Universidad Miguel Hernández de Elche

MASTER UNIVERSITARIO EN ROBÓTICA

“Desarrollo e implementación de un sistema LIDAR para escaneo y adquisición automática de entornos 3D”

Trabajo de Fin de Máster

Curso académico 2018/2019

Autor: Rubén Alcaraz Poblet
Tutor/es: David Valiente García

Tabla de Contenido

1. **Introducción** 1
   - **1.1. Justificación y Planteamiento del Problema** 1
   - **1.2. Objetivos** 1
   - **1.3. Resumen** 2
2. **Marco Teórico** 3
   - **2.1. Sistemas antecesores del lídar** 3
     - **2.1.1. Radar** 3
     - **2.1.2. Sónar** 4
     - **2.1.3. Láser** 5
     - **2.1.4. Radiación Infrarroja** 6
   - **2.2. Definición y Comienzos del Lídar** 7
   - **2.3. Lídar de Escaneo Monolínea** 8
   - **2.4. Lídar de Escaneo Multilínea** 9
   - **2.5. Lídar de No Escaneo tipo 3D Flash** 10
3. **Sistema de Medida Lídar** 15
4. **Diseño del Prototipo** 17
   - **4.1. Descripción de Componentes** 17
     - **4.1.1. Telémetro Láser LÍDAR-Lite 3** 17
     - **4.1.2. Telémetro Láser LÍDAR-Lite 3 de Alto Rendimiento (LLV3HP)** 18
     - **4.1.3. Arduino Uno R3** 18
     - **4.1.4. Lynxmotion Pan and Tilt Kit** 19
     - **4.1.5. Servomotores Hitec HS-422** 20
     - **4.1.6. Placa Freetronics con teclado y pantalla LCD compatible con Arduino** 20
     - **4.1.7. Placa de Potencia** 21
   - **4.2. Circuito Eléctrico** 23
   - **4.3. Sistema de Control** 24
5. **Implementación** 27
   - **5.1. Montaje** 27
   - **5.2. Algoritmo de Control** 31
     - **5.2.1. Arduino** 31
       - **5.2.1.1. Setup** 31
       - **5.2.1.2. Loop** 31

5.2.1.3. Iniciar_escaneo ... 32
5.2.1.4. MoveServos ... 33
5.2.1.5. DisplayPosition ... 33
5.2.1.6. UpdateModeDisplay ... 33
5.2.2. MatLab ... 33
5.2.2.1. Algoritmo Principal ... 33
5.2.2.2. GUI ... 34
5.2.2.3. Funciones Globales ... 36
5.2.2.3.1. Conexión ... 36
5.2.2.3.2. Desconexión ... 36
5.2.2.3.3. Filtrar ... 36
5.2.2.3.4. Dibujar ... 37
5.2.2.3.5. Precisión ... 37
5.2.2.3.6. Distancia ... 37
5.2.2.3.7. Notificar ... 38
5.2.2.4. Librería Point Cloud ... 38
6. Resultados ... 41
6.1. Telémetro Láser LÍDAR-Lite 3 ... 41
6.1.1. Escaneo ... 41
6.1.2. Precisión ... 45
6.2. Telémetro Láser LÍDAR-Lite 3 de Alto Rendimiento (LLV3HP) ... 46
6.2.1. Escaneo ... 46
6.2.2. Precisión ... 51
7. Conclusiones y Líneas Futuras ... 53
7.1. Conclusión ... 53
7.2. Trabajos Futuros ... 53
8. Bibliografía ... 55
9. Anexos ... 57
9.1. Código de Arduino ... 57
9.2. Código de MatLab ... 62
9.2.1. Init_lidar.m ... 62
9.2.2. LidarGUI.m ... 64
9.2.3. Funciones.m ... 74
9.3. Fichas Técnicas de los Componentes ... 76

Tabla de Ilustraciones

Ilustración 1: Radar de carretera que monitoriza la velocidad de los vehículos . 4
Ilustración 2: Esquema del funcionamiento de un sónar activo y otro pasivo 5
Ilustración 3: Apuntador emitiendo un haz de luz láser 6
Ilustración 4: Niveles de temperatura detectados en el cuerpo de un animal a través de la medida de la radiación infrarroja que emite 7
Ilustración 5: Principio de funcionamiento del lídar monolínea..... 8
Ilustración 6: Parámetros de algunos modelos de lídar de escaneo monolínea 9
Ilustración 7: Principio de funcionamiento del lídar multilínea 9
Ilustración 8: Parámetros de algunos modelos Velodyne de lídar de escaneo multilínea..... 10
Ilustración 9: Parámetros de algunos modelos SureStar de lídar de escaneo multilínea..... 10
Ilustración 10: Principio de funcionamiento del lídar 3D Flash 11
Ilustración 11: Estructura del lídar 3D Flash de LeddarTech..... 12
Ilustración 12: Prueba de uso del lídar de Benewake CE30 13
Ilustración 13: Esquema de funcionamiento del Tiempo de Vuelo (TOF) 15
Ilustración 14: Ciclo de señales generado por el sistema TOF 16
Ilustración 15: Lídar Lite 3 de la empresa Garmin..... 17
Ilustración 16: Lídar Lite 3 de alto rendimiento de la empresa Garmin 18
Ilustración 17: Placa de programación Arduino Uno R3..... 19
Ilustración 18: Estructura "Pan and Tilt" 20
Ilustración 19: Servomotor Hitec HS-422 20
Ilustración 20: Placa Freetronics con teclado y pantalla LCD compatible con Arduino..... 21
Ilustración 21: Placa de potencia..... 22
Ilustración 22: Esquema de la placa de potencia en Eagle 22
Ilustración 23: Esquema eléctrico del proyecto original..... 23
Ilustración 24: Esquema de control 24
Ilustración 25: Estructura lídar..... 28
Ilustración 26: Montaje intermedio del circuito eléctrico mediante placa de prototipado 29
Ilustración 27: Placas apiladas..... 30
Ilustración 28: Conector soldado a los pines del lídar 31
Ilustración 29: Interfaz Gráfico de MatLab (GUI) 35
Ilustración 30: Esquema del filtro de puntos..... 37

Ilustración 31: Visualización de una nube de puntos a través de la librería Point Cloud de MatLab 39
Ilustración 32: Escaneo de una pared con lídar de bajo rendimiento 41
Ilustración 33: Archivo PCD del escaneo de una pared con lídar de bajo rendimiento..... 42
Ilustración 34: Escaneo automático del entorno 3D con el dispositivo lídar de bajo rendimiento 43
Ilustración 35: Nube de puntos de la habitación con el dispositivo lídar de bajo rendimiento tras la exportación de los datos a formato PCD 44
Ilustración 36: Nube de puntos de la habitación con el dispositivo lídar de bajo rendimiento tras el filtrado de puntos con coeficiente 0,2..... 45
Ilustración 37: Gráfica que muestra la tendencia de la media de las medidas del lídar de bajo rendimiento 46
Ilustración 38: Escaneo de una pared con lídar de alto rendimiento 47
Ilustración 39: Archivo PCD del escaneo de una pared con lídar de alto rendimiento..... 48
Ilustración 40: Escaneo automático del entorno 3D con el dispositivo lídar de alto rendimiento 49
Ilustración 41: Nube de puntos de la habitación con el dispositivo lídar de alto rendimiento tras la exportación de los datos a formato PCD 50
Ilustración 42: Nube de puntos de la habitación con el dispositivo lídar de alto rendimiento tras el filtrado de puntos con coeficiente 0,2..... 51
Ilustración 43: Gráfica que muestra la tendencia de la media de las medidas del lídar de alto rendimiento 52

1. Introducción

1.1. Justificación y Planteamiento del Problema

Con las nuevas tecnologías, se han ido desarrollando nuevos modelos de dispositivos que se han aprovechado para cubrir las emergentes necesidades del mercado, derivando su uso en nuevas aplicaciones comerciales. Tal es el uso del lídar, dispositivo de medida telemétrico cuyas aplicaciones principales son la detección de objetos a diferentes distancias y escaneos del entorno tridimensional, formando una nube de puntos que posteriormente puede analizarse para detectar objetos, establecer rutas de navegación, etc.

El potencial del lídar ha sido llevado desde la más sencilla aplicación de evasión de obstáculos de la robótica móvil, hasta las más refinadas técnicas de análisis aéreo de cultivos, reconstrucción topográfica, aplicaciones policiales de detección de velocidades de vehículos, mapeado y guía de vehículos autónomos, y todo ello sin las desventajas que ofrecen las cámaras habituales, como por ejemplo la necesidad de una iluminación externa de la escena o el gasto computacional que requiere calcular la distancia a un objeto y la precisión de los píxeles, que disminuye enormemente cuanto más lejos se halla el objeto.

Por tanto, resulta interesante el uso e investigación de estos dispositivos, ya que cubren las necesidades que serían más difíciles de satisfacer a través de otros artefactos. Por ejemplo, el sensor de ultrasonidos no capta distancias tan grandes como el lídar ni con una precisión tan alta; además, el lídar es mucho más rápido que el radar, ya que su tecnología láser se aprovecha de la velocidad de la luz frente a la velocidad de propagación de las ondas electromagnéticas de radio.

1.2. Objetivos

El objetivo general de este proyecto es implementar el montaje del sistema y un algoritmo de control en Arduino que sea capaz de controlar los diferentes componentes del sistema lídar. Para ello, se han definido los siguientes objetivos específicos.

El primero de ellos, es crear un interfaz gráfico de usuario en MatLab a través del cual sea posible manejar la estructura de escaneo y cambiar sus parámetros de actuación. Además, también se pretende implantar la librería Point Cloud nativa de MatLab, que permite visualizar, cargar y guardar la nube de puntos escaneada en archivos PCD.

Una vez hecho esto, se pretende realizar un estudio de diferentes dispositivos lídar y comparar los resultados para averiguar el rendimiento de cada uno, y decidir si alguno de ellos es apto para su uso en una aplicación de escaneo estático de un entorno cerrado, entendiendo estático como una estructura de escaneo fija en un lugar del espacio.

1.3. Resumen

La memoria puede resumirse en los siguientes apartados:

- En el apartado 2 se introduce el marco teórico del lídar y se exponen algunos modelos ya desarrollados junto a sus parámetros de funcionamiento.

- En el apartado 3, se expone el principio de funcionamiento físico de los dispositivos tipo lídar, utilizando recursos gráficos y exponiendo las fórmulas utilizadas.

- En el apartado 4, se describen los componentes utilizados junto a sus parámetros técnicos y se realiza una breve explicación del circuito eléctrico. También, se expone el sistema de control implementado de forma general.

- En al apartado 5, se describe la implementación de los algoritmos de control utilizados, exponiendo uno por uno su funcionamiento y uso y clasificándolos en subapartados según su plataforma (Arduino MatLab) y el archivo en el que se localizan.

- En el apartado 6, se exponen los resultados experimentales de cada dispositivo lídar, siendo éstos tanto de la prueba de escaneo como de la de precisión.

- Finalmente, en el apartado 7 se comentan las conclusiones a las que se han llegado a través del experimento y se detallan posibles soluciones a los errores hallados y posibles líneas de experimentación futura.

2. Marco Teórico

A continuación, se realizará una definición del lídar y expondrán sus orígenes y distintas aplicaciones. Por otro lado, se pretenden describir distintos tipos de sistemas lídar actualmente en el mercado y mostrar algunos de sus parámetros competitivos.

2.1. Sistemas antecesores del lídar

Anteriormente al lídar, se utilizaban otro tipo de dispositivos de medición de diferentes características que proporcionaban buena cobertura para unos entornos, pero deficiencias en otros.

2.1.1. Radar

El radar (***ra****dio* ***d****etection* ***a****nd* ***r****anging*) es un dispositivo telemétrico emisor de ondas electromagnéticas que mide distancias, tales como altitudes, direcciones y velocidades de objetos estáticos o móviles [1]. Su funcionamiento se consiste en emitir un impulso de radiofrecuencia que se refleja en el objetivo y cuyo eco se recibe típicamente en la misma posición del emisor.

Inicialmente en 1904, el radar comenzó a utilizarse con fines de navegación para evitar colisiones con la superficie de la corteza terrestre, pero las tensiones bélicas anteriores a la Segunda Guerra Mundial, forzó a muchas naciones a desarrollar a lo largo del siglo XX aplicaciones de detección de objetivos utilizando, para ello, la tecnología de los radares. Así surgieron los sistemas de detección antiaérea, que permitirían detectar de forma rápida y fugaz aeronaves enemigas.

Hoy en día su uso está mucho más extendido, pues pueden encontrarse en aplicaciones como la vigilancia de los límites de velocidad en carreteras, detección de precipitaciones o incluso el control del tráfico aéreo.

Ilustración 1: Radar de carretera que monitoriza la velocidad de los vehículos

2.1.2. Sónar

El sónar (***so****und* ***n****avigation* ***a****nd* ***r****anging*) es una técnica que utiliza la propagación del sonido bajo el agua principalmente para la navegación, comunicarse o detectar objetos sumergidos [2]. Puede usarse como medio de localización acústica, funcionando de forma similar al radar, con la diferencia de que en lugar de emitir ondas electromagnéticas emplea impulsos sonoros subacuáticos.

Existen de dos tipos: activo y pasivo. El primero de ellos utiliza un emisor de sonido y un receptor, de modo que puede medir el tiempo que tarda el impulso en volver al receptor desde que es emitido por el emisor, mientras que el segundo detecta sin emitir.

