo *.txt* que nos permitirá hacer un estudio de los datos obtenidos con Excel.

3.2.1. Montaje banco de pruebas

Para realizar esta caracterización se va a elaborar un banco de pruebas que consistirá en la construcción de paredes de cartulina que formarán círculos a distintas distancias a las que tomaremos medidas colocando el dron en su centro. Estos círculos se fabricarán con distintos radios, más concretamente con 50, 75, 100, 125, 150 y 175 cm de radio. Se tomarán 300 medidas para cada una de estas distancias y con los datos obtenidos podremos hacer la caracterización adecuada del sistema en su totalidad.

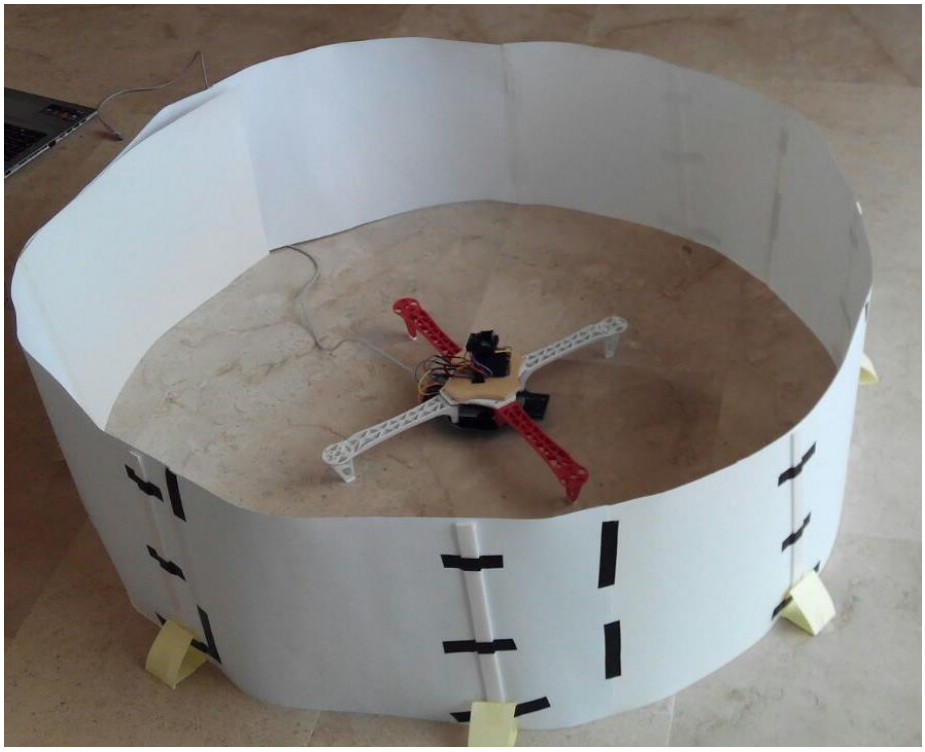


Figura 29: Círculo de 50 cm de radio.

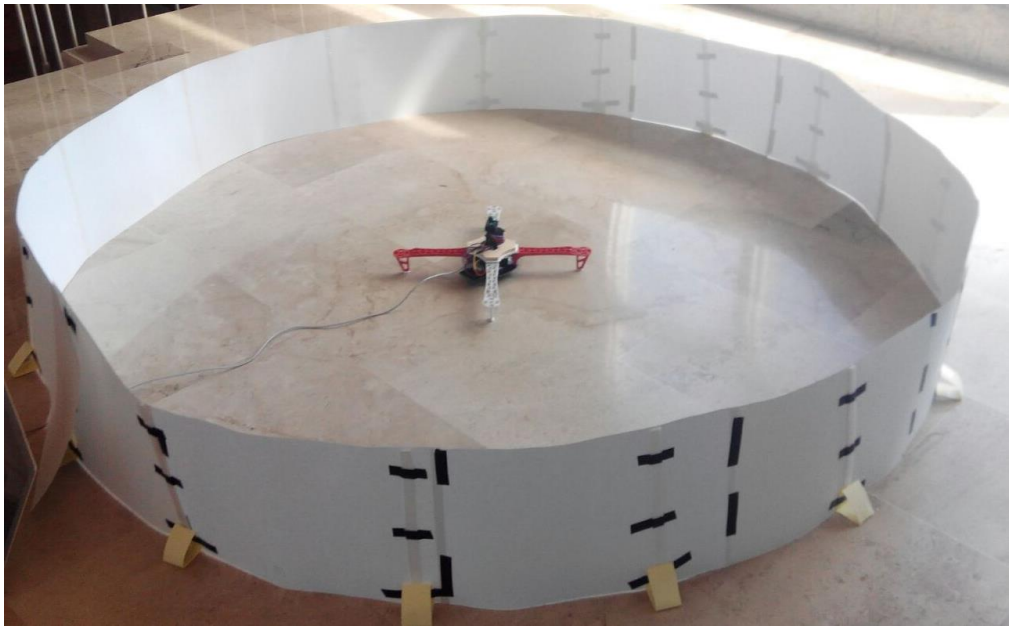


Figura 30: Círculo de 1 m de radio.

3.2.2. Programación

Arduino

Mediante la aplicación Arduino IDE podemos comunicarnos con la placa de Arduino y programarla con el fin de obtener los resultados deseados. En nuestro caso lo que debemos hacer es manejar un servo motor para que gire los grados que deseemos y en la dirección adecuada con el fin de poder barrer 180º e ir tomando medidas. Por otro lado, dichas medidas se deben tomar con dos sensores laser contrapuestos, es decir uno medirá la distancia en los radiales a los que se encuentra el servomotor, determinaremos su posición mediante la variable Angulo, y el otro medirá la distancia en los radiales suplementarios, es decir los que se encuentran en la posición Angulo+180º. En la Figura 31 se muestra un esquema de las direcciones de medida de cada uno de los sensores en función del ángulo en el que se encuentra el servo para su mejor comprensión. Por último, las variables Angulo y las dos distancias tomadas se deben enviar por el puerto serie para que el ordenador pueda identificarlas y trabajar con ellas.

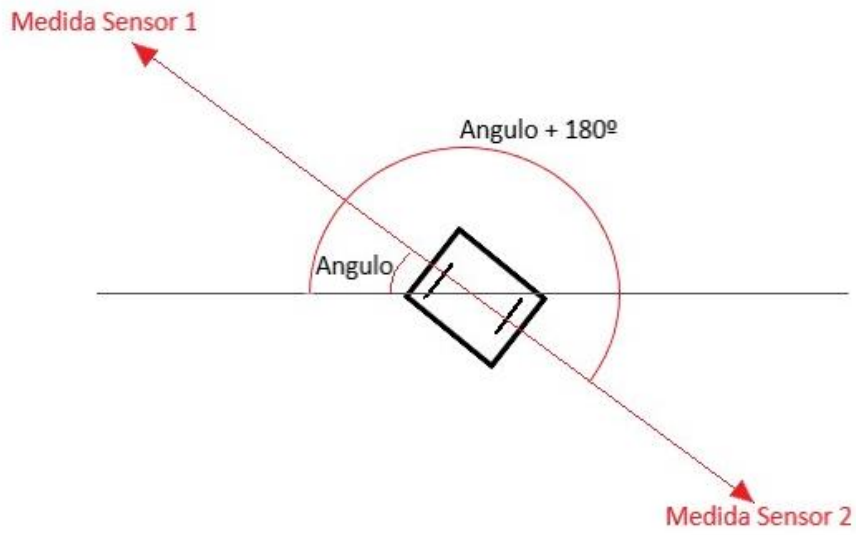


Figura 31: Radial medido por cada uno de los sensores en función de la variable Angulo.

Todo este proceso se hará con el siguiente programa que se irá explicando paulatinamente:

```
// Iniacilización

#include <Wire.h>
#include <VL53L0X.h>
#include <Servo.h>

#define XSHUT_pin2 6
#define XSHUT_pin1 5

//ADDRESS_DEFAULT 0b0101001 OR 41
//#define Sensor1_newAddress 41 no requiere cambio de dirección
#define Sensor2_newAddress 42

int Dist_1;
int Dist_2;
int Angulo;
unsigned long inicio, fin, transcurrido;

VL53L0X Sensor1;
VL53L0X Sensor2;

Servo Servo1;  //creamos un objeto servo

 void setup() {

   pinMode (XSHUT_pin1, OUTPUT);
   pinMode (XSHUT_pin2, OUTPUT);

   Servo1.attach(9);  // asignamos el pin 9 al servo.

   Serial.begin(9600); //Se inicia la cunicación serie con el sensor

   Wire.begin();
   Wire.setClock(50000);

   //Cambiar la dirección de los sensores

   pinMode(XSHUT_pin2, INPUT);
   delay(10);
   Sensor2.setAddress(Sensor2_newAddress);
   pinMode(XSHUT_pin1, INPUT);
   delay(10);

   Sensor1.init();
   Sensor2.init();

   Sensor1.setTimeout(500);
   Sensor2.setTimeout(500);

   // Ya que es Long Range configuramos el sensor de la siguiente manera:

   Sensor1.setSignalRateLimit(0.1);
   Sensor2.setSignalRateLimit(0.1);

   Sensor1.setVcselPulsePeriod(VL53L0X::VcselPeriodPreRange, 18);
   Sensor1.setVcselPulsePeriod(VL53L0X::VcselPeriodFinalRange, 14);
   Sensor2.setVcselPulsePeriod(VL53L0X::VcselPeriodPreRange, 18);
   Sensor2.setVcselPulsePeriod(VL53L0X::VcselPeriodFinalRange, 14);
 }
```

```
void loop() {
  //Barremos los 180° con 12 paradas, medimos la mitad al ir y la mitad al volver
  for (Angulo = 0; Angulo < 180 ; Angulo += 30) // Incrementamos el angulo de 30° en 30°
                                                // para recorrer de 0° a 180°
  {
    Servo1.write(Angulo); //Decirle al Servo que que varíe el ángulo
    delay(200);
    Dist_1 = Sensor1.readRangeSingleMillimeters();
    Dist_2 = Sensor2.readRangeSingleMillimeters();

    //Imprimimos en el puerto serie los datos obtenidos con el formato
    //("Angulo,Dist_sensor1,Dist_sensor2")

    Serial.print(Angulo);
    Serial.print(',');
    Serial.print(Dist_1);
    if (Sensor1.timeoutOccurred()) Serial.print(" TIMEOUT");
    Serial.print(',');
    Serial.print(Dist_2);
    if (Sensor2.timeoutOccurred()) Serial.print(" TIMEOUT");
    Serial.println();
  }

  for (Angulo = 165; Angulo > 0 ; Angulo -= 30) // Decrementamos el angulo de 30° en 30°
                                                // para recorrer de 165° a 15°
  {
    Servo1.write(Angulo); //Decirle al Servo que que varíe el ángulo
    delay(200);
    Dist_1 = Sensor1.readRangeSingleMillimeters();
    Dist_2 = Sensor2.readRangeSingleMillimeters();

    //Imprimimos en el puerto serie los datos obtenidos con el formato
    //("Angulo,Dist_sensor1,Dist_sensor2")

    Serial.print(Angulo);
    Serial.print(',');
    Serial.print(Dist_1);
    if (Sensor1.timeoutOccurred()) Serial.print(" TIMEOUT");
    Serial.print(',');
    Serial.print(Dist_2);
    if (Sensor2.timeoutOccurred()) Serial.print(" TIMEOUT");
    Serial.println();
  }
}
```

Podemos ver que la inicialización de los sensores es igual que la que hicimos en su caracterización y la única diferencia es la inicialización del servo motor que asignamos su señal al pin 9 de nuestra placa de Arduino.

Para realizar las medidas en la caracterización hacemos doce paradas, es decir, se hacen en radiales que están separados entre sí 15º, además hacemos la mitad de las medidas cuando el servo gira en una dirección y la otra mitad cuando gira en la otra. Para darle tiempo al servo a girar hasta el radial adecuado se debe hacer una espera que en este caso será de 200 ms. En un principio, siguiendo las indicaciones que nos da el fabricante en el datasheet del motor se había hecho un delay de tan solo 100 ms pero se detectaron errores en las medidas.

Por último, después de cada medida se deben enviar los datos por el puerto serie, y para ello elegimos el formato “Angulo, Distancia sensor 1, Distancia sensor 2 \n”. Esta forma en la que se envían los datos se debe tener en cuenta a la hora de adquirirlos en el programa Processing para que puedan ser interpretados correctamente.

PROCESSING

Mediante este software lo que se pretende es principalmente guardar los datos en un archivo .txt que después podamos abrir con Excel para hacer un estudio estadístico de los datos obtenidos, al igual que en la caracterización de los sensores individualmente. Para aprovechar las posibilidades que nos ofrece este software, que es principalmente un programa orientado a la creación de proyectos gráficos, dibujaremos los datos obtenidos pudiendo observar así gráficamente la distancia y la posición en la que se encuentran los obstáculos medidos. Además, así se podrá aprovechar este programa hecho inicialmente para la caracterización en posteriores etapas de este proyecto.

El código empleado para realizar los objetivos anteriormente descritos es el siguiente, que para una mejor comprensión se irá comentando sus funciones:

```
import processing.serial.*; //Importamos la librería Serial


Serial port; //Nombre del puerto serie

int xPos = 1;         // horizontal position of the graph
float x1 = 0;
float x2=0;
float y1=0;
float y2 = height / 2;
int iteraciones=0;
PrintWriter output;  //Para crear el archivo de texto donde guardar los datos

String  dist1, dist2, angulo, data;//Valor de la distancia
int anguloi, dist1i, dist2i;
int index1=0;
int index2=0;
float pixsDistance, pixsDistance2;

void setup()
{
  size (960, 1080);//Creamos una ventana de 960 píxeles de anchura por 1080píxeles de altura

  //println(Serial.list()); //Visualiza los puertos serie disponibles en la consola de abajo
  port = new Serial(this, Serial.list()[0], 9600); //Abre el puerto serie COM2

  output = createWriter("distancia_datos.txt"); //Creamos el archivo de texto


  background(0);//Fondo de color negro
}
```

Para comenzar debemos inicializar todas las variables y la sección setup() se creará la ventana gráfica, se abrirá el puerto serie con el que nos comunicaremos con la placa y se creará también el archivo *.txt* en el que se irán guardando las medidas tomadas.

```
void draw()
{
  fill(98, 245, 31); // color verde
  // Llamamos a las funciones para dibujar
  drawRadar();
  drawLine();
  drawObject();

  data=port.readStringUntil('\n'); //Lemos toda un linea de datos
  if (data!=null && data.length()>2 ) {

    data=data.substring(0, data.length()-2); //Eliminamos el \n del string
    index1=data.indexOf(','); //Buscamos la primera , (primer separador entre datos)
    index2=data.indexOf(',', index1+1); //Buscamos la segunda , (segundo separador entre datos)
    if (index1>0 && index2>0) {
      angulo=data.substring(0, index1); //Obtenemos el primer dato que nos da el string de datos
      dist1=data.substring(index1+1, index2); //Obtenemos el segundo dato que nos da el string de datos
      dist2=data.substring(index2+1, data.length()); //Obtenemos el tercer dato

      //Los convertimos a numeros para poder operar con ellos
      anguloi=int(angulo);
      dist1i=int(dist1);
      dist2i=int(dist2);
    }

    if (angulo != null && dist1!=null && dist2!= null) {
      // Lo escribes en el .txt
      iteraciones=iteraciones+1;
      output.print(angulo+","+dist1+","+dist2+"\n");
    }
  }
```

En esta otra función, draw(), comenzamos llamando al resto de funciones empleadas para dibujar. Posteriormente debemos obtener los datos que nos están llegando por el puerto serie, para ello lo primero que debemos hacer es guardar una línea entera que recibamos en la variable data y quitarle el “\n” de salto de línea. Después se buscarán las separaciones entre las variables que en nuestro caso están marcadas por una coma, por lo que con la función indexOf buscamos en qué posición de la cadena de caracteres data están las comas. Una vez tenemos delimitadas las variables dentro de la línea de datos leída se subdivide la cadena en las tres variables que estamos buscando mediante la función substring. Una vez ya tenemos las variables que se nos han enviado se pasan de carácter a entero para poder operar con ellas posteriormente y se guardan en el fichero.

```
void drawRadar() {
  pushMatrix();
  translate(480, 500); // mueve el centro de coordenadas
  noFill();
  strokeWeight(2);
  stroke(98, 245, 31);
  // dibuja los arcos
  arc(0, 0, 900, 900, 0, TWO_PI);
  arc(0, 0, 700, 700, 0, TWO_PI);
  arc(0, 0, 500, 500, 0, TWO_PI);
  arc(0, 0, 300, 300, 0, TWO_PI);
  // dibuja los radiales
  line(-480, 0, 480, 0);
  line(0, 0, -480*cos(radians(30)), -480*sin(radians(30)));
  line(0, 0, -480*cos(radians(60)), -480*sin(radians(60)));
  line(0, 0, -480*cos(radians(90)), -480*sin(radians(90)));
  line(0, 0, -480*cos(radians(120)), -480*sin(radians(120)));
  line(0, 0, -480*cos(radians(150)), -480*sin(radians(150)));
  line(0, 0, -480*cos(radians(180)), -480*sin(radians(180)));
  line(0, 0, -480*cos(radians(210)), -480*sin(radians(210)));
  line(0, 0, -480*cos(radians(240)), -480*sin(radians(240)));
  line(0, 0, -480*cos(radians(270)), -480*sin(radians(270)));
  line(0, 0, -480*cos(radians(300)), -480*sin(radians(300)));
  line(0, 0, -480*cos(radians(330)), -480*sin(radians(330)));
  line(-480*cos(radians(30)), 0, 480, 0);
  popMatrix();
}
```

Aquí lo que estamos dibujando es el entramado formado por radiales y arcos a distintos ángulos y distancias sobre los que se dibujarán nuestros datos, se ha elegido hacer cuatro arcos para darnos una idea de la distancia a la que se encuentra el obstáculo y radiales cada 30º en los 360º de circunferencia.

```
void drawLine() {
  pushMatrix();
  strokeWeight(9);
  stroke(30, 250, 60);
  translate(480, 500);
  line(0, 0, 480*cos(radians(anguloi)), -480*sin(radians(anguloi)));
  line(0, 0, 480*cos(radians(anguloi+180)), -480*sin(radians(anguloi+180)));
  popMatrix();
}
```

Se dibujarán líneas verdes en el ángulo en el que se está midiendo para eliminar posibles medidas de objetos en barridos anteriores y crear una base sobre la que dibujar nuevos objetos medidos.

```
void drawObject() {
  pushMatrix();
  translate(480, 500);
  strokeWeight(9);
  stroke(255, 10, 10); // color rojo
  pixsDistance = dist1i*0.2375; // convierte la distancia obtenida a pixeles en el dibujo(rango de 2 m)
  if (dist1i<2000) {
    // dibuja el objeto de acuerdo con el angulo y la distancia en la que se encuentra
    line(pixsDistance*cos(radians(anguloi)), -pixsDistance*sin(radians(anguloi)), ...
    480*cos(radians(anguloi)), -480*sin(radians(anguloi)));
  }
  pixsDistance2 = dist2i*0.2375;
  if (dist2i<2000) {
    line(pixsDistance2*cos(radians(anguloi+180)), -pixsDistance2*sin(radians(anguloi+180)), ...
    480*cos(radians(anguloi+180)), -480*sin(radians(anguloi+180)));
  }
  popMatrix();
}
```

Para acabar con la gráfica debemos dibujar el objeto detectado en sí, para ello lo primero que debemos hacer es convertir la distancia medida a número de pixeles en el dibujo siendo la distancia máximo 2 m que se corresponde a los 480 pixeles máximos. Si la distancia es menor de los 2 m que tiene este sensor como distancia máxima de medida la dibujamos con una línea roja en el radial correspondiente. Y el mismo procedimiento se ha de seguir con el segundo sensor, pero teniendo en cuenta que está barriendo los radiales suplementarios por lo que al ángulo enviado por Arduino se le debe sumar 180º.

```
void keyPressed() //Cuando se pulsa una tecla
{
  //Pulsar la tecla E para salir del programa
  if (key=='e' || key=='E')
  {
    output.flush(); // Escribe los datos restantes en el archivo
    output.close(); // Final del archivo
    exit();//Salimos del programa
  }
}
```

Para finalizar simplemente se añadirá la opción de salida rápida del programa pulsando la tecla “E” del teclado, cuando esto ocurra cerrará el archivo .txt y se cerrará la ventana gráfica.

El resultado final de las gráficas creadas por este programa se puede ver en las figuras 32 y 33.

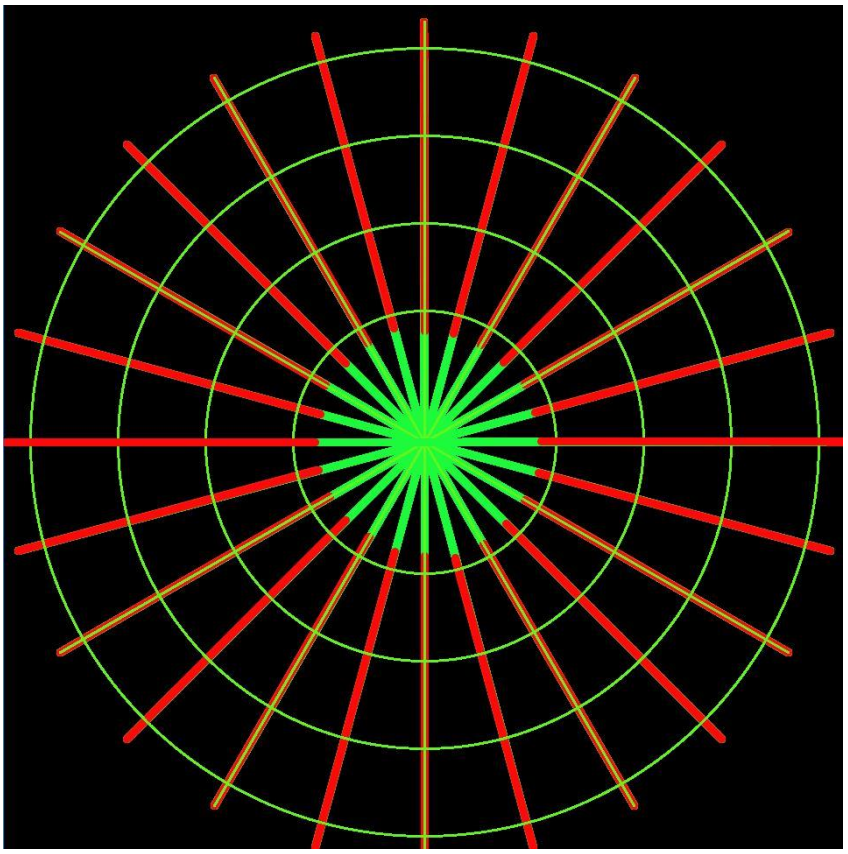


Figura 32: Caracterización del sistema con obstáculos a 50 cm.