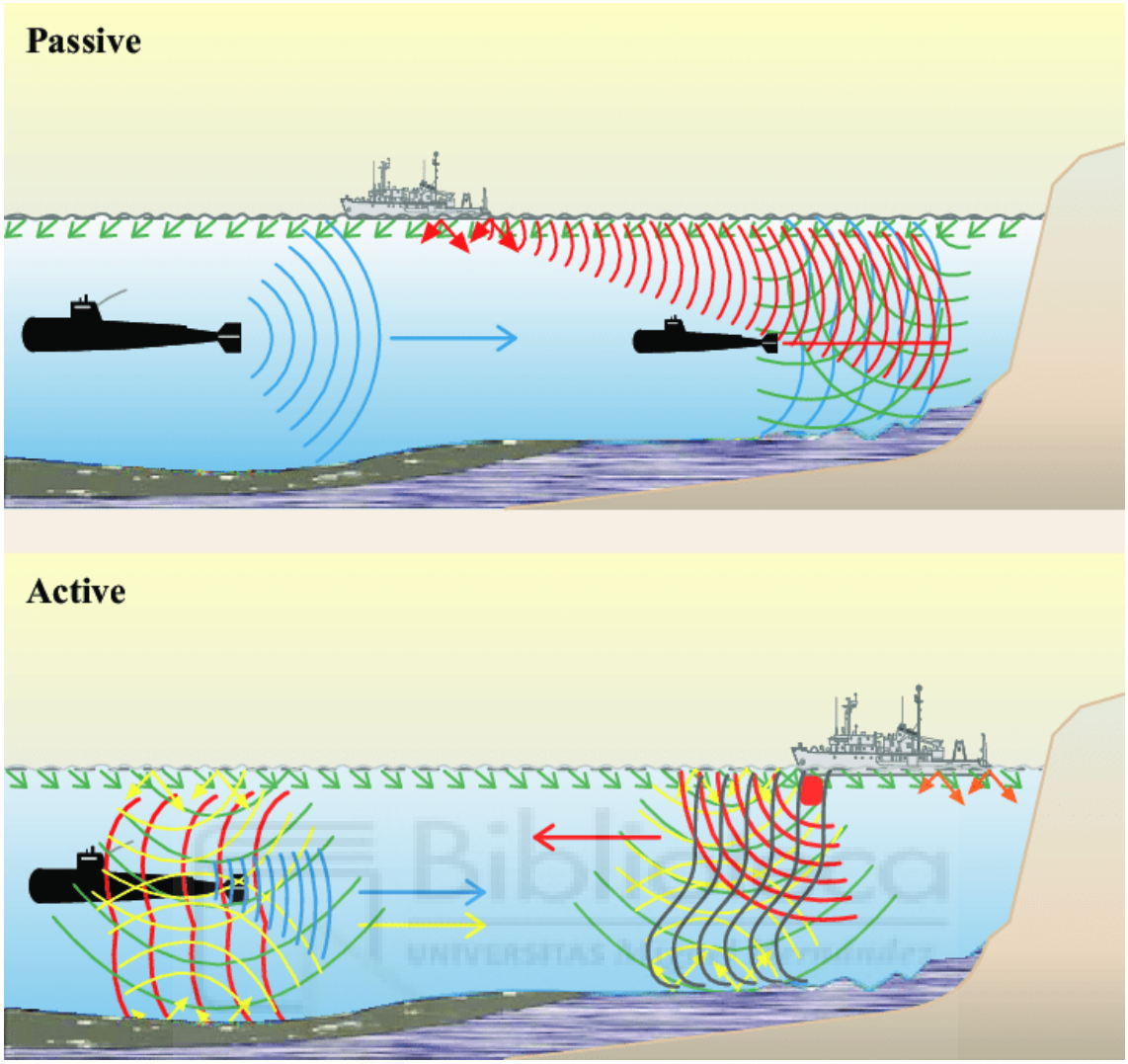


Ilustración 2: Esquema del funcionamiento de un sónar activo y otro pasivo

El primer uso del sónar fue registrado por vez primera por Leonardo da Vinci en 1490, de quien se decía que usaba un tubo metido en el agua para detectar barcos, poniendo un oído en su extremo. Por otro lado, en el siglo XIX se le dio la primera aplicación de comunicación, utilizando campanas subacuáticas como complemento a los faros para avisar del peligro a los marineros.

El uso de la «ecolocalización» submarina parece haber sido impulsado por el desastre del Titanic en 1912, mientras que, durante la Primera Guerra Mundial, y debido a la necesidad de detectar a los submarinos alemanes, se generalizó su aplicación en el ámbito militar.

Al sónar además se le deben atribuir ciertas aplicaciones aparte de las militares como, por ejemplo: cálculo de profundidades, cálculo de velocidades de buques, estimaciones de biomasa, medida de olas o arqueología subacuática, entre otras.

2.1.3. Láser

Un láser (**l**ight **a**mplification by **s**timulated **e**mission of **r**adiation) es un dispositivo que utiliza la emisión inducida o estimulada para generar un haz de luz coherente tanto espacial como temporalmente [3]. La coherencia espacial se corresponde con la capacidad de un haz para permanecer con un pequeño tamaño al transmitirse por el vacío en largas distancias, es decir, disiparse poco a poco manteniendo el mismo grosor del haz durante largas distancias, y la coherencia temporal se relaciona con la capacidad para concentrar la emisión en un rango espectral muy estrecho, es decir, con una variación muy pequeña de la longitud de onda del haz (tono y color constantes). Esta tecnología es implementada por los lídar para emitir un haz de luz unidireccional que impacte contra el objetivo, y cuya onda de regreso pueda ser registrada por el mismo.

Ilustración 3: Apuntador emitiendo un haz de luz láser

El primer láser fue construido con rubí por Theodore Maiman y funcionó por primera vez el 16 de mayo de 1960. Dos años después, Robert Hall inventa el láser generado por semiconductor y en 1969 se encuentra su primera aplicación industrial al ser utilizado en las soldaduras de los elementos de chapa en la fabricación de vehículos. Al año siguiente Gordon Gould patenta otras muchas aplicaciones prácticas para el láser.

Actualmente, los láseres son empleados en muchas más aplicaciones, tales como: lectura y almacenamiento de datos, topografía y alcance, industria textil, espectroscopia o escáner de código de barras, entre otras.

2.1.4. Radiación Infrarroja

La radiación infrarroja, o radiación IR, es un tipo de radiación electromagnética de mayor longitud de onda que la luz visible, por lo que es imposible de ser percibida por el ojo humano, pero menor que la de las microondas. Su rango de longitudes de onda va desde unos 0,7 hasta los 1000 micrómetros [4]. La radiación infrarroja es emitida por cualquier cuerpo cuya temperatura sea mayor que 0 Kelvin, es decir, −273,15 grados Celsius (cero absoluto). El lídar de este proyecto utiliza, junto al emisor láser, este tipo de radiación para que sea menos sensible a la luz visible.

Ilustración 4: Niveles de temperatura detectados en el cuerpo de un animal a través de la medida de la radiación infrarroja que emite

Los infrarrojos fueron descubiertos en 1800 por William Herschel, quien colocó un termómetro de mercurio en el espectro obtenido por un prisma de cristal con el fin de medir el calor emitido por cada color. Descubrió que el calor era más fuerte al lado del rojo del espectro y observó que allí no había luz. Esta es la primera experiencia que muestra que el calor puede transmitirse por una forma invisible de luz.

La medición de este fenómeno puede hallarse en múltiples aplicaciones, tales como: visión nocturna, comunicaciones a corta distancia (como por ejemplo el mando de televisión), comunicación de alta velocidad (fibra óptica) y astronomía.

2.2. Definición y Comienzos del Lídar

El lídar (*LIght Detection And Ranging*) puede definirse como una técnica de teledetección óptica, que utiliza la luz de láser para obtener una muestra densa de objetos y superficies produciendo mediciones exactas en coordenadas cartesianas respecto de un sistema local [5].

Fue utilizado por primera vez en 1971 para crear mapas de La Luna durante la misión del Apollo 15 aprovechándose de la metodología TOF (*Time of Flight*), y más tarde usado para arqueología y arquitectura. Su primera aparición en el campo de los vehículos autónomos fue en 2005, durante el reto de conducción automática DARPA. Además de la medición lunar, el método TOF se emplea

principalmente en mediciones topográficas, detección de obstáculos frontales, rastreo de misiles guiados, medición láser mediante lídar, y mediciones de distancia de tierra y satélites artificiales, etc.

Hoy en día, existen muchas empresas que dedican sus recursos a la investigación y optimización de los aparatos lídar tales como Velodyne, Quanergy, Innovusion, Ibeo, SICK, SureStar, Hesai Technology, Benewake, y Leishen Intelligent Technology, entre otras.

2.3. Lídar de Escaneo Monolínea

El lídar es originario de inicios de la década de 1960, poco después de la invención del láser, combinando éste con la capacidad de calcular el tiempo que tarda en ir y volver de un objeto [6]. Fue utilizado inicialmente por el Centro Nacional de Investigación Atmosférica (NCAR) de los Estados Unidos para medir las nubes, pero actualmente su uso se ha extendido a muchísimos campos distintos, entre ellos la conducción de vehículos autónomos.

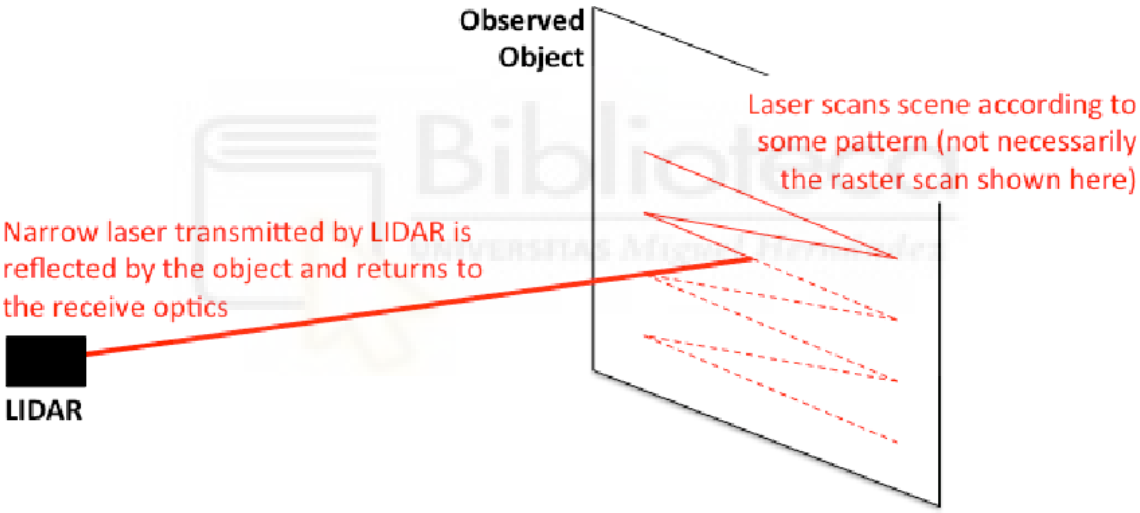


Ilustración 5: Principio de funcionamiento del lídar monolínea

A continuación se presenta una tabla de parámetros de los principales productos y fabricantes de lídar de escaneo de una sola línea [7] en la Ilustración 6.

type	LS1XX	LS2XX	LS3XX	LS4XX	UTM-30LX	TiM561
Scanning angle	270°					
Angle resolution	0.25°/0.125°/0.0625°				0.25°	0.33°
Distance range(m)	0.5to50	0.5to100	0.5~8	0.5~20	0.1 ~30	0.05~10
10% reflectance maximum distance	30m	50m	40m	10m	0.1 to 30m	8m
producer	Osight	Osight	Osight	Osight	HOKUYO	SICK

Ilustración 6: Parámetros de algunos modelos de lídar de escaneo monolínea

Como puede apreciarse, la empresa china Osight presenta ciertas ventajas en relación a la competencia, sobretodo en relación al rango de distancias y la resolución angular.

2.4. Lídar de Escaneo Multilínea

Por otro lado, el lídar de escaneo multilínea ha seguido un desarrollo caracterizado por múltiples haces de luz que rastrean un mismo entorno a distintas altitudes, para así crear mapas tridimensionales más completos englobados por un mayor número de planos a distintas alturas del entorno. Así pues, destacan en la creación de este tipo de productos empresas como Velodyne [8] y SureStar [9] entre otras.

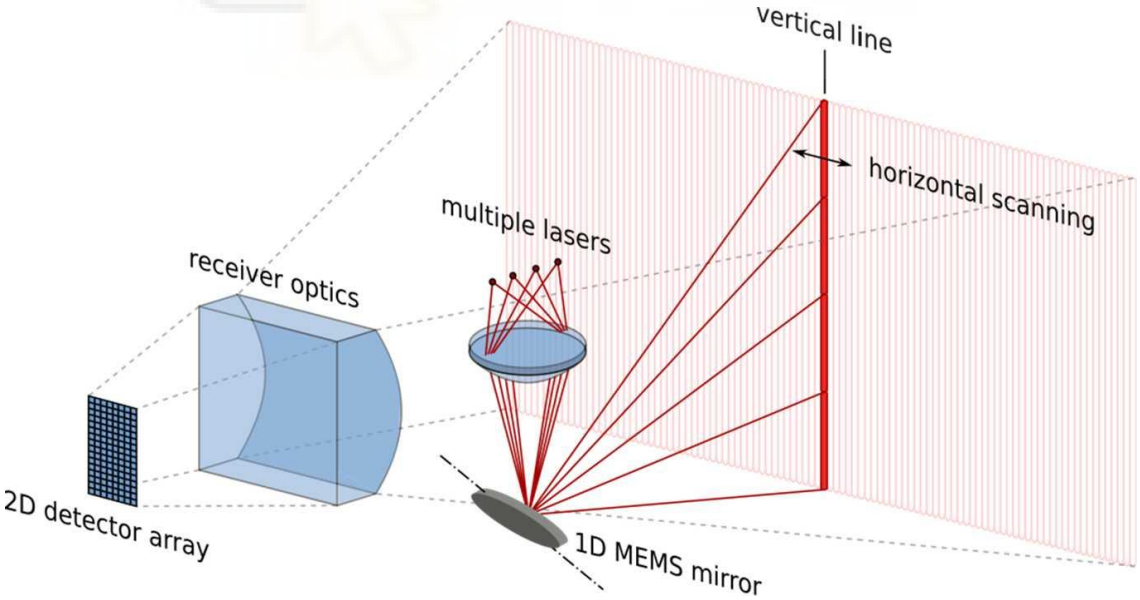


Ilustración 7: Principio de funcionamiento del lídar multilínea

En 2007, Velodyne sacó al mercado un lídar de 64 líneas, que fue primero utilizado en los vehículos autónomos de Google [7]. Más tarde, produjeron lidars de 16 y 32 líneas, que fueron ampliamente utilizados en pilotaje automático,

rastreo del terreno y estudios agrícolas y geográficos, entre otros. Finalmente, en 2018 se desarrolló un nuevo lídar de 128 líneas.

type	distance	Output points(s)	FOV(°)	angle resolution(°)	ranging accuracy(cm)
VLP-16	100m	300,000	360*30	0.1~0.4*2	±3
HDL-32E	70m	300,000	360*41.34	0.16* 1.33	<2
HDL-64E	120m	1,330,000	360*40	0.1*0.4	2
VLS-128	300m	2,200,000	360*40	0.1* 0.1	

Ilustración 8: Parámetros de algunos modelos Velodyne de lídar de escaneo multilínea

La Ilustración 8 presenta las características de algunos de los productos lídar multilínea de la marca Velodyne [7]. Como puede apreciarse, se destaca la mejora de la exactitud del escaneo y de la resolución angular. Estos atributos permiten localizar objetos cada vez más pequeños de forma precisa en aplicaciones de vehículos autónomos. No obstante, las series de lídar de Velodyne son tan caras (el modelo HDL-64 cuesta 70 mil dólares) que la mayoría de la gente no puede permitírselas.