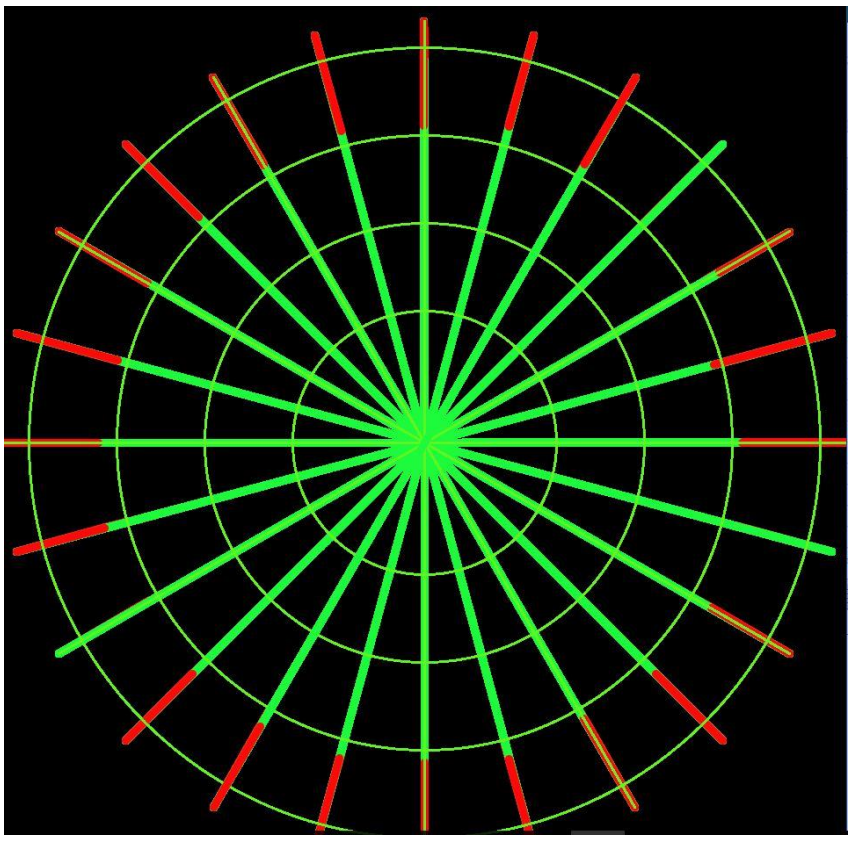


Figura 33: Caracterización de los sensores con obstáculos a 1.5 m.

3.2.3. Análisis de datos.

En esta sección se va a hacer un estudio de los datos adquiridos durante la caracterización del sistema completo. Como se ha comentado anteriormente las medidas se van a tomar a distintas distancias, siempre en el mismo entorno y con las mismas condiciones para que estas no influyan en la caracterización.

De los datos obtenidos se va a estudiar la media de las medidas tomadas a cada una de las distintas distancias (tras eliminar los datos espurios), el error cometido en dichas medidas y el porcentaje de medidas no válidas (%MNV). Se tomarán como medidas no válidas todas las que podamos identificar como incorrectas, es decir, distancias menores a 10 mm o superiores a 8000 mm (cuando no detecta nada en vez de devolver un infinito devuelve un 8190). Esto se hace siguiendo las siguientes fórmulas e implementándolas en una tabla de Excel para facilitar su estudio:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

$$\varepsilon = \bar{x} - Valor\ real$$

Los resultados obtenidos en este estudio se muestran en la siguiente tabla para que sean fácilmente analizables de un solo vistazo:

Distancia REAL (mm)		Media SIN espurios (mm)	MEDIA (mm)	%MNV		Error (mm)	
500	Sensor 1	544,59	546,82	0%	0%	44,59	46,82
	Sensor 2	549,05		0%		49,05	
750	Sensor 1	790,45	793,57	1%	0%	40,45	43,57
	Sensor 2	796,68		0%		46,68	
1000	Sensor 1	1058,37	1050,49	3%	2%	58,37	50,49
	Sensor 2	1042,61		2%		42,61	
1250	Sensor 1	1290,96	1303,53	1%	3%	40,96	53,53
	Sensor 2	1316,10		4%		66,10	
1500	Sensor 1	1566,68	1558,81	17%	20%	66,68	58,81
	Sensor 2	1550,94		22%		50,94	
1750	Sensor 1	1797,37	1800,02	65%	52%	47,37	50,02
	Sensor 2	1802,67		40%		52,67	

Tabla 13: Análisis de datos de la caracterización del sistema.

Como se puede ver en la Tabla 13 el %MNV es prácticamente nulo hasta una distancia de 1 metro y 25 centímetros, pero a partir de un metro y medio sube muy bruscamente llegando en el metro 75 a ser de un 52%. Así solo se podría llegar a detectar los obstáculos que se encuentren hasta a un metro y medio de distancia con seguridad. Para

poder mejorar esto se aplicará un filtrado muy básico pero que supone una mejora considerable de los resultados y con el que nos quedaremos del lado de la seguridad. Este filtro se explicará en el apartado 3.3..

3.3. Filtrado de la señal

Con la finalidad de poder detectar obstáculos a una distancia mayor implementaremos un postprocesado de las medidas tomadas. Puesto que no podemos utilizar filtros complejos, como el famoso filtro de Kalman, ya que necesitaríamos varias medidas y de distintas naturalezas, optamos por una opción más sencilla. Además, ya que éste es un sistema crítico a la hora de que pueda cumplirse la misión, es decir, un fallo del sistema de detección de obstáculos podría desencadenar en no poder llevar a cabo la misión establecida, nos quedaremos del lado de la seguridad. Con esto se está consiguiendo que en el caso de que haya un obstáculo sea más probable detectarlo, pero también aumenta la posibilidad de que se perciba un obstáculo que en realidad no exista.

En este filtro lo que estamos haciendo es comprobar primero si se detecta un obstáculo, en caso de que así sea se dibuja dónde se encuentra. Si no se detectase nada se comprueba si en el barrido anterior había alguno, de ser así se dibuja este obstáculo en un color diferente para indicar que probablemente ahí siga habiendo algo. Esto es así ya que lo normal es que un obstáculo no aparezca y desaparezca repentinamente, sino que se vaya alejando hasta desaparecer, así en el caso de que se deje de detectar un objeto hay una alta posibilidad de que sea un fallo de medición y este siga estando ahí.

El programa de Processing tras implementar esta modificación queda como sigue, en este caso solo se comentarán los cambios con respecto al programa realizado para la caracterización del sistema:

```
import processing.serial.*; //Importamos la librería Serial


Serial port; //Nombre del puerto serie

int xPos = 1;         // horizontal position of the graph
float x1 = 0;
float x2=0;
float y1=0;
float y2 = height / 2;
int iteraciones=0;
PrintWriter output;  //Para crear el archivo de texto donde guardar los datos


String  dist1, dist2, angulo, data;//Valor de la distancia
int anguloi, indice;
int index1=0;
int index2=0;
float pixsDistance, pixsDistance2, pixsDistanceold, pixsDistance2old;
int[] dist1i={8000,8000,8000,8000,8000,8000,8000,8000,8000,8000,8000,8000};
int[] dist2i={8000,8000,8000,8000,8000,8000,8000,8000,8000,8000,8000,8000};
int[] dist1old={8000,8000,8000,8000,8000,8000,8000,8000,8000,8000,8000,8000};
int[] dist2old={8000,8000,8000,8000,8000,8000,8000,8000,8000,8000,8000,8000};
```

En este caso las distancias deben guardarse en un vector, con tantas posiciones como lecturas se hagan en un barrido, para poder guardar las lecturas que se han hecho en el barrido anterior para el mismo ángulo.

```
void draw()
{
  fill(98, 245, 31); // color verde
  // Llamamos a las funciones para dibujar
  drawRadar();
  drawLine();
  drawObject();
  drawObjectOld();

  data=port.readStringUntil('\n'); //Lemos toda un linea de datos
  if (data!=null && data.length()>2 ) {

    data=data.substring(0, data.length()-2); //Eliminamos el \n del string
    index1=data.indexOf(','); //Buscamos la primera , (primer separador entre datos)
    index2=data.indexOf(',', index1+1); //Buscamos la segunda , (segundo separador entre datos)
    if (index1>0 && index2>0) {
      angulo=data.substring(0, index1); //Obtenemos el primer dato que nos da el string de datos
      dist1=data.substring(index1+1, index2); //Obtenemos el segundo dato que nos da el string de datos
      dist2=data.substring(index2+1, data.length()); //Obtenemos el tercer dato que nos da el string de datos
```

Aquí el único cambio que podemos observar con respecto al programa de la caracterización es que se llama a una nueva función de dibujar llamada drawObjectOld() que se especificará su funcionamiento más adelante.

```
      //Los convertimos a numeros para poder operar con ellos
      anguloi=int(angulo);
      switch(anguloi) {
      case 0:
        indice=0;
        break;
      case 15:
        indice=1;
        break;
      case 30:
        indice=2;
        break;
      case 45:
        indice=3;
        break;
      case 60:
        indice=4;
        break;
      case 75:
        indice=5;
        break;
      case 90:
        indice=6;
        break;
      case 105:
        indice=7;
        break;
      case 120:
        indice=8;
        break;
      case 135:
        indice=9;
        break;
      case 150:
        indice=10;
        break;
      case 165:
        indice=11;
        break;
      }
```

```
      // Guardamos los datos antiguos para hacer un filtrado
      dist1old[indice]=dist1i[indice];
      dist2old[indice]=dist2i[indice];

      //Actualizamos los datos
      dist1i[indice]=int(dist1);
      dist2i[indice]=int(dist2);
    }

    if (angulo != null && dist1!=null && dist2!= null) {
      // Lo escribes en el .txt
      iteraciones=iteraciones+1;
      output.print(angulo+","+dist1+","+dist2+"\n");
      //println(angulo);
      //println(dist1);
      //println(dist2);
    }
  }
}
```

A diferencia del caso anterior aquí lo primero que vamos a hacer tras separar cada uno de los datos recibidos es conocer el ángulo en el que se están tomando las medidas, para cada uno de los ángulos posibles se le asigna una posición (variable índice) en los vectores de distancias medidas. Así podemos guardar la distancia medida en la posición correspondiente de las variables dist1 y dist2, además previamente los valores anteriores de distancias medidas se guardan en dist1old y dist2old para poder comparar los datos actuales con los previos y realizar así el filtrado correspondiente.

Las funciones drawRadar() y drawLine() quedan igual que en el programa anterior y los cambios se efectúan en las funciones donde se dibujan los obstáculos detectados:

```
void drawObject() {
  pushMatrix();
  translate(480, 500);
  strokeWeight(9);
  stroke(255, 10, 10); // color rojo
  pixsDistance = dist1i[indice]*0.2375; // convierte la distancia obtenida a pixeles en el dibujo
  if (10< dist1i[indice] && dist1i[indice]<2000) {
    // dibuja el objeto de acuerdo con el angulo y la distancia en la que se encuentra
    line(pixsDistance*cos(radians(anguloi)), -pixsDistance*sin(radians(anguloi)), ...
    480*cos(radians(anguloi)), -480*sin(radians(anguloi)));
  }
  pixsDistance2 = dist2i[indice]*0.2375;
  //println(pixsDistance2+","+dist2i+","+dist2);
  if (10<dist2i[indice] &&  dist2i[indice]<2000) {
    line(pixsDistance2*cos(radians(anguloi+180)), -pixsDistance2*sin(radians(anguloi+180)), ...
    480*cos(radians(anguloi+180)), -480*sin(radians(anguloi+180)));
  }
  popMatrix();
}
```

En la función drawObject(), donde se dibujan los obstáculos detectados con una línea roja, el cambio efectuado es que a la hora de dibujar dicha línea se debe trabajar únicamente con el valor de distancia que se encuentra en la posición *indice* adecuada. Además, se ha impuesto una condición más para que el valor sea tomado por válido, que en este caso es que la distancia sea mayor a un centímetro. Esto se ha hecho ya que cuando el sensor no está bien conectado devuelve una distancia de 0 mm o, en raras ocasiones, valores menores de 10 mm que son incorrectos. Por otro lado, debido a las dimensiones del dron, cualquier valor tan pequeño, aunque fuera medido realmente

con el sensor laser, se trataría de una interferencia con la propia estructura del dron. Así con esta nueva restricción no estamos eliminando nunca un obstáculo real.

```
void drawObjectOld() {
  pushMatrix();
  translate(480, 500);
  strokeWeight(9);
  stroke(300, 10, 100); // color rojo flojito
  pixsDistanceold = dist1old[indice]*0.2375; // convierte la distancia obtenida a pixeles en el dibujo
  if (dist1i[indice]>2000 && 10< dist1old[indice] && dist1old[indice]<2000) {
    // dibuja el objeto donde se encontraba anteriormente aunque en esta medida no esté
    line(pixsDistanceold*cos(radians(anguloi)), -pixsDistanceold*sin(radians(anguloi)), ...
    480*cos(radians(anguloi)), -480*sin(radians(anguloi)));
  }
  pixsDistance2old = dist2old[indice]*0.2375;
  if (dist2i[indice]>=2000 && 10< dist2old[indice] && dist2old[indice] < 2000) {
    line(pixsDistance2old*cos(radians(anguloi+180)), -pixsDistance2old*sin(radians(anguloi+180)), ...
    480*cos(radians(anguloi+180)), -480*sin(radians(anguloi+180)));
  }
  popMatrix();
}
```

Por último, se ha añadido una nueva función, llamada drawObjectold() en la que se dibujará el posible obstáculo aun cuando el sistema no lo esté midiendo en realidad. Para ello seguimos el mismo procedimiento para dibujar que cuando se detecta un objeto, pero con líneas en otro color para poder distinguirlo, en este caso serán magenta. La condición para dibujar este posible obstáculo es que no se haya detectado nada en este barrido, es decir que la distancia medida sea mayor a 2 m, pero que en el barrido anterior si haya un obstáculo detectado, a una distancia entre 10 mm y 2 metros. Si se cumple esto se dibuja una línea magenta donde podría encontrarse un obstáculo no detectado.

3.3.1. Análisis de datos.

Con las mismas medidas que se tomaron para la caracterización del sistema se hará un estudio de cómo actúa el filtrado. Teniendo en cuenta este cambio haremos el mismo estudio estadístico de la media y los errores que se tienen en las medidas a distintas distancias.

Una vez más los resultados obtenidos en este nuevo estudio se muestran en una tabla para poder evaluarlos más fácilmente:

Distancia REAL (mm)		Media SIN espurios (mm)	MEDIA (mm)	%MNV		Error (mm)	
500	Sensor 1	544,59	546,82	0%	0,0%	44,59	46,82
	Sensor 2	549,05		0%		49,05	
750	Sensor 1	790,49	793,59	0%	0,0%	40,49	43,59
	Sensor 2	796,68		0%		46,68	
1000	Sensor 1	1058,77	1050,95	0%	0,0%	58,77	50,95
	Sensor 2	1043,12		0%		43,12	
1250	Sensor 1	1290,86	1303,14	0%	0,3%	40,86	53,14
	Sensor 2	1315,42		1%		65,42	
1500	Sensor 1	1564,13	1557,16	5%	5,2%	64,13	57,16
	Sensor 2	1550,19		6%		50,19	
1750	Sensor 1	1834,67	1830,17	41%	30,2%	84,67	80,17
	Sensor 2	1825,68		20%		75,68	

Tabla 14: Caracterización del sistema tras el filtrado.

Como se puede observar en la Tabla 14 este tipo de postprocesado de datos funciona muy bien cuando el número de medidas no válidas es relativamente pequeño, esto se puede ver por ejemplo con el %MNV en distancias de 1,50 m en la que disminuye de un 20% a un 5%. Para distancias mayores como puede ser 1,75 m el %MNV también disminuye considerablemente, de un 52% a un 30%, pero aun así sigue teniendo valores altos. Aun así, gracias a este filtrado podemos trabajar con distancias mayores con bastante seguridad y con una reducción muy considerable de fallo al tomar las medidas.

4. Validación

Tras el montaje y programación se este sistema, mostrada con detalle en el apartado 2., y el proceso de caracterización que nos ha permitido conocer mejor nuestro sistema y solucionar errores, por ejemplo mediante filtrado, hemos llegado al sistema final.

Una vez ya se tiene todo el sistema montado y caracterizado lo único que nos queda es probar su funcionamiento y estudiar la respuesta que nos da cuando se le presentan distintos obstáculos, de diferentes formas, superficies y colores, y a distintas distancias ya que en definitiva este sistema debe poder trabajar en situaciones reales.

Se tomaron medidas con tres obstáculos diferentes para comprobar los resultados. La localización y forma de dichos obstáculos se muestra en la Figura 34 y las medidas que nos mostraba nuestro programa final se muestran en las Figuras 35 y 36.

Figura 34: Obstáculos a medir para validar nuestro sistema.

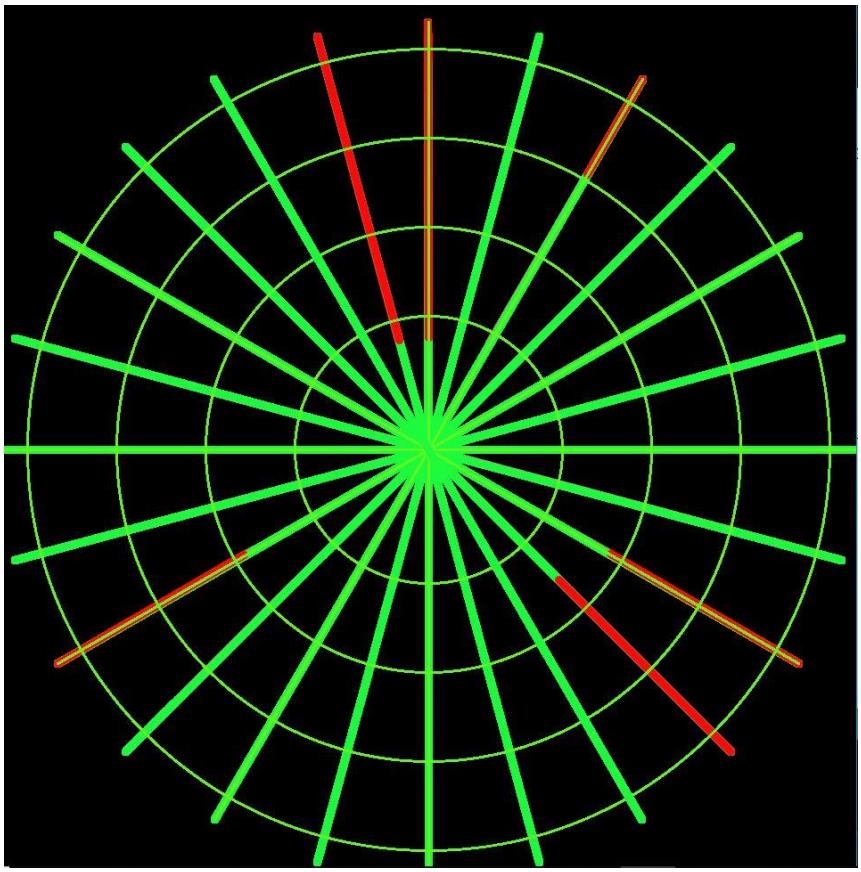


Figura 35: Resultado medición de obstáculos para validación.

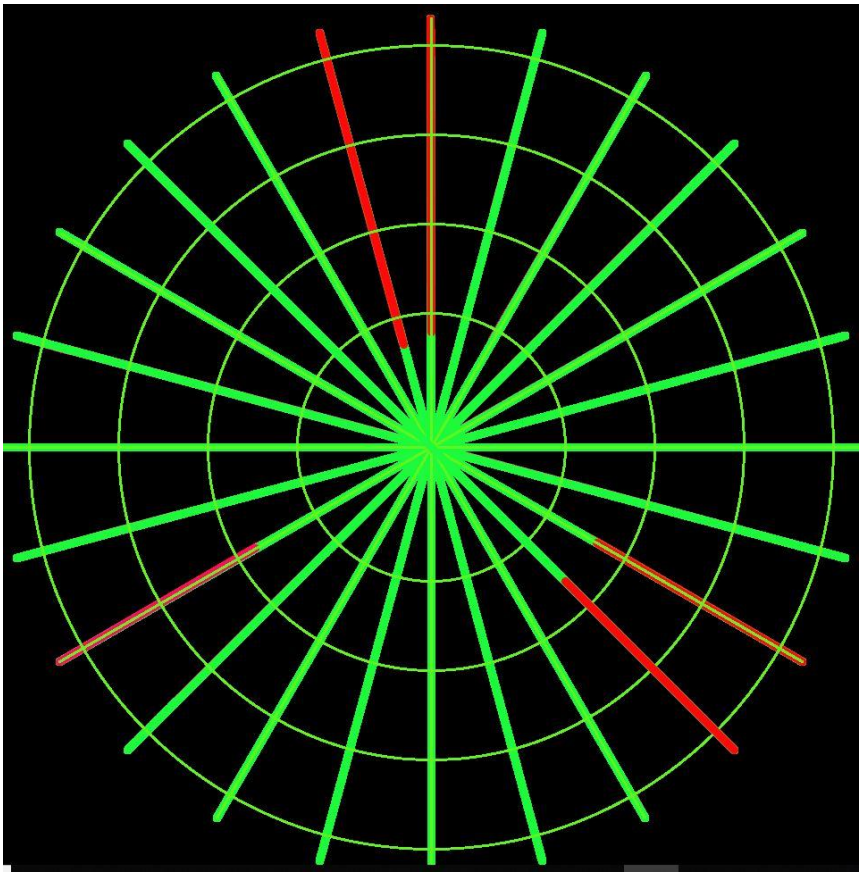


Figura 36: Resultado medición de obstáculos para validación 2.

En la Figura 35 podemos observar que los objetos son medidos sin problemas, pero aparece un obstáculo no existente en el radial 120 a unos 150 cm del dron. Este dato obviamente es erróneo. Seguramente sea debido a un reflejo provocado por la ventana, ver la configuración del banco de medidas en la Figura 34. Este tipo de error es fácilmente corregible utilizando sensores mejores, algo que en nuestro caso no era muy recomendable ya que sólo es una investigación a nivel educativo y el presupuesto era muy limitado, pero en una empresa que utilizase este sistema el gasto es afrontable y se obtendrían beneficios económicos importantes. Además, en nuestro caso nos estamos poniendo siempre del lado de la seguridad y nos centramos más en que se detecten todos los obstáculos que en detectar algo que no exista.

Por otro lado, en la Figura 36 vemos que uno de los objetos, en este caso la mochila, sí que es detectado, pero no por el láser en sí sino como producto del postprocesado. Esto lo podemos ver ya que en la imagen este obstáculo se muestra en color morado lo que, tal y como se explica en el apartado 3.3., significa que no se ha detectado nada, pero por filtrado sabemos que hay una alta posibilidad de que en esa localización haya un objeto.

En definitiva, con estas, y otras muchas, medidas se ha llegado a validar este sistema comprobando que efectivamente detecta los obstáculos independientemente de sus características. En este ejemplo podemos ver que cada obstáculo es de un color, con una forma y un tamaño diferentes, y cada uno se encuentra a una distancia, y aun así los distingue todos sin problema. Es cierto que se detectan ciertos errores que en un sistema crítico real no serían tolerables, pero en este Trabajo Fin de Grado se asienta una base para un futuro desarrollo y perfeccionamiento de este sistema.

5. Conclusiones

El objetivo de este Trabajo Fin de Grado era poder detectar los obstáculos que se encuentran en el entorno más inmediato del dron con la finalidad de, en proyectos futuros, poder evitarlos.

Para dicha detección se han investigado las distintas posibilidades de sistemas de medición de distancia con el objetivo de encontrar el que más se adapte a nuestras necesidades. Concluimos que la mejor opción es el empleo de sensores láser, más concretamente del sensor VL53L0X, debido a sus buenas características de precisión para su bajo coste y relativamente baja dificultad computacional.