En 2016, SureStar sacó al mercado su primer lídar de escaneo de 16 líneas R-Fans-16. Más tarde, en 2017 comercializó los modelos de 32 líneas R-Fans-32 y R-Fans-32G y la serie de 64 líneas C-Fans [7]. Actualmente, la empresa está investigando los lídar flash 3D.

type	distance	scanning line spacing(°)	FOV(°)	angle resolution(H°)	ranging accuracy(cm)
R-Fans-16	200m	1/2	360×24	<0.1	<3
R-Fans-32	200m	1°	360*31	<0.1	<3
R-Fans-32-G	200m	0.3/1/2/3	360*40	<0.1	<3
R-Fans-128	200m	0.23&0.46(VAR)	150*30	0.05	<2

Ilustración 9: Parámetros de algunos modelos SureStar de lídar de escaneo multilínea

De la Ilustración 9, puede deducirse que conforme crece el espacio entre líneas de escaneo (*scanning line spacing*) la resolución angular horizontal mejora y además manteniendo un precio aceptable. No obstante, en comparación con Velodyne existe todavía una gran diferencia respecto a la resolución angular tanto vertical como horizontal.

2.5. Lídar de No Escaneo tipo 3D Flash

Aparte de los tipos de lídar de escaneo, se han desarrollado otros que no requieren de elementos rotacionales mecánicos, clasificados, así como lídar de no escaneo. De entre ellos, cabe destacar el lídar tipo 3D flash, desarrollado por empresas como Benewake [10], Continental [11], GuangBo o LeddarTech [12].

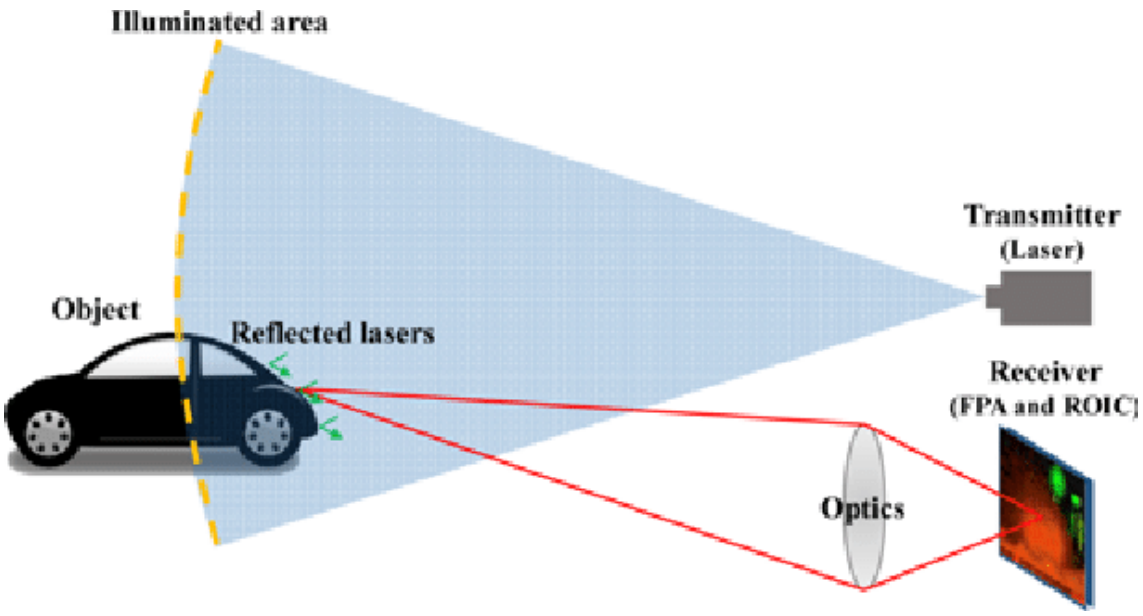


Ilustración 10: Principio de funcionamiento del lídar 3D Flash

En 2017, la empresa alemana Continental mostró su prototipo de lídar 3D Flash conocido como HFL. Este dispositivo utiliza una fuente láser de 1.064 nm y puede medir distancias de hasta 327 m con una exactitud de 0,04 m, ángulo de visión de 145º (H) X 3,2º (V) y una resolución angular de 0,25º y 4 capas de 0,8º, respectivamente. Se estima un error de rango menor a 0,1 m y de 0,25 m/s de velocidad. Este proyecto ha supuesto un éxito en vehículos como los Audi A8, A7 y A6, y se prevé su producción en masa para 2020.

En octubre de 2017 Guangpo sacó al mercado el GP003, utilizando para ello la tecnología "array de estado sólido" aplicada principalmente en robótica y vehículos autónomos. Este dispositivo es capaz de medir hasta una distancia de 150 m con un ángulo de visión de 360º (H) X 31º (V) cuando mide distancias de entre 0,3 y 20 m, y de 25º X 16º cuando cubre un rango de 20 a 100 m. La compañía se plantea utilizar este tipo de lídar para aplicaciones de seguridad, robots de reparto, UAVs (*Unmanned Air Vehicles*), piloto automático y muchas otras áreas.

Más tarde, en 2018, la empresa canadiense LeddarTech mostró su primer chip de lídar de estado sólido 3D, el cual puede verse instalado en la placa de la Ilustración 11.

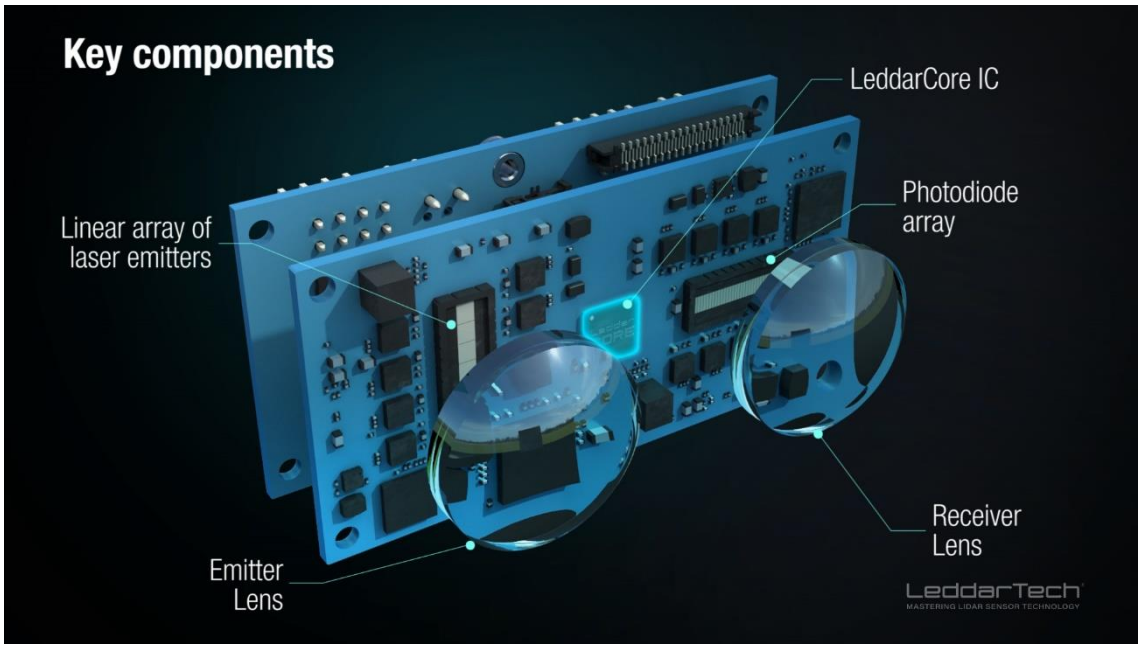


Ilustración 11: Estructura del lídar 3D Flash de LeddarTech

El mismo año, Leishen Intelligent System anunció una futura colaboración con la compañía israelí NEWSIGHT para investigar y desarrollar el V-lídar. Se prevé que este dispositivo tenga un alcance de 200 m y cumpla con los requisitos regulatorios. Tal y como se conoce hasta ahora, ya hay cinco o más empresas involucradas en el desarrollo del V-lídar.

Durante 2017, Benewake lanzó al mercado el lídar 3D flash modelo CE-30D, que fue satisfactoriamente probado por los vehículos de Audi. A pesar de lo complejo que resulta detectar objetos a larga distancia y a altas velocidades en un vehículo de conducción automática, ello no le impide rastrear un entorno con relativa exactitud dentro de su rango máximo de 30 m. También, es capaz de realizar una sencilla función ADAS tal como encendido/apagado automático o detección de punto ciego de otro vehículo en un camino transitado.

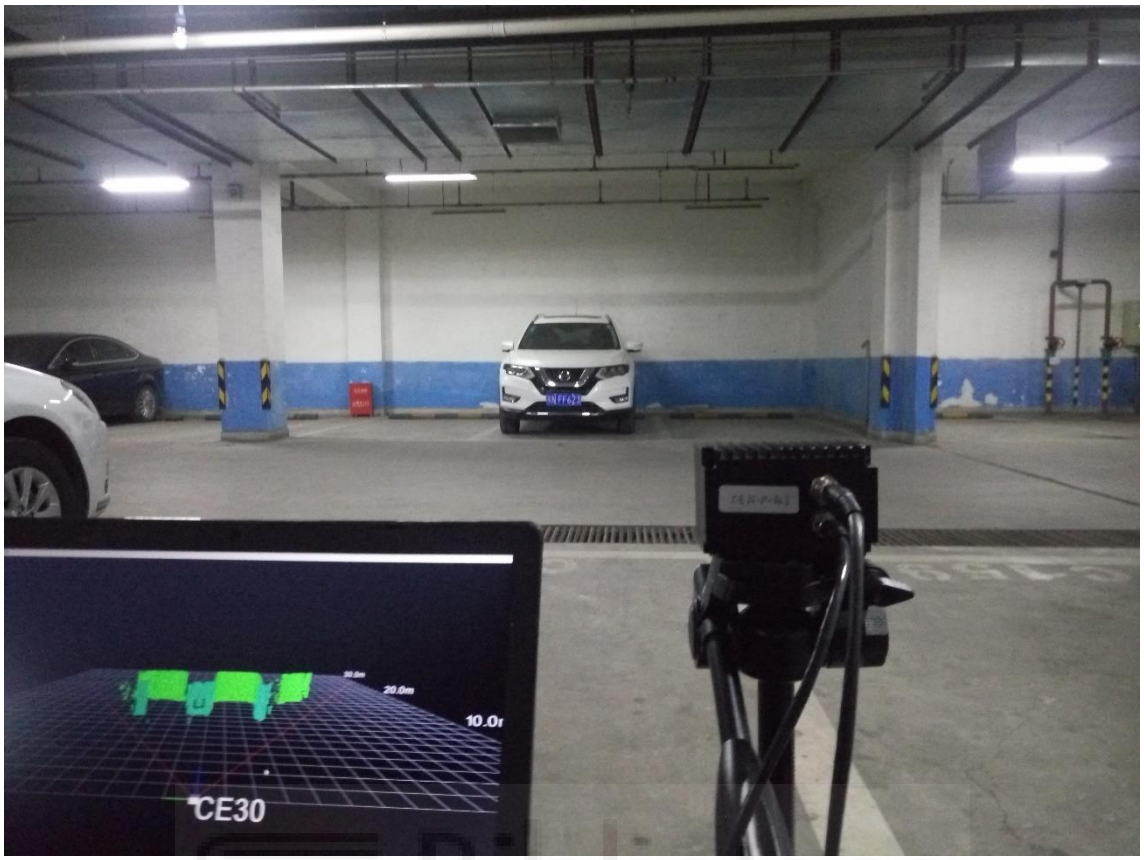


Ilustración 12: Prueba de uso del lídar de Benewake CE30

3. Sistema de Medida Lídar

A diferencia del funcionamiento de un láser habitual, el lídar hace uso de los principios físicos de éste para rastrear el entorno desplazándose angularmente y entregar los datos espaciales de un solo punto (un haz láser) distinto cada vez, ya sea a través de elementos mecánicos o no (3D Flash). El sistema de medición del lídar se fundamenta en el principio del tiempo de vuelo o TOF (*Time of Flight*), que consiste principalmente en la emisión de un pulso láser hacia un objeto y calcular el tiempo que tarda en volver desde que es emitido; es decir, el tiempo en el que el haz de luz se transmite por el entorno desde que es emitido por el dispositivo hasta que vuelve al receptor [13]. Se trata del funcionamiento más extendido para un lídar aplicado a cualquier tipo de vehículo autónomo o robot móvil. Un circuito interno se encarga de contar el tiempo Δt que tarda en volver el haz láser desde que es emitido, y así, calcular la distancia que ha recorrido [14].