En lo referente al diseño del sistema completo se deduce que es susceptible de mejora ya que el movimiento del servomotor empleado es demasiado lento si se quieren tener buenas medidas. A la hora de implementar este sistema en un UAV real habría que buscar una forma de acelerar el proceso de medidas para poder hacer un vuelo seguro y a una velocidad razonable.

Por otro lado, tras la caracterización tanto de los sensores como del sistema en su conjunto se puede observar que a partir del metro y medio empiezan a fallar las medidas. Si queremos optimizar este sistema para ser empleado en la vida real esto debe mejorarse hasta conseguir llegar a los 2 metros con fiabilidad. En este proyecto se ha propuesto la solución de un filtrado de la señal muy sencillo, que ha dado unos resultados considerablemente buenos. En posteriores investigaciones se propone hacer un filtrado de la señal más avanzado hasta conseguir erradicar, en la medida de lo posible, este problema. Con todo esto no debemos olvidar que estamos trabajando con sistemas de tiempo real por lo que no se puede proponer un procesado demasiado complicado.

Aun con los problemas mencionados anteriormente, se ha podido demostrar en la validación de este sistema la viabilidad de la prueba de concepto desarrollada en este trabajo. Se ha conseguido el objetivo perseguido de desarrollar un sistema de detección de obstáculos mediante el uso de sensores láser.

Asimismo, se ha intentado optimizar al máximo el sistema electrónico, aunque el programa realizado en Arduino puede ser planteado de otras maneras que sean igual de válidas o mejores, el realizado en este trabajo funciona correctamente y consigue minimizar los tiempos con respecto a otras posibilidades planteadas durante el desarrollo del mismo. Algo parecido ocurre con la aplicación de Processing, que se ha hecho de la forma que nos ha parecido la más correcta, pero las posibilidades que ofrece este programa son infinitas y se pueden encontrar otras múltiples soluciones. El filtrado realizado en esta aplicación, obviamente, puede mejorarse y el diseño de la representación gráfica es algo muy subjetivo, se ha hecho siguiendo nuestros gustos, pero puede cambiarse sin mucha dificultad en caso de que se tenga otra idea.

En definitiva, se concluye que la prueba de concepto presentada en este Trabajo Fin de Grado ha alcanzado los objetivos que se habían planteado inicialmente. Lo desarrollado

en este proyecto se trata de un concepto válido como sistema de detección de obstáculos basado en sensores láser eficaces, precisos y de bajo coste.

En cuanto a las posibles líneas de investigación futuras sobre este trabajo se derivan especialmente cuatro ramas de estudio. Toda esta investigación esta basada en lo expuesto en este Trabajo Fin de Grado y se propone como futuras ampliaciones de este.

Como primer avance, sin cambiar ningún elemento del sistema, se puede realizar un postprocesado más complejo que el estudiado en el presente trabajo. El primer avance, sería dar aviso de posible obstáculo comprobando, no solamente la medida inmediatamente anterior en ese radial, sino también las de los radiales contiguos y estudiando las distancias obtenidas con más de una medida anterior en memoria. A partir de ahí, se puede complicar todo lo deseado. Existen otras opciones, como añadir sensores de una naturaleza distinta, por ejemplo los ultrasónicos, y realizar una fusión de los datos mediante el filtro de Kalman. Con esto las opciones son infinitas y según lo deseado se puede optar por una alternativa u otra.

Otra vía de desarrollo necesaria antes de poder certificar este sistema para poder integrarlo en un UAV real es conseguir realizar el barrido completo de los 360º a una velocidad mayor. La solución más inmediata sería cambiar el servomotor MG995 empleado en este proyecto por otro que complete los movimientos más rápidamente y así no tener que realizar esperas tan largas entre medidas. Además, se pueden añadir más sensores para aprovechar más el movimiento y tomar más medidas en cada una de las posiciones que tome el motor.

También, se plantea como posible ampliación futura, el desarrollo competo de un sistema LIDAR de bajo coste a partir de los desarrollado en este Trabajo con los sensores VL53L0X. Esta aplicación requeriría de un alto grado de investigación de todos los elementos necesarios para conseguir un sistema robusto, preciso y barato de medición. Este proyecto es un primer paso hacia ese objetivo, ya se le ha dado movilidad en el eje z a los sensores, haciendo un barrido de 360º, pero se necesita también realizar un barrido vertical en cada uno de los radiales para tener datos en tres dimensiones. Este tipo de sistema se integraría posteriormente en un UAV real y así se podría mapear todo el entorno para conocer los posibles obstáculos.

Por último, aunque es algo que quedaba fuera completamente de este proyecto, se puede realizar un algoritmo con el cual, una vez conocidos todos los obstáculos, consiga eludirlos y llevar al dron por un entorno seguro evitando cualquier posible colisión. Podría llegar incluso a conseguirse el vuelo totalmente autónomo, sin necesidad de ser pilotado a distancia, conociendo completamente su entorno y su misión.

6. Presupuesto

Para finalizar con este Trabajo Fin de Grado sobre el desarrollo de un sistema de detección de obstáculos para drones basado en sensor láser se especifican en este apartado los costes aproximados de la realización del proyecto en su totalidad.

Los costes se desglosan principalmente en dos grandes bloques. El primero de ellos corresponde al importe de los componentes necesarios para el correcto desarrollo e implementación del sistema propuesto. En cambio, el segundo bloque se centra en los gastos derivados de la mano de obra, investigación y los ensayos experimentales y de calibración. De esta forma, cada uno de los dos bloques representa un presupuesto parcial que, en su conjunto, constituye el presupuesto total aproximado del proyecto.

Asimismo, se ha de tener en cuenta que el presupuesto versa sobre la estructura FY450 utilizada y no contempla la posible fabricación en serie del mismo en drones reales lo que haría incrementar su coste de forma muy considerable.

De igual forma, el valor de los costes asociado a los componentes necesarios para la implementación del sistema presentados en este presupuesto podría verse afectado por el desarrollo posterior de este proyecto con elementos de mayor calidad y precisión, o por la variación de precios en un mercado global.

MATERIALES

En lo referente al coste del material necesario para la elaboración del estudio, el correcto desarrollo y posterior implementación del sistema propuesto, el importe proviene del precio de venta al público suministrado por cada uno de los fabricantes oficiales de los componentes empleados.

Así, se incluye dentro del material empleado, todo lo relacionado con el propio sistema eléctrico: la placa Arduino UNO, los sensores láser VL53L0X, el cableado, etc. Asimismo, todos los elementos relaciones con el montaje del prototipo final son también incluidos dentro de este presupuesto parcial. Además, se tendrán en cuenta también todos los elementos necesarios para la elaboración del banco de pruebas utilizado en la calibración.

Por otro lado, habría que tener en cuenta el coste asociado al software empleado, dado que tanto Arduino como Processing son plataformas de uso libre (open-source), no se incluye ningún gasto referido a licencias de este software. Por otro lado, Autodesk proporciona una versión para estudiante tanto de Inventor como para el resto de sus productos que, al identificarnos como estudiantes de la Universidad Politécnica de Valencia, podemos descargar sin coste alguno. Y, por último, se está trabajando con la versión de Microsoft Office suministrada por la universidad a todos sus alumnos, la licencia correspondiente a este software sí que está siendo pagada por lo que entrará dentro de nuestro presupuesto.

De esta forma, se muestra en la Tabla 15 el cómputo total del coste del material necesario con el 21% del IVA ya aplicado sobre cada uno de los componentes.

Descripción	Cantidad (ud.)	Coste ud. (€/ud.)	Importe (€)
Estructura FY450	1	15.89	15.89
Arduino UNO Rev3	1	20	20
Módulo VL53L0X	2	10.85	21.70
Servomotor MG995	1	11.99	11.99
Cable Jupiter Macho-Hembra	20	0.11	2.20
Tornillos 4x40 (mm)	4	0.25	1.00
Palos de madera	1	0.75	0.75
Cartulina	16	0.50	8
Cinta aislante	1	0.75	0.75
Cinta de doble cara	1	1.50	1.50
Tabla de madera (20x20 cm)	1	0.50	0.50
Microsoft Office 365	5 meses	8.80 usuario/mes	44
TOTAL			**128.28**

Tabla 15: Presupuesto para materiales empleados.

MANO DE OBRA

En lo referente a la mano de obra, se ha trabajado con diversos perfiles dependiendo de la tarea a realizar. En este sentido, para el desarrollo del Trabajo, se ha requerido mano de obra ingenieril para la investigación, desarrollo conceptual, ensayos experimentales y montaje del sistema (un graduado en Ingeniería Aeroespacial) y, también, mano de obra técnica para el desarrollo del montaje (técnico de laboratorio).

El precio unitario de la mano de obra se ha calculado como media de las bases y tipos de cotización de 2018 para el grupo de cotización 2, correspondiente a la Categoría Profesional de “Ingenieros Técnicos, Peritos y Ayudantes Titulados”, publicado por la Seguridad Social.

Así, la Tabla 16 hace referencia a dichos costes asociados a la mano de obra, dependiendo de la categoría del perfil y de unas horas determinadas para el desarrollo completo del proyecto.

Mano de obra	Horas	Coste por hora (€/hora)	Importe (€)
Graduado en ingeniería	150	14.12	2118.00
Técnico de Laboratorio	10	7.16	71.60
TOTAL			**2189.60**

Tabla 16: presupuesto para mano de obra.

PRESUPUESTO TOTAL

Finalmente, y considerando el desglose del presupuesto elaborado, se obtiene el importe total del proyecto realizado. Dicho presupuesto incluye todo lo referente a la investigación, desarrollo e implementación de todo lo expuesto en la memoria.

A dicho coste se le aplicará, únicamente sobre la mano de obra, el vigente impuesto sobre el valor añadido (IVA) del 21 %, puesto que el resto de componentes lo llevan incluido al ser adquiridos de forma externa.

Además, debemos incluir también una serie de gastos dentro de un concepto denominado medios auxiliares con un valor atribuido del 5% sobre la cuantía total de la mano de obra y los materiales. Dicho concepto comprende todo lo relacionado con posibles costes indirectos de la realización del proyecto, tales como: el mantenimiento de las instalaciones, consumo eléctrico, material adicional, horas extra del personal, imprevistos varios, etc.

		Base Imponible (€)	Importe (€)
Material empleado			128.28
Mano de obra			2189.60
Medios auxiliares	**5%**	2317.88	115.90
IVA	**21%**	2189.60	459.82
TOTAL			**2893.60**

Tabla 17: Presupuesto total del Trabajo Fin de Grado.

Con todo esto, tras desglosar los diferentes conceptos, el coste total que este Trabajo Fin de Grado ha acarreado es de **2893.60 €.**

7. Bibliografía

[1] Y. Wu, Y. Sui y G. Wang, «Vision-Based real-time aerial object localization and tracking for UAV Sensing System,» *IEEE Access,* Octubre 2017.

[2] N. Gageik, S. Montenegro y P. Benz, «Obstacle Detection and Collision Avoidance for a UAV with complementary low-cost sensors,» *IEEE Access,* Junio 2015.

[3] R. Rambabu, M. R. Bahiki y S. Azrad, «Relative position-based collision avoidance system for swarming UAVs using multi-sensor fusion.,» *Journal of Engineering and Applied Sciences,* Noviembre 2015.

[4] C. Baraniuk, «US military tests swarm of mini-drones launched from jets,» *BBC News,* 10 Enero 2017.

[5] X. Wang, V. Yadav y S. Balakrishnan, «Cooperative UAV Formation Flying Obstacle/Collision Avoidance,» *IEEE Transactions on control systems technology,* vol. 15, nº 4, Julio 2007.

[6] J. B. Saunders, B. Call, A. Curtis y R. W. Beard, «Static and Dynamic Obstacle Avoidance in Miniature Air Vehicles,» *AIAA Infotech@Aerospace,* Septiembre 2005.

[7] « https://www.arduino.cc,» [En línea].