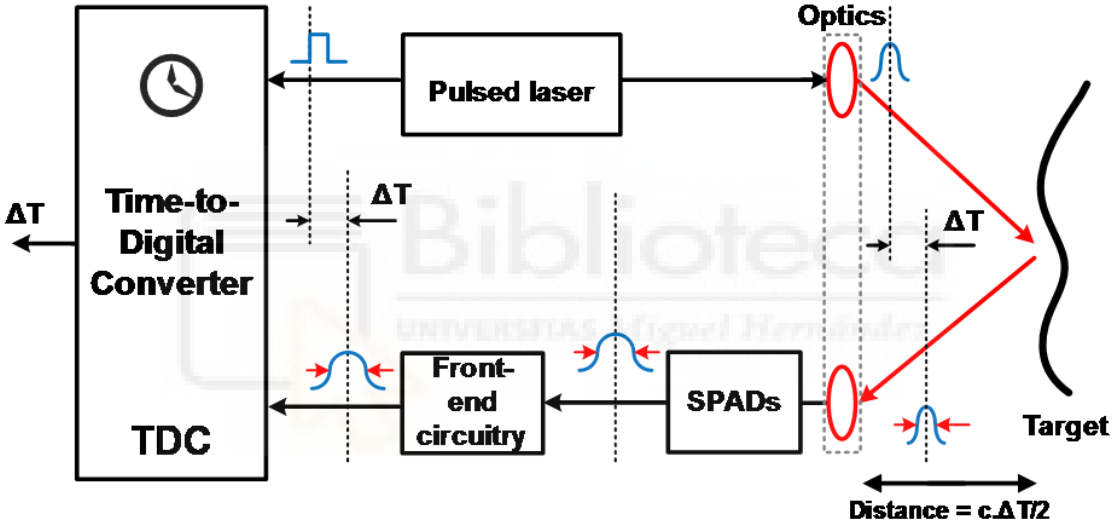


Ilustración 13: Esquema de funcionamiento del Tiempo de Vuelo (TOF)

El proceso de funcionamiento se produce tal cual se describe en la Ilustración 13. Primero, empieza a emitirse el pulso de luz hacia el objeto en cuestión, atravesando así un muestreador que activa un circuito contador. Posteriormente, el haz luminoso impacta contra el obstáculo y vuelve hacia el sensor, atravesando así el detector fotoeléctrico y produciendo un pulso eléctrico, amplificado por el amplificador, que sirve para indicar al contador que pare de contar. El cálculo de la distancia se describe en la Ilustración 14.

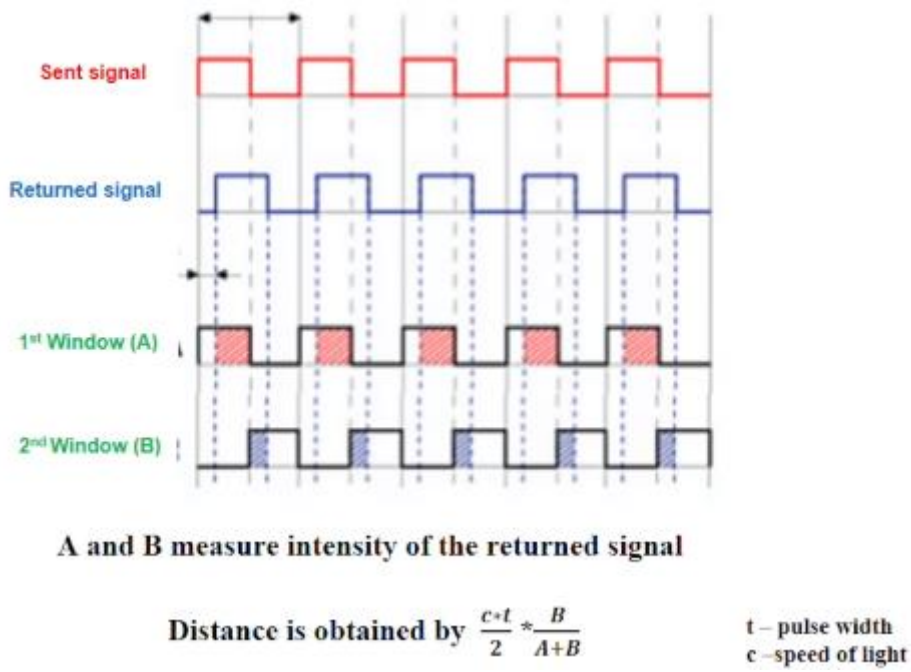


Ilustración 14: Ciclo de señales generado por el sistema TOF

Los factores principales de la precisión de esta medida son el flanco ascendente del pulso láser, el ancho de banda del canal receptor, el factor de ruido del detector y la precisión del intervalo de tiempo.

4. Diseño del Prototipo

El presente proyecto se inició en base a otro proyecto ya expuesto [15]. En este apartado se proponen algunas modificaciones y mejoras respecto del proyecto original y una descripción general de sus componentes, el circuito eléctrico y el sistema de control.

4.1. Descripción de Componentes

A continuación, se procede a la descripción de cada uno de los componentes utilizados en el presente proyecto.

4.1.1. Telémetro Láser LÍDAR-Lite 3

Se trata del componente principal del proyecto, mostrado en la Ilustración 15, el cual realiza mediciones de distancia a través del sistema de medida del tiempo de vuelo (TOF) descrito en el apartado 3, con un alcance de hasta 40 metros en objetos con un mínimo del 70% de reflectividad.

Ilustración 15: Lídar Lite 3 de la empresa Garmin

Este dispositivo depende de una alimentación de corriente continua de entre 4,5 y 5,5 V, siendo ésta de 105 mA estando parado y 135 mA en operación continua. Posee una resolución de espacial de ±1 cm con una precisión de ±2,5 cm a menos de 5 metros, siendo esta no lineal si se trata de una distancia menor a 1 metro, y de ±10 cm a una distancia igual o mayor a 5 metros. Además, su frecuencia de muestreo típica es de 270 Hz siendo de 650 Hz en modo rápido, con reducida sensibilidad, y mayor a 1000 Hz sólo aplicable a rangos cortos.

Además, posee un interfaz de comunicación I2C que funciona a una velocidad de 400 kbit/s en modo rápido y un interfaz PWM. El láser posee una longitud de onda de 905 nm (infrarrojo) con picos de potencia de hasta 1,3 W, cuya duración del pulso es de 0,5 µs.

4.1.2. Telémetro Láser LÍDAR-Lite 3 de Alto Rendimiento (LLV3HP)

Se trata de una versión de mayor rendimiento que el componente anterior, mostrado en la Ilustración 16, el cual realiza mediciones de distancia a través del sistema de medida del tiempo de vuelo (TOF) descrito en el apartado 3, con un alcance de hasta 40 metros en objetos con un mínimo del 70% de reflectividad.

Ilustración 16: Lídar Lite 3 de alto rendimiento de la empresa Garmin

Este dispositivo es resistente al agua, con catalogación IPX7, pudiendo sumergirse hasta 1 metro bajo el agua durante un tiempo máximo de 30 minutos. Depende de una alimentación de corriente continua de entre 4,5 y 5,5 V, siendo ésta de 65 mA estando parado y 85 mA durante la adquisición de un dato. Posee una resolución de espacial de ±1 cm con una precisión de ±5 cm a menos de 2 metros, siendo esta no lineal si se trata de una distancia menor a 1 metro, y de ±2,5 cm a una distancia igual o superior a 5 metros. Además, su frecuencia de muestreo típica es mayor a 1 kHz.

Por otro lado, posee un interfaz de comunicación I2C que funciona a una velocidad de 400 kbit/s en modo rápido y un interfaz PWM. El láser posee una longitud de onda de 905 nm (infrarrojo) con picos de potencia de hasta 1,3 W, cuya duración del pulso es de 0,5 µs.

4.1.3. Arduino Uno R3

A continuación, se presenta el dispositivo de control programable utilizado. Arduino es una plataforma de *software* libre muy extendida alrededor del mundo, con una potencia de cálculo suficiente para el presente proyecto y de bajo coste, cuya placa se muestra en la Ilustración 17.

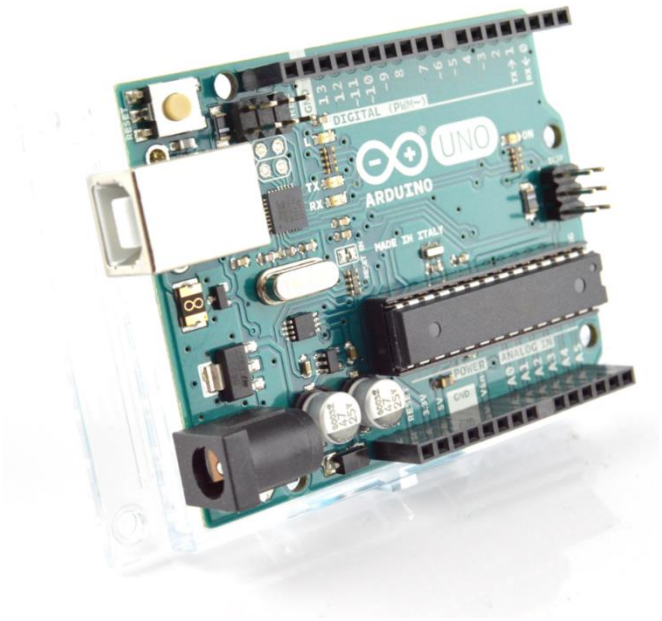


Ilustración 17: Placa de programación Arduino Uno R3

Lleva instalado un chip programable modelo ATMega 328 P del fabricante ATmel, que contiene una memoria flash de 32 KB de los cuales 0,5 KB son usados por el *bootloader*, SRAM de 2 KB y EEPROM de 1 KB [16]. Dispone de un conector de puerto serie hembra con un cable que lo adapta a conector USB macho, 14 entradas y salidas digitales de las cuales 6 pueden ser utilizadas para PWM, 6 entradas analógicas, un oscilador de 16 MHz, puerto Jack de alimentación, un ICSP header, un microLED incorporado (pin 13) y un botón de reinicio.

Su tensión de alimentación recomendada oscila entre 7 y 12 V, siendo el límite entre 6 y 20 V y la tensión de operación de 5 V. Además, esta placa puede ser alimentada a través del conector Jack, del pin Vin, y del conector del puerto serie. También posee dos pines integrados para alimentar a otros componentes, siendo estos de 5 V y de 3,3 V.

4.1.4. Lynxmotion Pan and Tilt Kit

Ilustración 18: Estructura "Pan and Tilt"

Estructura *Pan and Tilt* de Lynxmotion fabricada en aluminio con dos grados de libertad (horizontal y vertical) y dos servomotores Hitec HS-422 acoplados, mostrada en la Ilustración 18.

4.1.5. Servomotores Hitec HS-422

Ilustración 19: Servomotor Hitec HS-422

2 servomotores acoplados a la estructura *Pan and Tilt* con el objetivo de añadir los 2 grados de libertad (horizontal y vertical) al conjunto. Utilizan un sistema de control vía PWM y poseen un rango de alimentación de entre 4,8 V y 6 V, siendo la corriente de 8 mA cuando están quietos y de 150 mA en movimiento sin carga. Su rango de acción es de 0º a 180º, siendo 90º el valor de orientación frontal (mirando hacia adelante). Además, su velocidad de operación sin carga y par pico varían dependiendo de su alimentación, de 0,21 segundos en recorrer 60º a 4,8 V y 3.3 kg*cm de par pico, hasta 0,16 segundos alimentado a 6 V y 4.1 kg*cm de par pico; en este caso, se supone una alimentación de 5 V. El dispositivo se muestra en la Ilustración 19.

4.1.6. Placa Freetronics con teclado y pantalla LCD compatible con Arduino

Ilustración 20: Placa Freetronics con teclado y pantalla LCD compatible con Arduino

Esta placa, tal y como se muestra en la Ilustración 20, está adaptada a la configuración de pines de Arduino y tiene instalada una placa LCD de dos líneas de texto y una botonera de 5 botones con salida analógica. La señal analógica de salida de estos botones se debe a que todos comparten la misma salida y, por tanto, se establece un comportamiento jerárquico mediante un circuito analógico que establece una prioridad distinta para cada botón (cadena de resistencias), generando así distintas tensiones de referencia para la salida. El componente requiere de una alimentación de 5 V.

4.1.7. Placa de Potencia

Esta placa ha sido fabricada aparte para incluir algunos otros componentes de potencia necesarios para la alimentación del resto de dispositivos. Este aparato se alimenta a través de un conector Jack de barril de 2,1 mm, cuya tensión recomendada es de entre 7,5 V y 12,5 V, e integra un diodo para alimentar la placa Arduino de forma unilateral. Además, posee un regulador de tensión de 5 V modelo LM78M05CT para alimentar a los servomotores. También, se disponen de 2 condensadores electrolíticos de 1 mF y 25 V, siendo éstos uno para la alimentación de los servomotores y otro para el lídar.