[8] Atmel, *ATmega328/P DATASHEET,* 2016.

[9] STMicroelectronics, *VL53L0X DATASHEET,* 2016.

[10] STMicroelectronics, *VL53L0X, User Manual,* 2016.

[11] «processing.org,» [En línea].

A. Anexo I: Diseño pieza motor

Con el fin de poder acoplar los dos sensores láser VL53L0X de forma adecuada sobre el motor servo que los mueve se ha fabricado una pieza mediante la impresora 3D cuyo diseño se ha realizado con el programa Autodesk Inventor. Este es un software de modelado CAD 3D para diseño mecánico y de productor. A partir de este es muy sencillo imprimir la pieza diseñada con una impresora 3D, esto más el hecho de que el software sea gratuito para estudiantes y tenga buenos tutoriales para usuarios nos ha hecho decantarnos por este a la hora de realizar esta pieza.

A.1. Diseño

En este programa, al usar formato CAD, lo que vamos a hacer en cada fase del diseño siempre va a ser hacer un primer croquis o *Sketch* en 2D de la planta de la forma a realizar. A partir de este se va a extruir una pieza en tres dimensiones, es decir, se le va a dar un valor para la tercera dimensión a este *Sketch* obteniendo así tanto bloques sólidos como orificios según se realice esta operación.

Para comenzar debemos dibujar la base de la figura en un *Sketch* para posteriormente hacer la extrusión adecuada, a partir de la cual irá surgiendo la pieza definitiva mediante distintas operaciones. En esta primera pieza base vamos a conformar el ortoedro sobre el que luego se harán los cambios adecuados. Además, aprovecharemos esta primera figura para realizar los dos agujeros circulares con los cuales daremos salida a los cables.

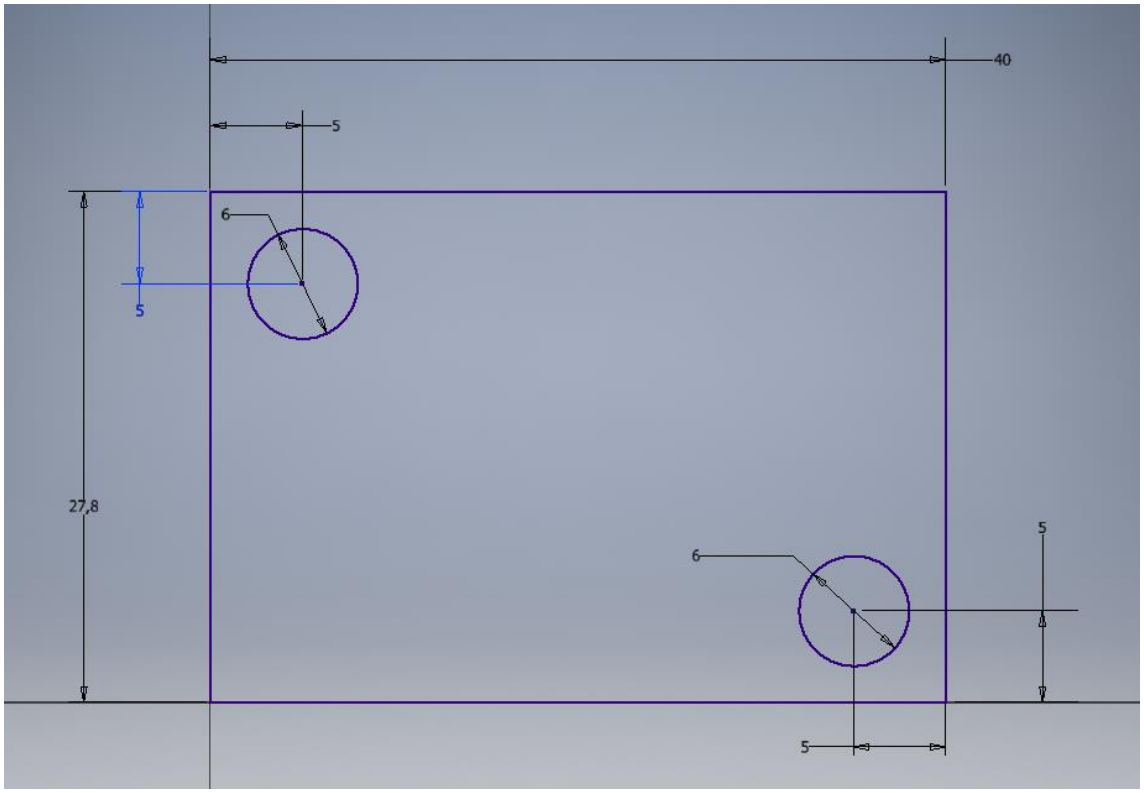


Figura 37: Sketch 1, figura base.

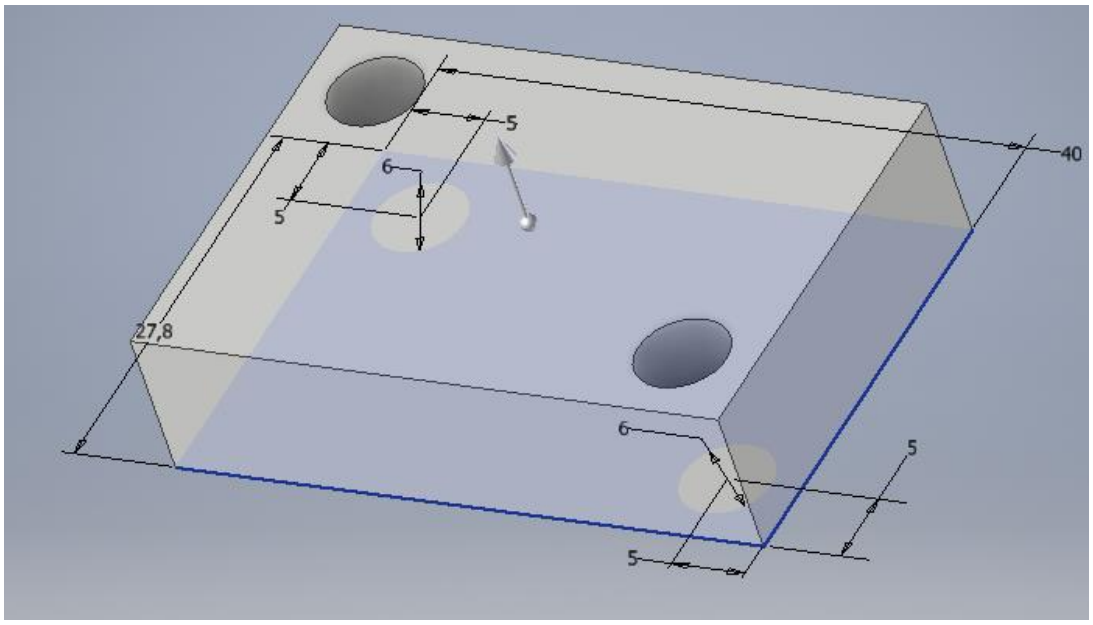


Figura 38: Extrusión 1, figura base.

A partir de aquí vamos a hacer dos hendiduras en el ortoedro para encajar los sensores y guiar los cables hasta los agujeros para darles salida. Puesto que es hacen falta dos hendiduras iguales para los dos sensores se deben hacer dos extrusiones y dos Sketch diferentes. En las siguientes imágenes mostramos el esquema donde se representan las dimensiones de las hendiduras a realizar y las dos extrusiones efectuadas.

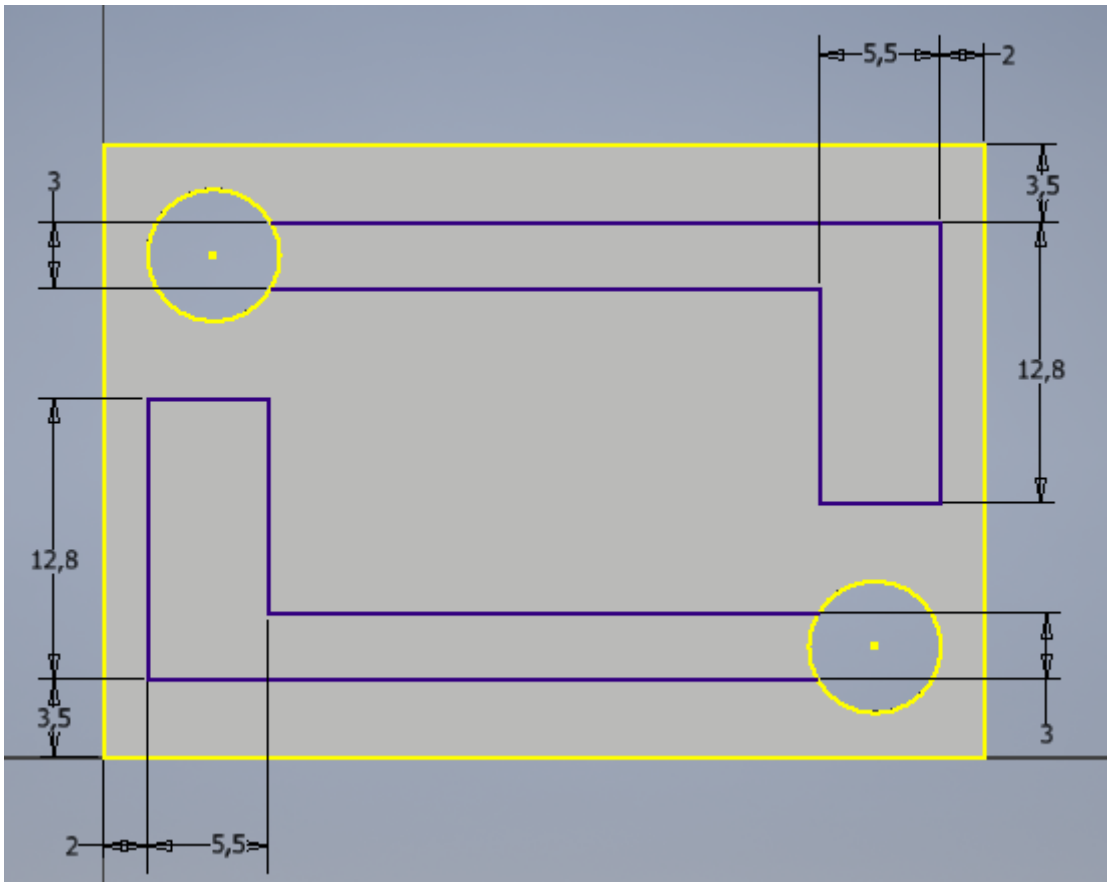


Figura 39: Sketch 2 y 3, hendiduras de los sensores.