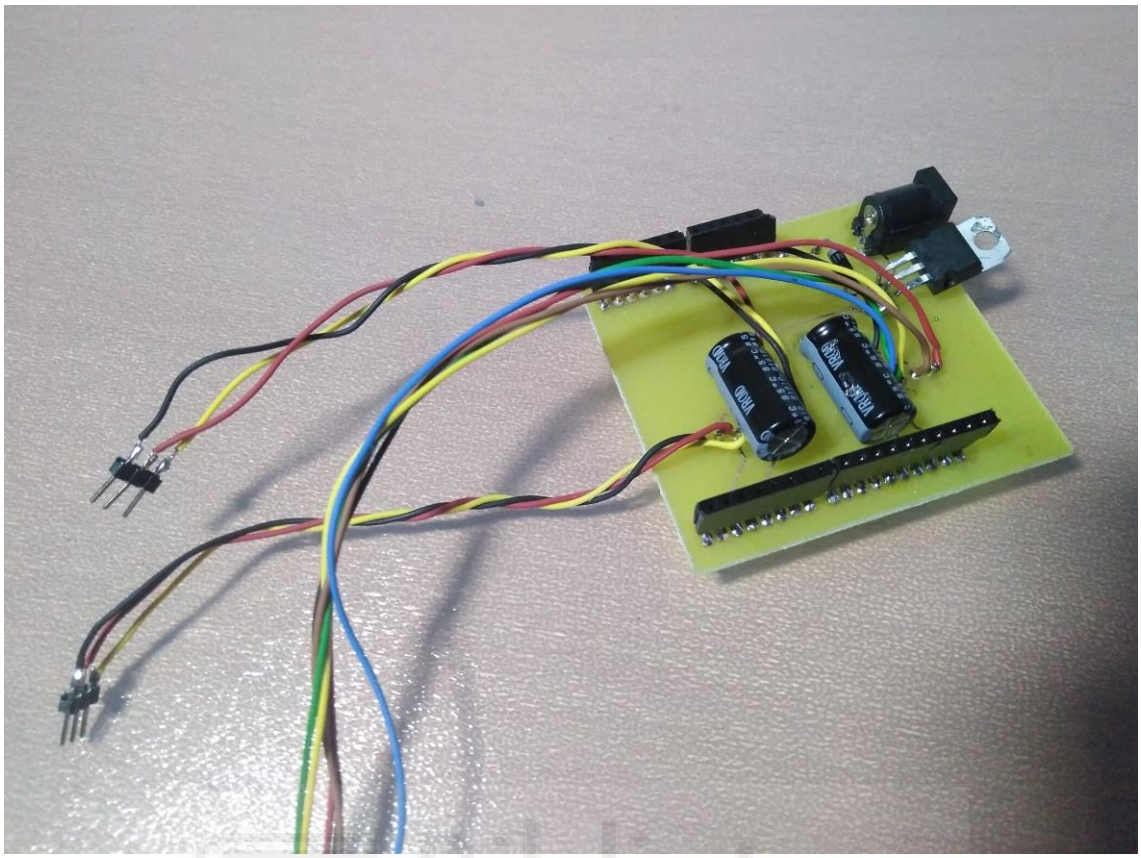


Ilustración 21: Placa de potencia

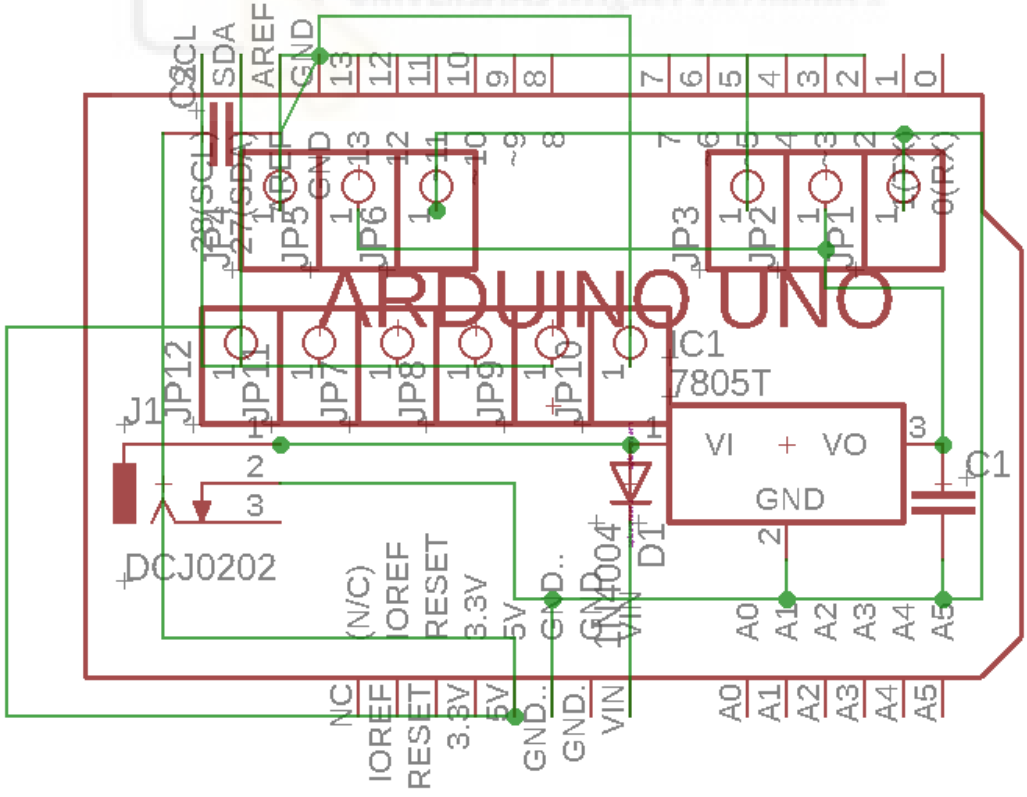


Ilustración 22: Esquema de la placa de potencia en Eagle

4.2. Circuito Eléctrico

A continuación, se presenta en la Ilustración 23 el esquema eléctrico del proyecto original; en el cual, el regulador de tensión, diodo, y los condensadores se han incluido en la placa de potencia descrita en el apartado 4.1.7.

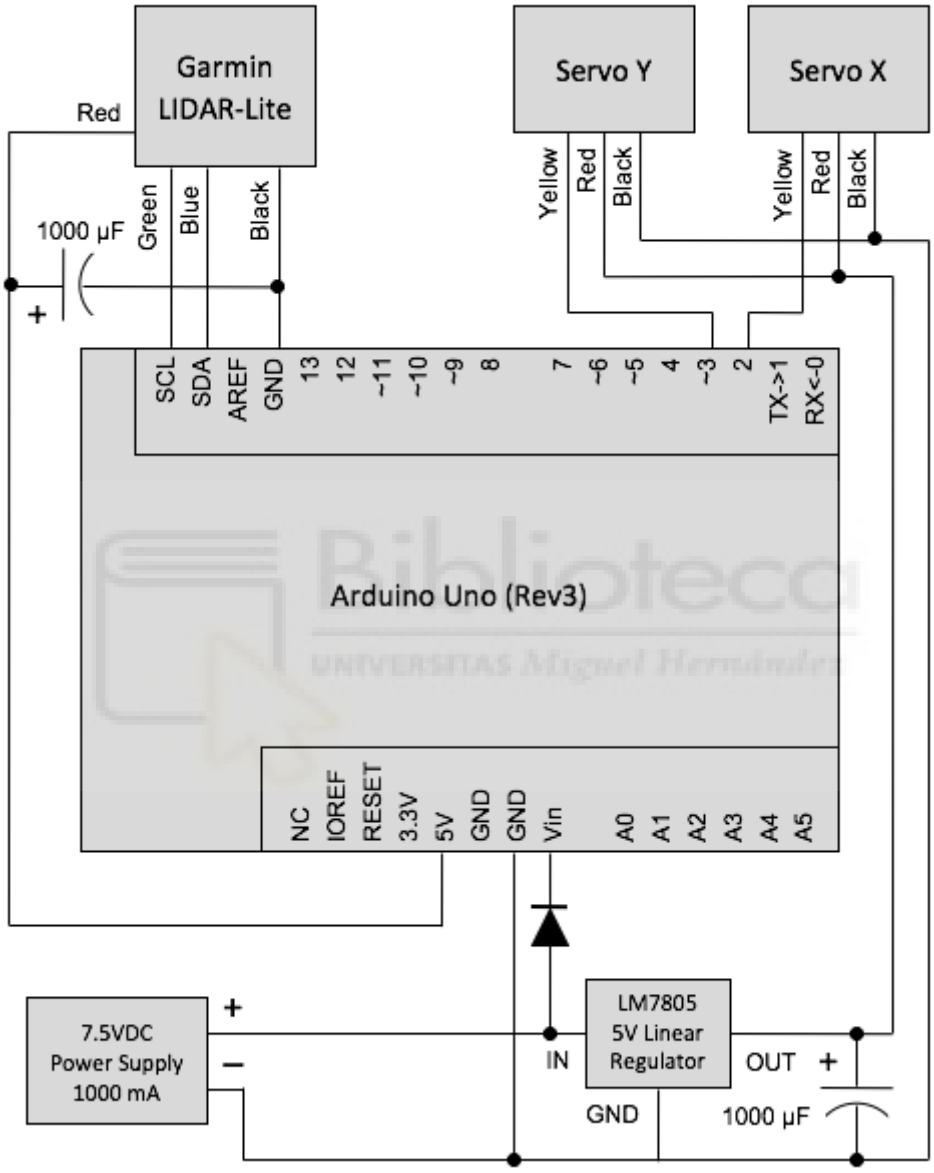


Ilustración 23: Esquema eléctrico del proyecto original

De esta forma, los únicos componentes que quedan externos a cualquiera de las placas mencionadas en el apartado 4.1 son la fuente de alimentación, el lídar y los dos servomotores. Las 3 placas, al poseer la misma distribución de pines que la placa Arduino, se apilan unas encima de otras en el siguiente orden de abajo a arriba: Arduino (apartado 4.1.3), potencia (apartado 4.1.7) y LCD (apartado 4.1.6).

4.3. Sistema de Control

A continuación en la Ilustración 24, se presenta el esquema de control utilizado para el desarrollo del proyecto. Como puede observarse, casi todos los componentes poseen comunicaciones bilaterales puesto que son tanto receptores como emisores de datos, mientras que los servomotores sólo son dispositivos de salida.

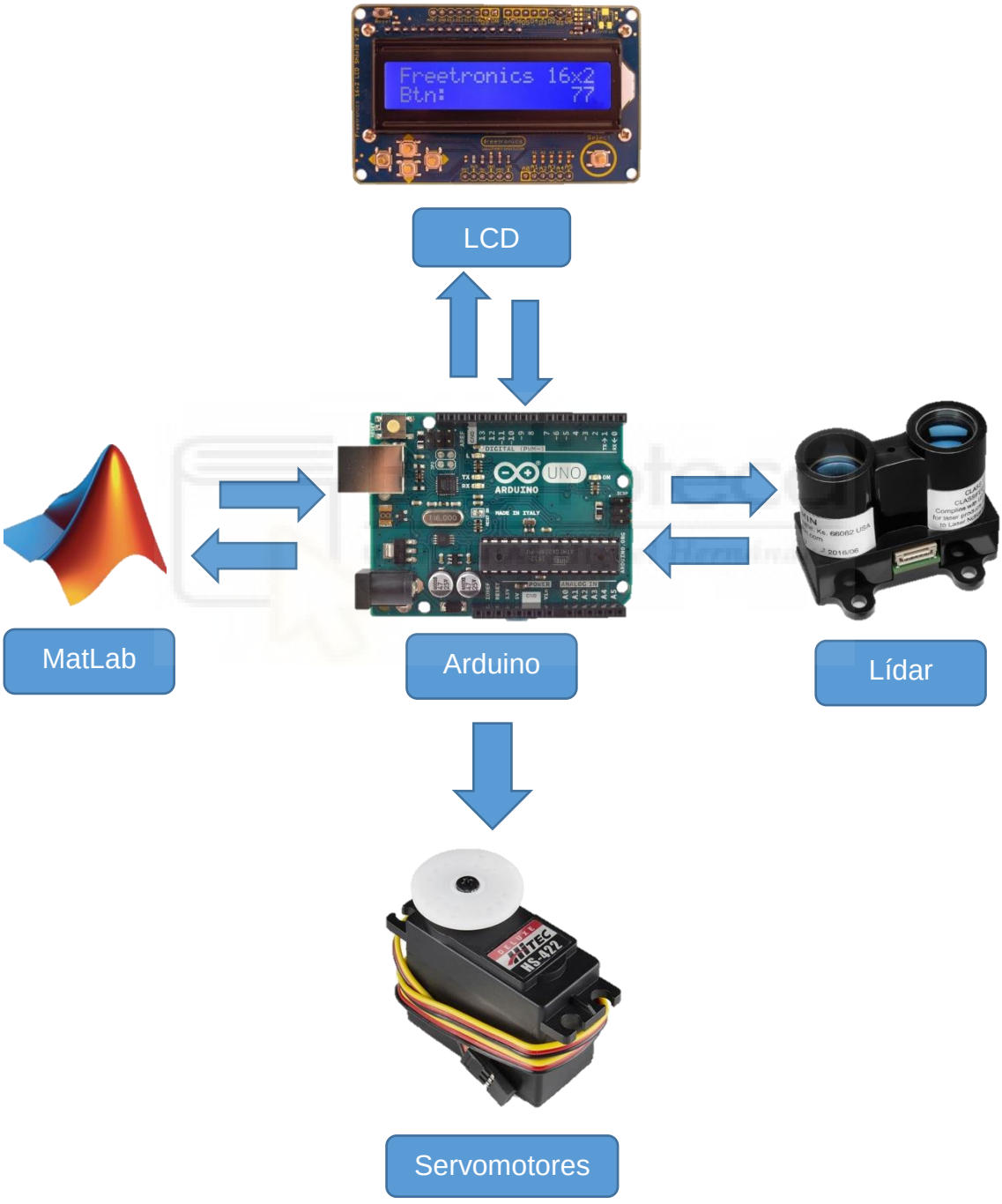


Ilustración 24: Esquema de control

Para empezar, el interfaz de control interactivo humano – máquina es implementado en MatLab a través de un GUI (*Grafical User Interface*) con el objetivo de conseguir un entorno más intuitivo y así facilitarle al operario humano el entendimiento y manejo del mecanismo, haciendo posible el envío y recepción de datos de forma bilateral con Arduino. Asimismo, Arduino compone el sistema central de control del resto de componentes, siendo totalmente independiente de MatLab para el manejo de éstos (excepto modificación de parámetros) y conservando el control total de cada componente.

La comunicación establecida con el lídar también es bilateral. Esto se debe a que se utiliza un interfaz de comunicación I2C que permite controlar a través de comandos cuándo debe realizar una medida y al mismo tiempo recibir los datos de esa medida. Por otro lado, la placa LCD constituye principalmente un dispositivo de salida, pero la botonera incorporada le dota de la capacidad de controlar el manejo de los motores y el lídar en modo manual, de forma que es capaz de adquirir datos concretos del entorno manualmente y reproducirlos en la pantalla LCD. También utiliza el botón SELECT como seta de emergencia para interrumpir un escaneo automático y volver al modo manual.

Por último, los servomotores son únicamente dispositivos de salida. Al no poseer ningún interfaz para verificar su posición real, no es posible averiguar el error de medición debido a los servomotores, por lo que tan solo utilizan una conexión PWM unidireccional.

5. Implementación

En esta sección, se explica cómo ha sido realizado el montaje de la estructura y qué pasos se han seguido. Posteriormente, se realiza una descripción del código utilizado para implementar el control.

5.1. Montaje

Para realizar el montaje, se ha construido la estructura *pan and tilt* siguiendo las indicaciones mostradas en la web [17]. Posteriormente, el lídar ha sido anclado mediante dos pernos (en esquinas opuestas) a una tabla de madera con la misma distribución de agujeros, tal y como se muestra en la Ilustración 25, que se utiliza para que el conjunto quede agarrado a presión al componente de aluminio superior de la estructura, estando éste último entre la madera y el lídar. Como la base de la estructura queda un poco pequeña (ya que es directamente la base del servomotor de azimut), se le ha añadido una base artesanal fijada a presión mediante dos tornillos al servomotor.

Ilustración 25: Estructura lídar

Para realizar las primeras pruebas de escaneo, al no haberse fabricado todavía la placa de potencia, se ha utilizado una *protoboard* para unir cada uno de los componentes de dicha placa de forma temporal mediante cables y así medir los primeros resultados.

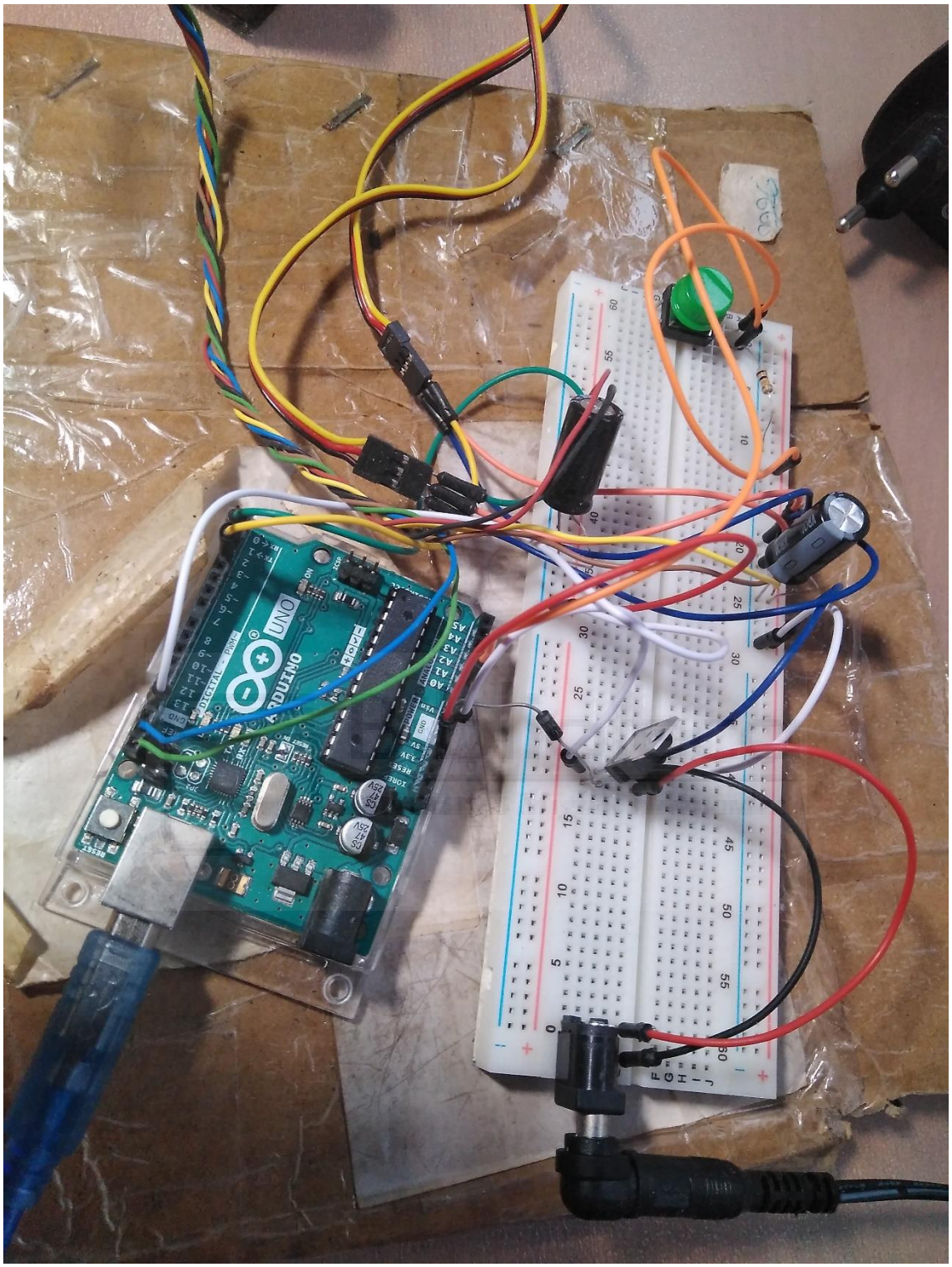


Ilustración 26: Montaje intermedio del circuito eléctrico mediante placa de prototipado

Más tarde, tras la fabricación de la placa de potencia final, queda la cuestión del montaje y conexión de las 3 placas. Éstas, al igual que como se describe en el apartado 4.2: se apilan unas encima de otras en el siguiente orden de abajo a arriba: Arduino (apartado 4.1.3), potencia (apartado 4.1.7) y LCD (apartado 4.1.6).

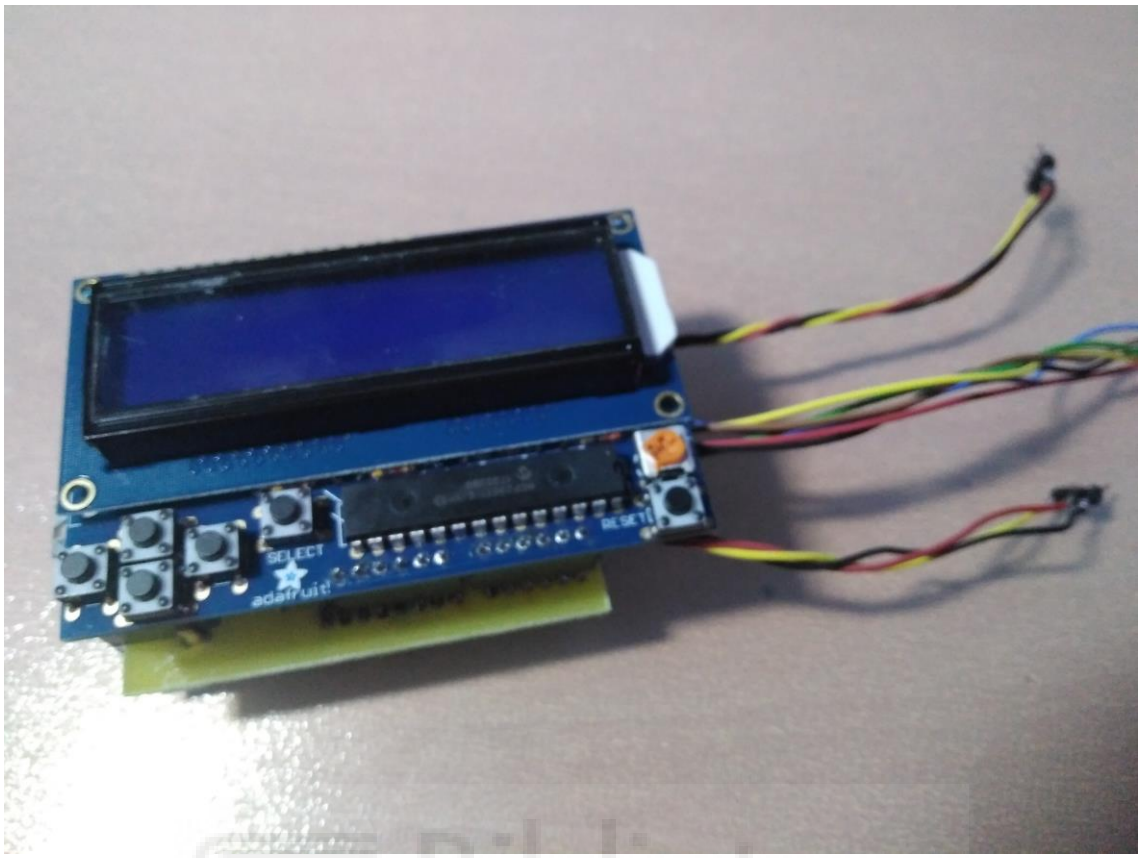


Ilustración 27: Placas apiladas

Finalmente, los servomotores y el lídar se conectan mediante cables a la placa de potencia. No obstante, en cuanto al lídar, se ha necesitado soldar un componente de pines para unir los cables y que no queden sueltos tal y como se muestra en la Ilustración 28, pudiendo facilitar la conexión con la placa. También, al ser éstos demasiado cortos, se ha fabricado un alargador con una distribución similar para permitir al lídar un movimiento más libre evitando tirones innecesarios.

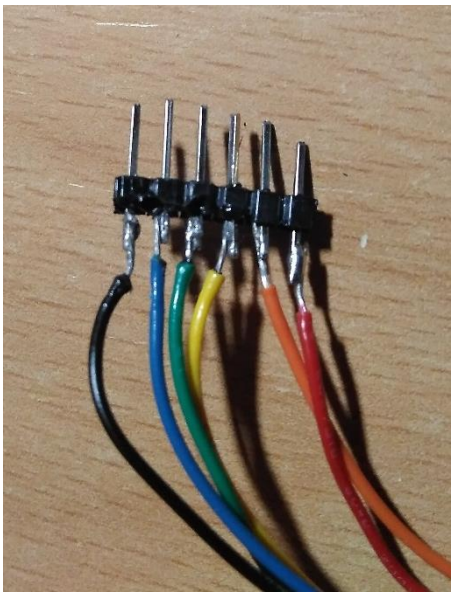


Ilustración 28: Conector soldado a los pines del lídar

5.2. Algoritmo de Control

A continuación, se procede a describir el código utilizado para implementar el control del conjunto, dividiendo en subapartados según el componente, archivos de código que utiliza cada dispositivo y algoritmos implementados en cada archivo.

5.2.1. Arduino

El algoritmo implementado en la placa Arduino utilizada (apartado 4.1.3) se divide, a su vez, en dos tramos de código principales: `setup()` y `loop()`. El primero de ellos es utilizado por defecto por Arduino para ejecutarse una sola vez al principio con el objetivo de realizar las primeras configuraciones del sistema, mientras que el segundo, es el que realiza la función principal y se ejecuta después de forma cíclica un número infinito de veces. A su vez, estos dos algoritmos principales utilizan funciones auxiliares que permiten ordenar el código y facilitar su funcionamiento y comprensión.

5.2.1.1. Setup

Éste es el primer algoritmo en ejecutarse. Configura los servomotores, el lídar, la pantalla LCD y el pin con el que lee la señal analógica de la botonera. Posteriormente, inicia la transmisión de datos por puerto serie y espera a que éste se conecte con algún dispositivo. Finalmente, utiliza una llamada a la función `UpdateModeDisplay` con la que comunica su estado (escaneo/manual) a través de la pantalla LCD y el puerto serie.

5.2.1.2. Loop

Se trata del algoritmo principal. En él, primero se lee la señal de la botonera y se inicia una estructura de selección según su estado (escaneo/manual). Si está en modo de escaneo, incrementa el ángulo al que debe posicionarse el motor teniendo en cuenta la dirección de desplazamiento, ya que cada vez que recorre una línea de azimut entera cambia el sentido de giro. Además, ajusta los datos de posición siguiente para que no sobrepasen los límites establecidos por los parámetros anteriormente determinados mediante MatLab. Cuando llega al final del ángulo de elevación o si detecta que se ha pulsado el botón SELECT cambia a modo manual. También, cuando se establece en MatLab el modo de prueba de precisión, se determina asimismo un número máximo de valores a tomar, y una vez se alcanza este número cambia a modo manual. Por otro lado, cuando se encuentra en modo manual pasa a dirigirse directamente con la botonera, variando el giro en diferencias de 1º. En este caso, el botón SELECT se utiliza para cambiar a modo de escaneo a través de la función `Iniciar_escaneo`.

El siguiente bloque de código lee los datos transmitidos a través del puerto serie, en el cual se determinan los siguientes valores y en el siguiente orden: número máximo de escaneos, azimut mínimo y máximo, elevación mínima y máxima, incrementos angulares en azimut y elevación y finalmente tiempo de espera. Cuando se quiere realizar la prueba de precisión del lídar, se recibe un valor para la variable del número máximo de escaneos mayor a 0, en cuyo caso se omiten los valores de azimuts, elevaciones e incrementos, sustituyéndolos por 90, 90, 90, 90, 0 y 0 respectivamente, de forma que el lídar tomará muestras en una posición estática frontal. Finaliza realizando una llamada a `Iniciar_escaneo`.

Posteriormente, ajusta los valores de azimut y elevación, que después utiliza para mover los servomotores utilizando la función `MoveServos` y muestra su posición mediante `DisplayPosition`. A continuación, utiliza un contador para recalibrar el lídar cada 100 escaneos y toma una muestra del entorno a través del lídar. Posteriormente, actualiza en la pantalla LCD la distancia muestreada cada vez que ésta sea al menos 2 cm distinta de la que se tomó la última vez que se actualizó este mismo dato en el LCD, es decir, que este dato en la pantalla se actualiza cada vez que varía más de 2 cm. Por último, cada vez que el sistema esté en modo de escaneo o los servomotores se hayan movido, las coordenadas esféricas se convierten a coordenadas cartesianas y se envían a través del puerto serie, además de actualizarse estos datos en la pantalla LCD.

5.2.1.3. Iniciar_escaneo

Esta función inicia el modo de escaneo automático cambiando la variable de estado correspondiente, reduciendo al mínimo los valores de azimut y elevación, estableciendo el sentido de giro y reiniciando el contador del número de escaneos. Posteriormente, realiza un retardo de tiempo cuyo parámetro ha sido establecido a través de MatLab (se utiliza para darle tiempo al operario a salir de una habitación que vaya a escanearse) y utiliza `UpdateModeDisplay`.

5.2.1.4. MoveServos

Se trata de una función que envía los datos de posición a cada servomotor y devuelve un valor booleano según se hayan enviado o no estos datos a alguno de ellos; ya que, como puede recordarse en el apartado 4.3, no se puede saber con certeza si ha habido movimiento en alguno de ellos. Conserva los últimos valores utilizados de azimut y elevación por esta misma función y los compara con los actuales para saber si son distintos, en cuyo caso, calcula cada desplazamiento angular y envía el dato al servomotor correspondiente. Por último, se genera un retardo en el código relativo a la cantidad de desplazamiento del servomotor que más tiene que moverse, para darles tiempo a completar el desplazamiento. Para esto último se supone que tardan 5 ms en recorrer 1º.