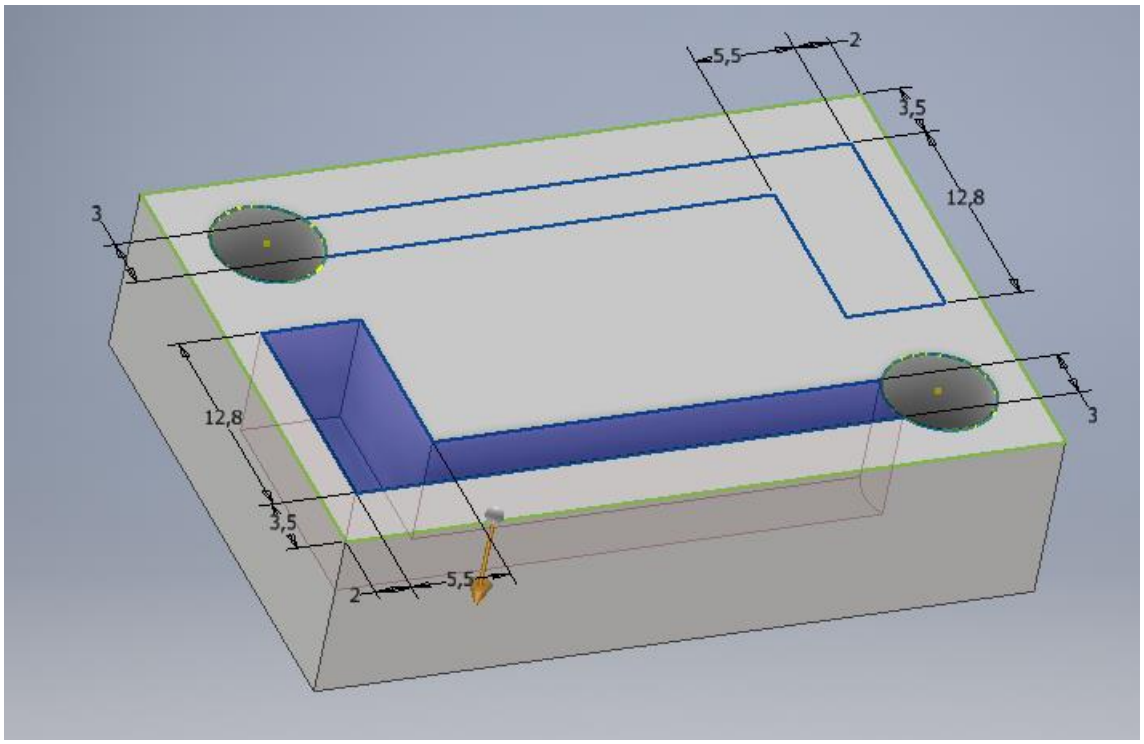


Figura 40: Extrusión 2, hendidura 1.

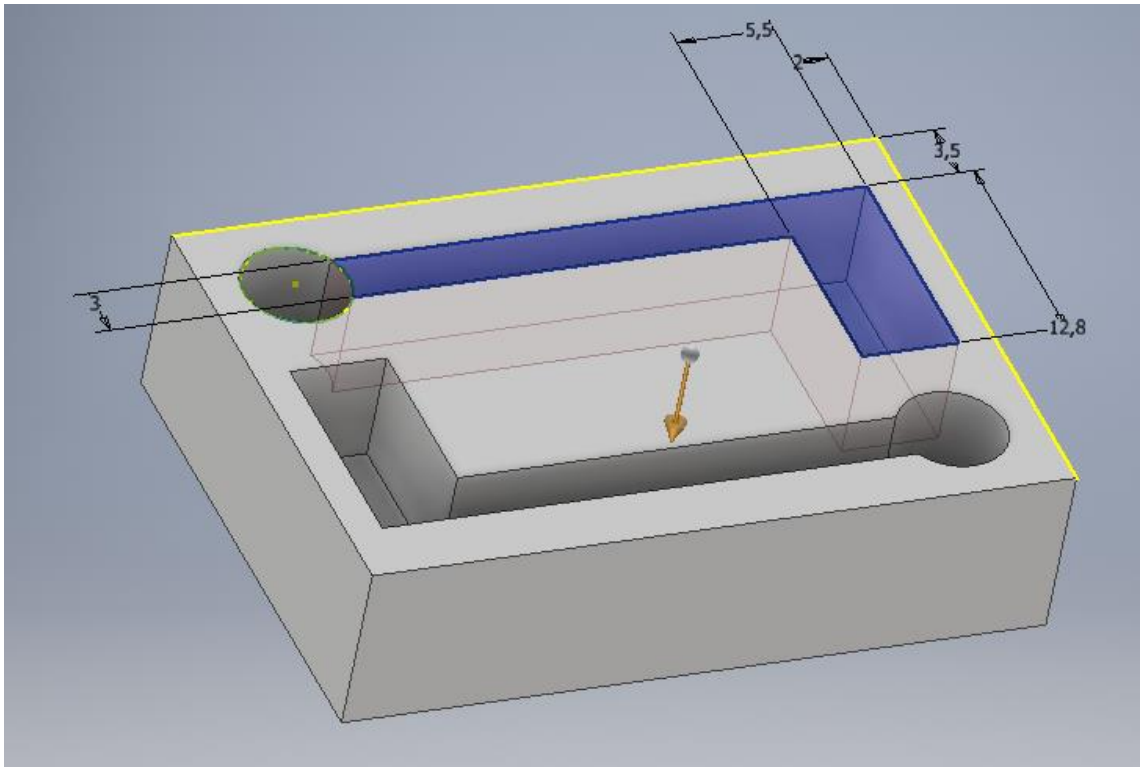


Figura 41: Extrusión 3, hendidura 2.

Por último, para terminar la pieza nos queda diseñar cómo vamos a unirla al motor. En este caso se va a acoplar a este mediante un cilindro hueco sin dientes de engranaje ya que se acoplará directamente por presión y en caso de que quedara holgada la unión se adherirá mediante silicona caliente. Para realizar esta última parte se debe hacer en dos

pasos, primero se hace el cilindro completamente sólido y después se realiza un orificio en forma cilíndrica de un radio menor al del cilindro inicial.

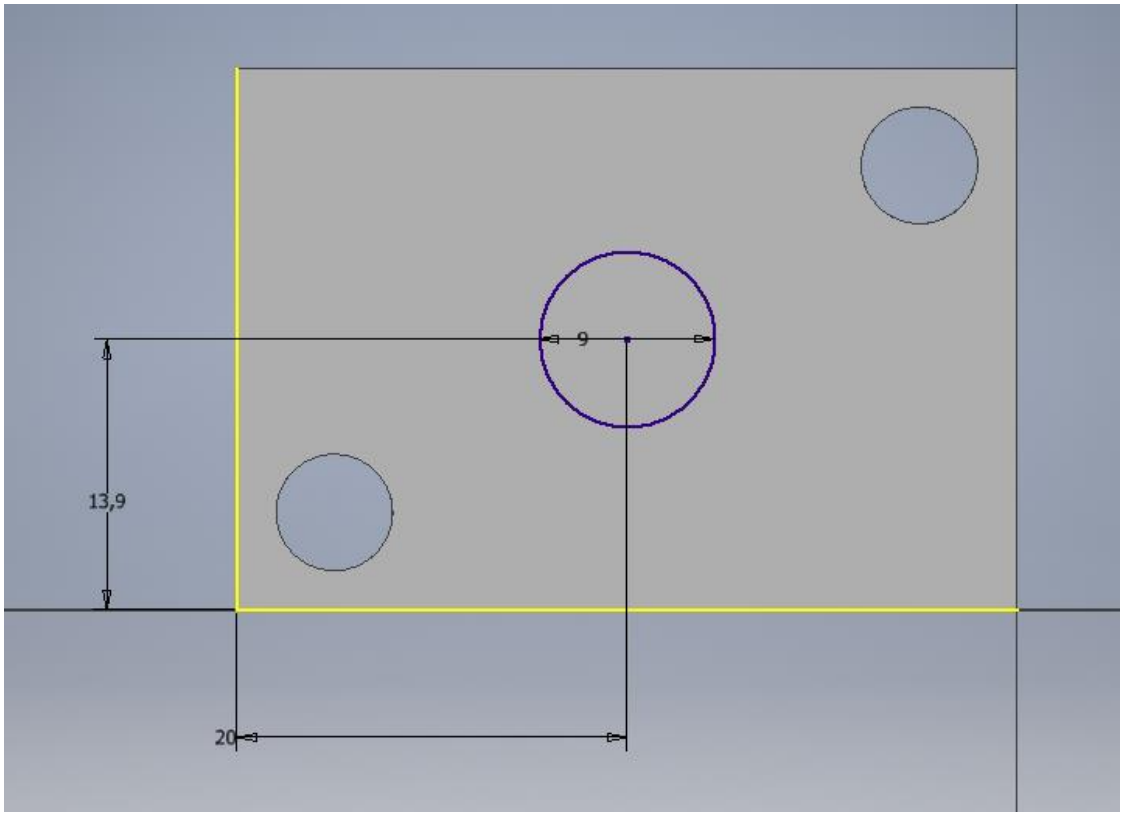


Figura 42: Sketch 4, cilindro exterior.

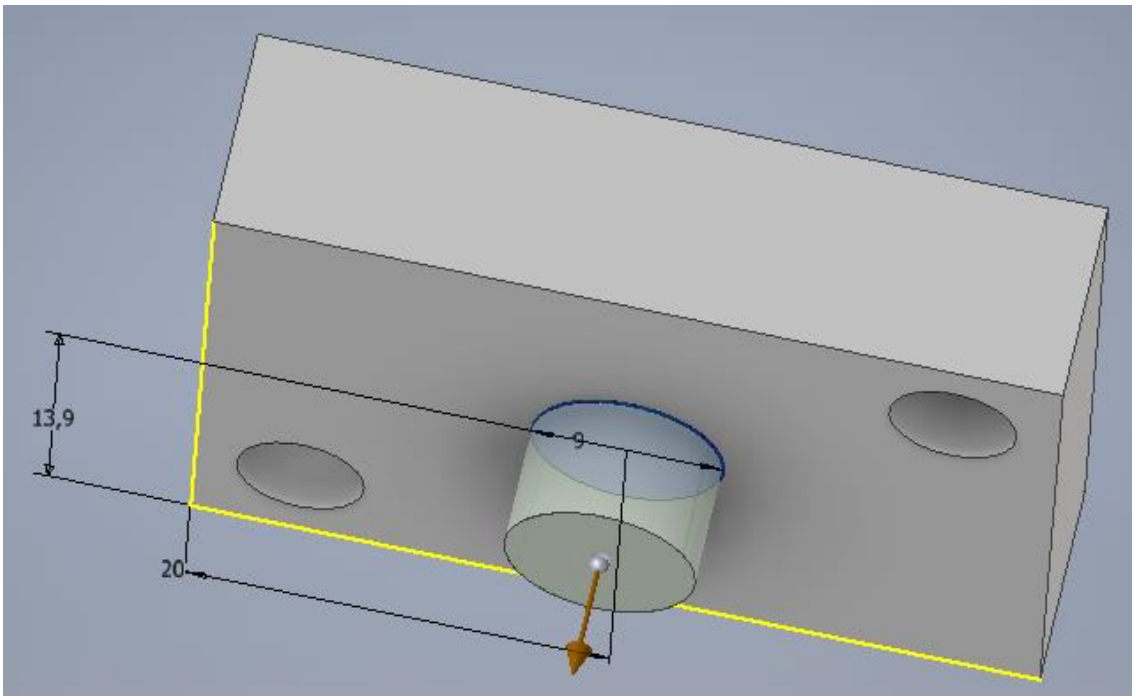


Figura 43: Extrusión 4, cilindro exterior.

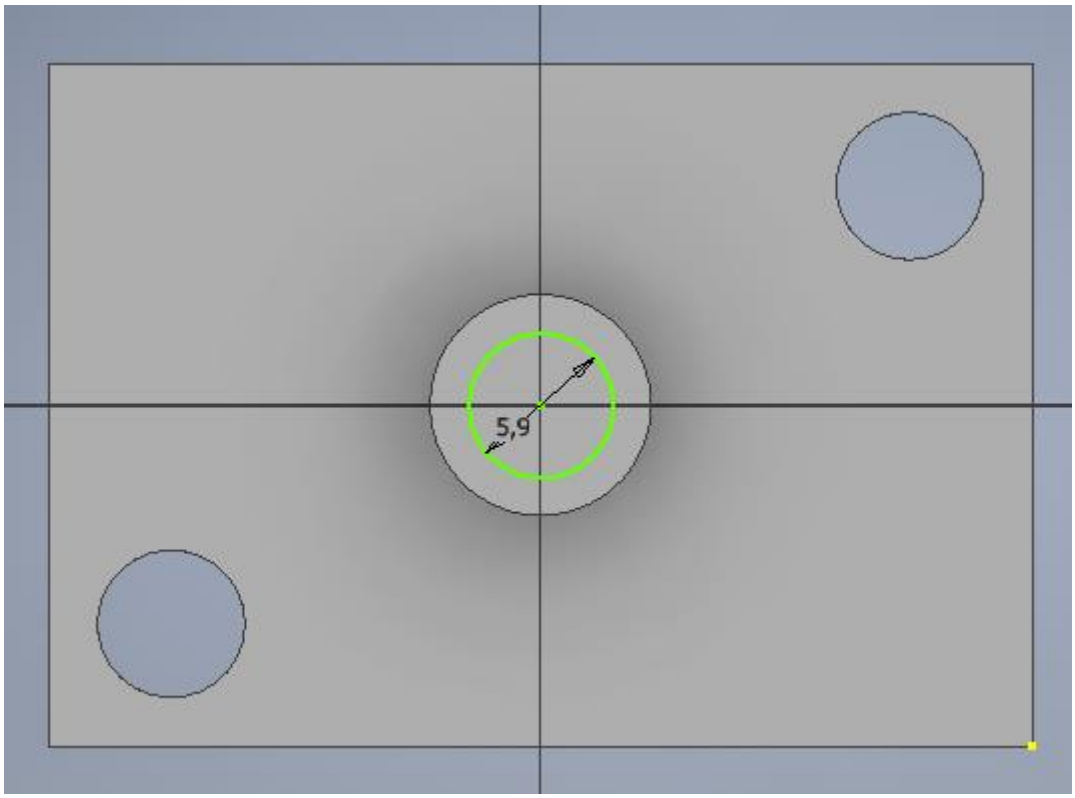


Figura 44: Sketch 7, cilindro interior.