5.2.1.5. DisplayPosition

Función que se utiliza para actualizar en la pantalla LCD los datos de azimut (X) y elevación (Y) cuando son distintos a los últimos utilizados.

5.2.1.6. UpdateModeDisplay

Muestra en la pantalla LCD el estado del sistema (escaneo/manual) y lo envía a través del puerto serie.

5.2.2. MatLab

MatLab es el interfaz de control utilizado para este proyecto debido a su potencia de cálculo y su gran abanico de posibilidades a la hora de programar. Además, se utiliza un GUI o interfaz gráfico que facilita enormemente la labor del usuario del sistema lídar.

Se compone principalmente de 3 archivos repletos de código que, en conjunto, se complementan para formar el algoritmo de control completo: el algoritmo principal llamado "`intit_lidar.m`", los archivos del GUI "`LidarGUI.m`" (funciones) y "`LidarGUI.fig`" (figura gráfica), y el archivo de funciones globales "`funciones.m`".

5.2.2.1. Algoritmo Principal

Este archivo contiene el código principal a ejecutar a través de la línea de comandos mediante el comando "`init_lidar`". Su función consiste en iniciar el programa y ejecutar el bucle principal del proceso.

Para ello, primero borra la línea de comandos y se asegura de que ni hay ningún bucle paralelo iniciado ni haya permanecido abierto ningún puerto de comunicación con Arduino, borrándolos así en tal caso. Después, inicia el GUI

(interfaz gráfico) entrando en el bucle principal y esperando a que el usuario interactúe con ello. A partir de aquí, dependiendo de la decisión del usuario el funcionamiento del algoritmo varía. Si el usuario decide cerrar el entorno a través del botón “Cerrar Entorno”, éste se cierra y el programa finaliza; si se cierra mediante la X de la ventana del programa, el *script* no finalizará correctamente. En cambio, si decide escanear o realizar una prueba de precisión, se procede a la conexión con Arduino a través de la función Conexión.

Una vez conectado, entra en otro bucle en el cual se escanean constantemente los datos recibidos a partir del puerto serie. En cada bucle, se ejecuta una estructura de decisión *switch* que diferencia entre las distintas posibilidades de datos recibidos. Si recibe el comando “*Scanning*”, inicia el modo de escaneo vaciando la nube de puntos e inicializando un temporizador que más adelante temporizará el tiempo de escaneo de la estructura lídar. Por otro lado, si recibe el comando “*Manual*”, desconecta y muestra el tiempo del temporizador, forzando también a desconectarse de Arduino y así poder habilitar de nuevo la interacción con el GUI. Esto último se debe a la problemática que supone el hecho de que MatLab no responde mientras está ejecutando código, cuestión que ha debido solucionarse mediante la desconexión forzada de éste. En Arduino, conectar y desconectar varias veces desde un dispositivo externo no supone ningún problema, no es necesario utilizar el botón de reinicio. Por último, en caso de no cumplirse ninguno de los casos anteriores el sistema analiza los datos de entrada suponiendo que son datos cartesianos (X, Y y Z), filtrándolos de modo que si no se encuentra la estructura esperada los descarta.

Finalmente, una vez se haya indicado la necesidad de desconectar, el bucle finaliza y se inicia la función de Desconexión. Tras ella, en caso de haberse realizado la prueba de precisión se muestran los resultados mediante la función Precisión, sino, se inicia el filtro de puntos a través de la función Filtrar en caso de haber sido activado este parámetro y se muestran los resultados mediante la función Dibujar.

5.2.2.2. GUI

Se trata del interfaz gráfico de usuario a través del cual se controlan los parámetros del sistema y se envían los comandos correspondientes. Se divide, a su vez, en dos archivos separados: “`LidarGUI.fig`”, que contiene la información gráfica del entorno, y “`LidarGUI.m`”, que contiene la información correspondiente al código utilizado.

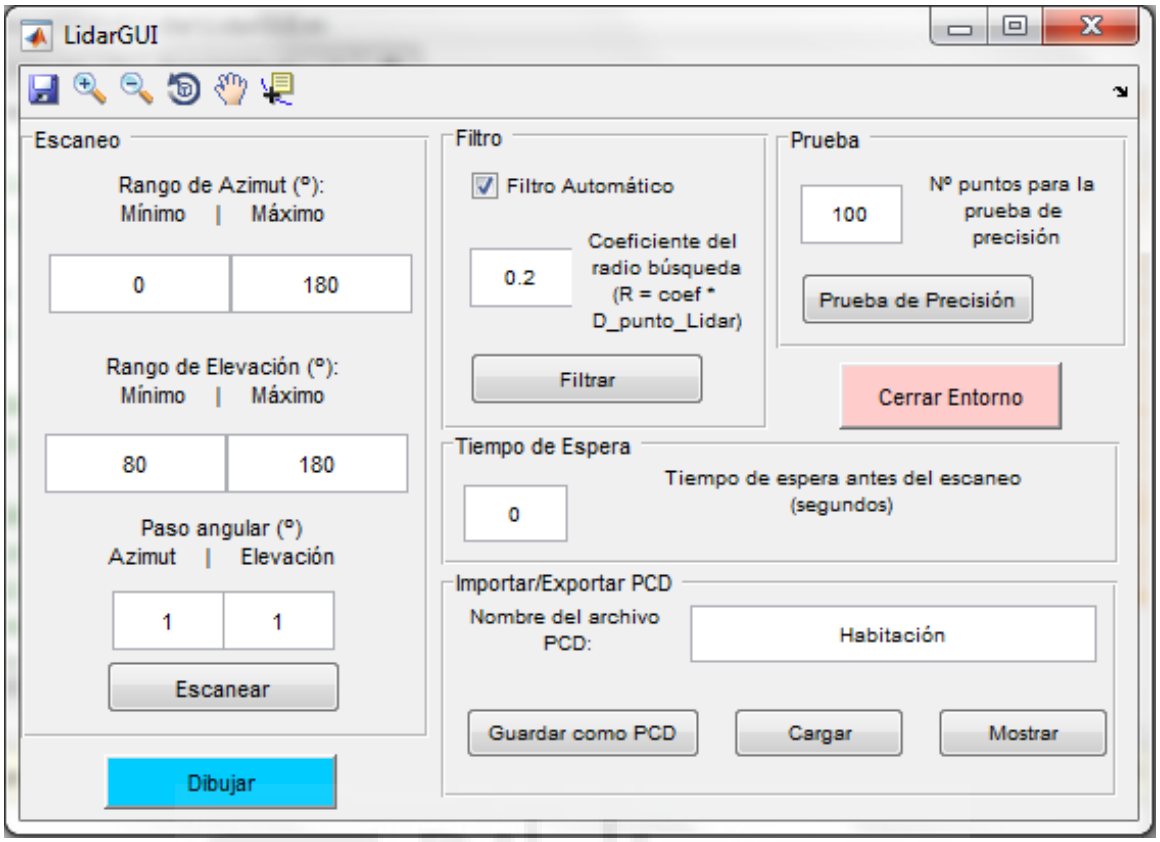


Ilustración 29: Interfaz Gráfico de MatLab (GUI)

Como puede verse en la Ilustración 29, el GUI dispone de varios apartados separados según su función. Así, se puede distinguir entre 5 contextos diferentes: Escaneo, Filtro, Prueba, Tiempo de Espera e Importar/Exportar PCD.

En el apartado Escaneo, se configuran los parámetros que indican los límites sobre los cuales ha de ajustarse Arduino para realizar el escaneo automático, además del paso angular establecido tanto en azimut como en elevación. El botón “Escanear” inicia el modo de escaneo automático del lídar.

Por otro lado, en el apartado Filtro, se puede configurar tanto el coeficiente de búsqueda como la activación del filtro de forma automática. El botón “Filtrar” activa el filtro de forma manual a través de la función Filtrar.

Mientras tanto, la sección Prueba, está compuesta por el parámetro del número máximo de puntos o datos a adquirir para la prueba de precisión y el botón que inicia dicha prueba.

También, se puede encontrar la sección Tiempo de Espera, la cual especifica el tiempo de retardo o espera que ejecutará Arduino desde su conexión con MatLab hasta que inicie el escaneo automático.

El apartado Importar/Exportar PCD, tal y como su nombre indica, contiene las herramientas necesarias para guardar la nube de puntos al formato .pcd,

extraer datos desde ese formato y visualizarlos. Los botones “Guardar como PCD”, “Cargar” y “Mostrar” realizan esas mismas acciones, respectivamente. El campo de texto “Nombre del archivo PCD” establece el nombre con el que se cargará o guardará el archivo .pcd. Para visualizar datos .pcd, antes hay que guardar o cargar una nube de puntos previa.

Finalmente, el botón “Cerrar Entorno” finaliza de forma segura el *script* y “Dibujar” permite visualizar la nube de puntos adquirida de forma manual.

5.2.2.3. Funciones Globales

Por último, el siguiente archivo contiene todas las funciones que se utilizan de forma global tanto por el algoritmo principal (apartado 5.2.2.1) como por el GUI (apartado 5.2.2.2). Para definirlas se ha utilizado una clase, como en C++, cuya definición se denomina “funciones” (`classdef funciones`), que a su vez contiene métodos (funciones ligadas a una clase) que son estáticos.

5.2.2.3.1. Conexión

Esta función permite a MatLab conectarse a Arduino a través del puerto serie. Crea el objeto serie en memoria, lo configura y abre la comunicación. Después, espera a que se establezca el primer contacto y envía los valores de los parámetros configurados a través del GUI.

5.2.2.3.2. Desconexión

Cierra la conexión y borra el objeto serie.

5.2.2.3.3. Filtrar

Esta función trata de filtrar la nube de puntos adquirida mediante la búsqueda de puntos vecinos a una distancia euclídea inferior a un radio especificado de forma relativa a la distancia que hay entre el punto y el origen de coordenadas del sistema lídar. El coeficiente utilizado especifica la longitud del radio relativa a la distancia a la que se halla el punto del lídar, es decir, el radio es igual al coeficiente por la distancia que hay entre el punto y el centro de coordenadas del lídar. Por tanto, cuanto más lejos esté el punto, mayor será el radio con el que buscará puntos vecinos. Se presupone, que estos “vecinos cercanos” son puntos del entorno virtual que han sido muestreados correctamente bajo un cierto margen de error aceptable. Esto se ha planteado así dado que cuanto más lejos están los puntos del lídar mayor separación se genera entre ellos, debido a que el sistema lídar rastrea el entorno en movimiento angular. De esta forma, el filtro sólo permitirá el paso de aquellos puntos que tengan almenos un vecino cercano a una distancia inferior a la del radio, de lo contrario, ese punto “solitario” será eliminado de la nube de puntos. El objetivo de este filtro es evitar conservar muestras falsas de puntos que se hayan

quedado demasiado lejos del resto debido a una medida errónea del entorno, es decir, puntos aislados. También dispone de un temporizador que cronometra el tiempo que tarda en filtrarse los datos.

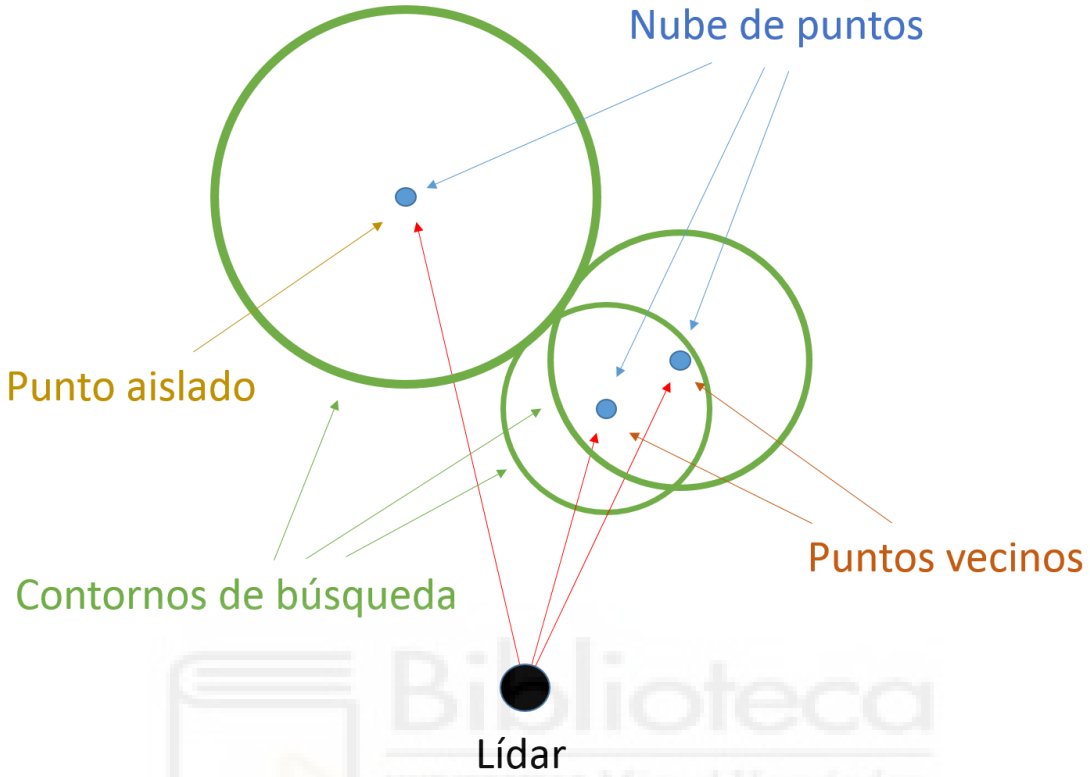


Ilustración 30: Esquema del filtro de puntos

5.2.2.3.4. Dibujar

Esta función dibuja en una ventana emergente los valores muestreados por el lídar. Para ello, establece un margen según los valores máximos de cada dimensión (X, Y y Z) y así visualizar los datos en un espacio cúbico, de forma que todos los ejes sean iguales y se muestren los datos en la misma escala tanto en X como en Y y en Z. Posteriormente, establece la escala de los ejes de coordenadas absolutos como 0,2 veces la escala de los ejes anteriores, para así mostrar el origen de coordenadas y la pose de los ejes absolutos. Finalmente, se dibujan los datos y se configura la ventana con perspectiva cónica.