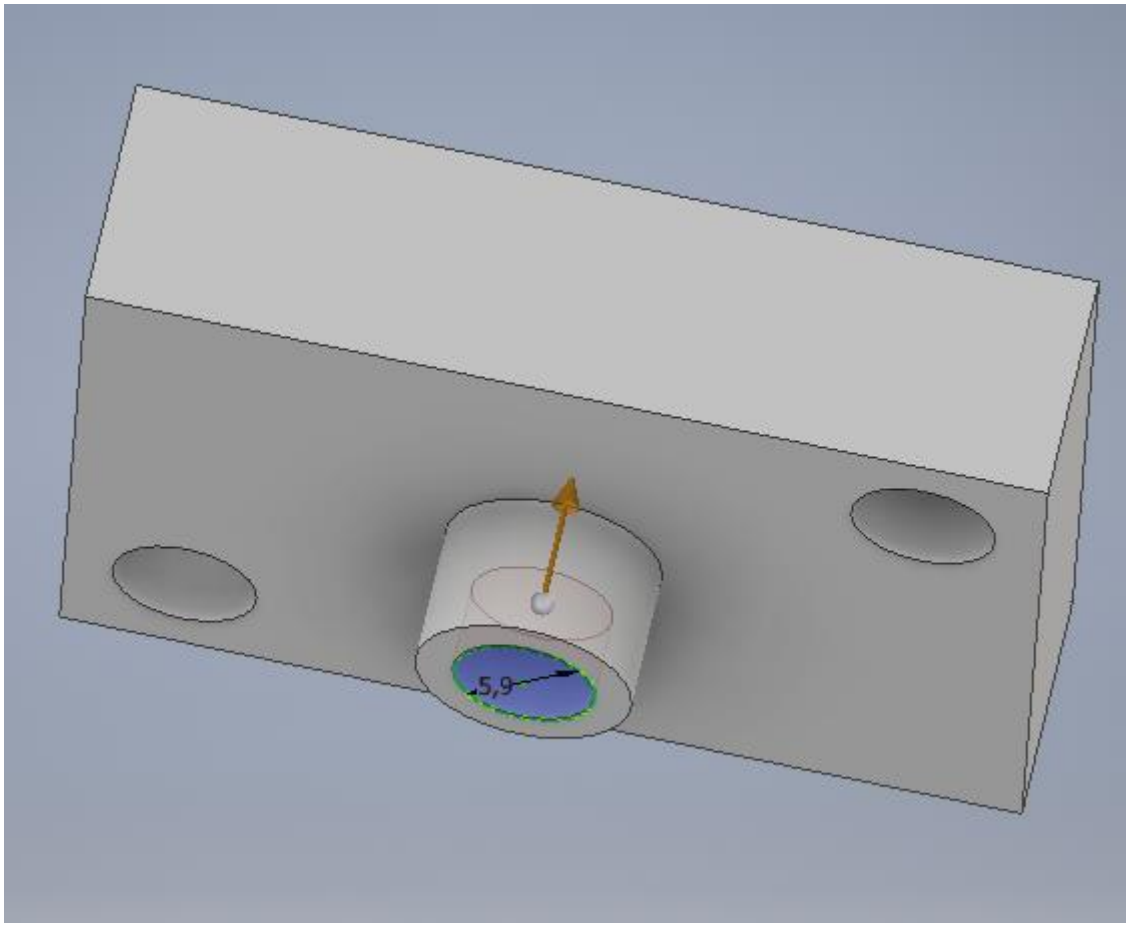


Figura 45: Extrusión 7, cilindro interior.

Con todo esto ya tenemos la pieza definitiva y la podemos mandar a imprimir a una impresora 3D ya que Inventor nos da muchas facilidades para esto. Simplemente se debe exportar a un archivo ".stl" para poder ser leído desde la impresora. A la hora de imprimir elegimos aligerar peso en esta pieza haciendo que por dentro estuviera parcialmente hueca, en nuestro caso solo el 20% de la pieza está conformada por material.

Los planos de la pieza se pueden ver en el apartado A.2. de este mismo anexo y en la siguiente imagen se muestra la pieza ya montada con los sensores y el motor incorporados.

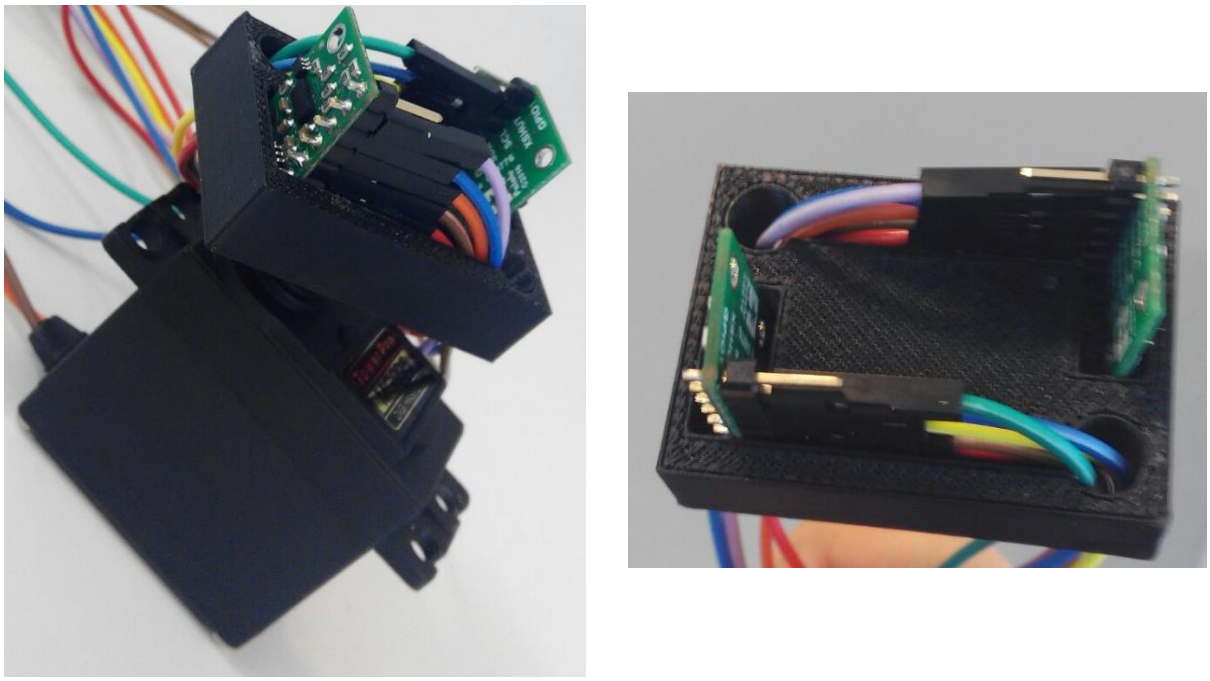
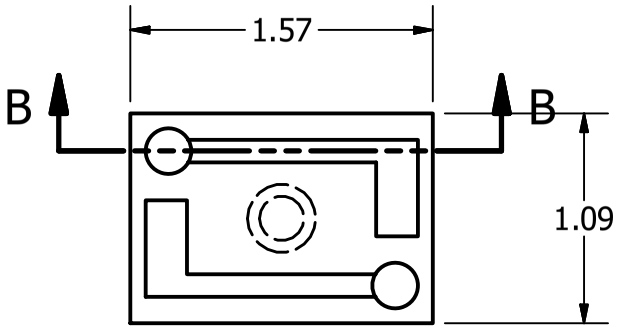


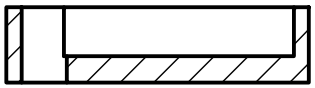
Figura 46: Resultado final de la pieza del motor.

A.2. Planos de la pieza

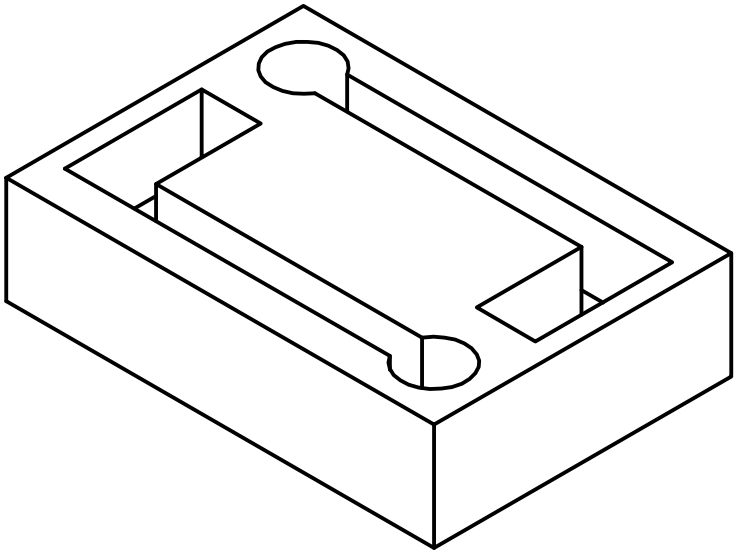
En este apartado se muestran los planos de la pieza creada que son proporcionados por el mismo programa Autodesk Inventor.

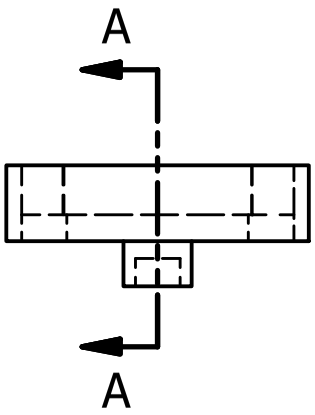
Podemos ver que las vistas de planta y alzado están a escala 1:1 y que la perspectiva isométrica de la pieza está dibujada a escala 2:1. En los propios planos se muestran las dimensiones más importantes, aunque estas están mucho más desglosadas en las figuras que muestran los sketches realizados para crear la pieza. También se han efectuado dos cortes, tanto en la planta como en el alzado, con la finalidad de ayudar a la comprensión de esta pieza.



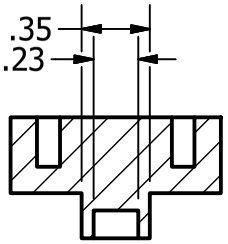


SECTION B-B
SCALE 1 : 1





SECTION A-A
SCALE 1 : 1



DRAWN: Lucía Morcillo Martínez		
TITLE: PIEZA MOTOR		
SIZE: A4		29/04/2018
SCALE: 1 : 1		SHEET 1 OF 1