5.2.2.3.5. Precisión

Función que muestra los resultados de la media y la desviación típica de la nube de puntos adquirida a través de la prueba de precisión.

5.2.2.3.6. Distancia

Función que determina de forma booleana si dos puntos son distantes a través del coeficiente del filtro. Es utilizada por la función Filtrar y utiliza el fundamento de cálculo expresado en el apartado 5.2.2.3.3. Cuando los dos puntos son equivalentes, ésta determina que también son distantes, para de esta forma eliminar puntos repetidos cuando éstos, a su vez, son distantes al resto. De lo contrario, en caso de haber dos puntos aislados equivalentes (superpuestos), no se eliminarían. Esta última característica no es necesaria implementarla para el sistema lídar, ya que no está programado para escanear dos veces en una misma dirección, pero se pretende que pueda ser utilizada para otras nubes de puntos que puedan poseer valores repetidos, eliminando así tanto falsos positivos como sus respectivos duplicados.

5.2.2.3.7. Notificar

Se trata de una función utilizada por la función Conexión para desconectar el dispositivo y mostrar el tipo de error que se ha producido.

5.2.2.4. Librería Point Cloud

Se trata de una librería de código nativo de MatLab que ofrece distintas opciones que permiten tanto visualizar como exportar e importar archivos en formato PCD. Los archivos PCD son conjuntos de datos en formato de nube de puntos, los cuales llevan asociados diversos parámetros como la localización de cada punto, su color, intensidad o sus respectivos vectores normales. Para la aplicación de esta librería se han utilizado hasta cuatro funciones, las cuales son: `pcread`, `pcwrite`, `pcshow` y `pointCloud`.

La primera de todas, `pcread`, se encarga de leer un archivo de datos PCD previamente guardado y asignar sus valores a una variable en formato `pointCloud`. Del mismo modo, `pcwrite` se encarga del proceso inverso; es decir, guardar los valores de una variable `pointCloud` en un archivo PCD. Para la visualización de estos valores se utiliza la función `pcshow`, que parte de una variable en formato `pointCloud` como parámetro. Y finalmente, la función `pointCloud` se encarga de convertir un conjunto de puntos de un sistema de coordenadas 3D (*array* numérico de NX3) en un objeto de tipo `pointCloud`.

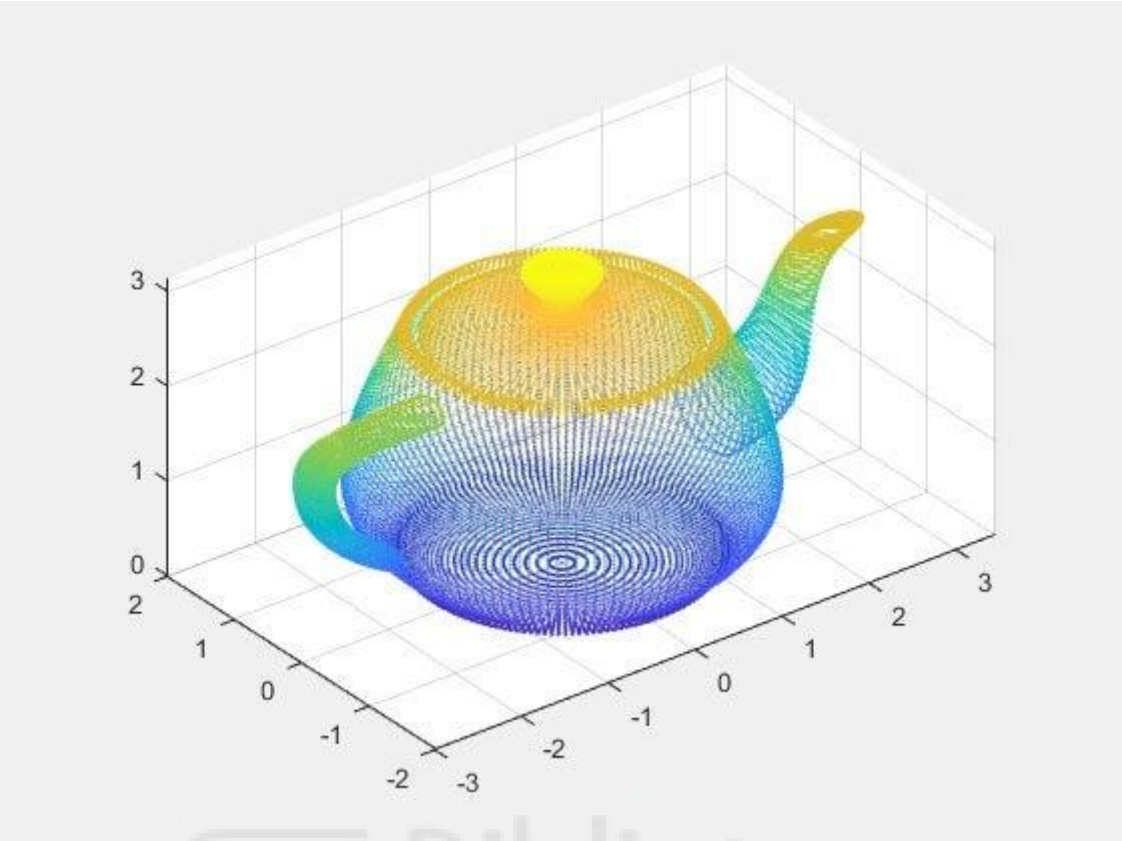


Ilustración 31: Visualización de una nube de puntos a través de la librería Point Cloud de MatLab

6. Resultados

A continuación, se muestran los resultados de algunas de las pruebas realizadas con la estructura de escaneo 3D. Para ello, se han utilizado los dos dispositivos lídar disponibles con tal de realizar una comparación experimental entre ambos y el experimento se ha dividido en dos partes: Un escaneado de una habitación con un posterior filtrado de puntos con coeficiente 0,2 y dos pruebas de precisión para cada lídar, siendo éstas a 40 cm y 2 m de distancia de una pared.

6.1. Telémetro Láser LÍDAR-Lite 3

6.1.1. Escaneo

Para empezar, se ha utilizado el dispositivo lídar de bajo rendimiento. Primero se realiza un escaneo a una pared plana para determinar posibles curvaturas de la nube de puntos o desviaciones angulares del conjunto, utilizando parámetros de azimut de 85º a 95º, elevación de 100º a 110º e incrementos angulares de 1º, dando lugar al siguiente resultado.

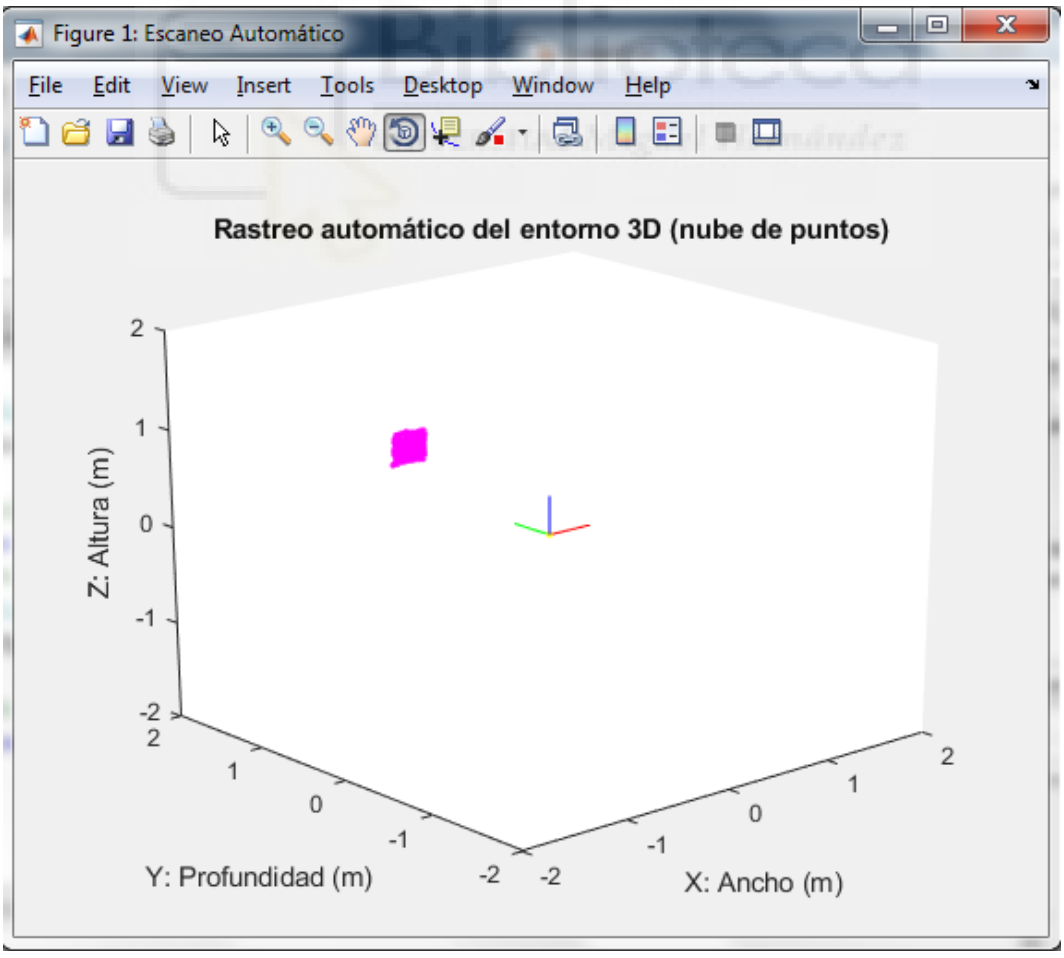


Ilustración 32: Escaneo de una pared con lídar de bajo rendimiento

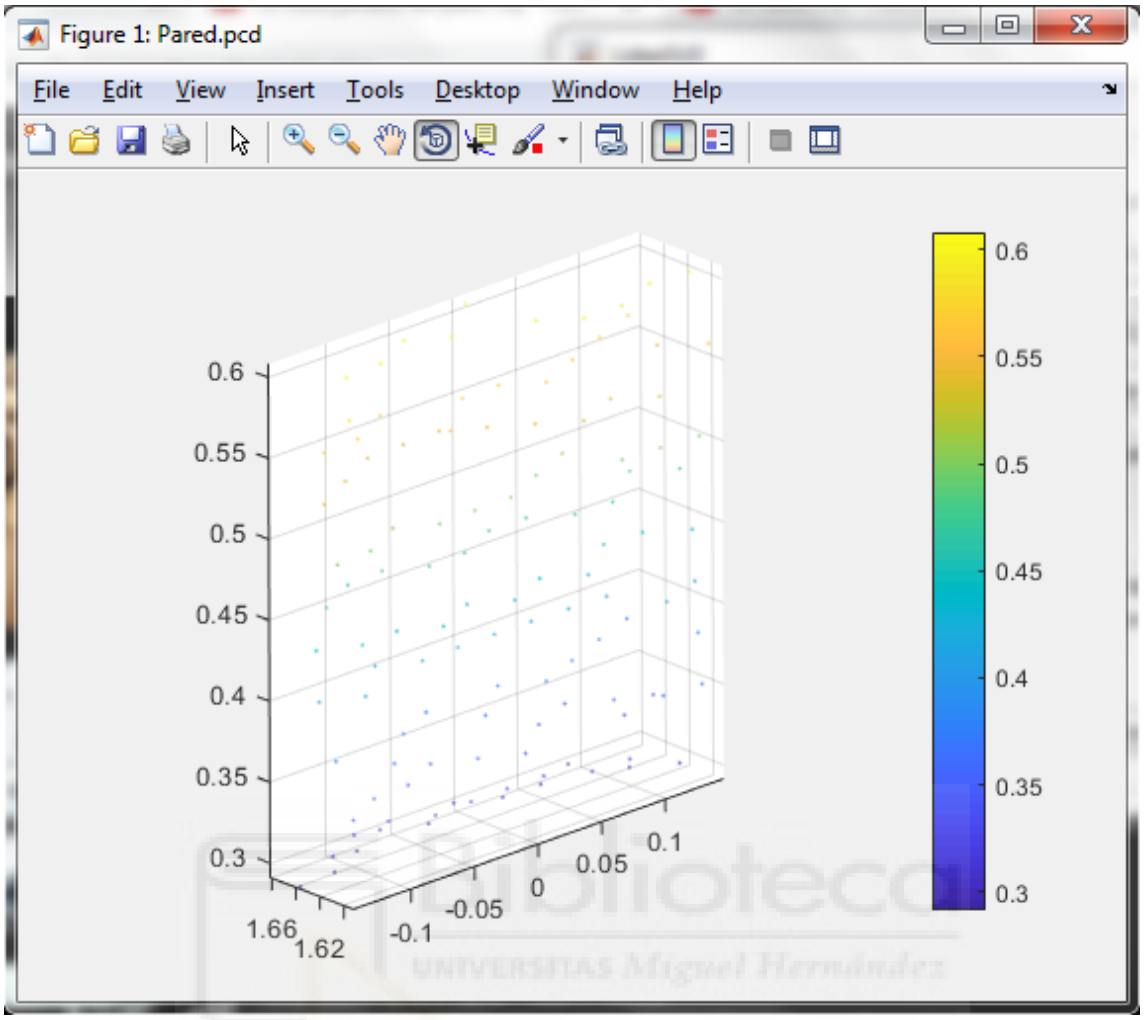


Ilustración 33: Archivo PCD del escaneo de una pared con lídar de bajo rendimiento

La nube de puntos se muestra muy densa, sobre todo para el escaneo de una pared lisa, lo cual indica que el lídar es muy poco preciso.

A continuación, se ha procedido al escaneo de una habitación. Se han utilizado los parámetros por defecto de 0º a 180º de azimut y de 80º a 180º de elevación, con incrementos angulares en ambos grados de libertad de 1º. Tras la finalización del escaneo se reciben los datos siguientes:

```
Comando recibido: Manual
Tiempo de escaneo: 552.974180 segundos
Iniciando modo manual
```

Dando así lugar a un dibujo del entorno tridimensional con algunos datos erróneos fácilmente perceptibles puesto que se muestran como puntos aislados del resto sin ninguna continuidad.

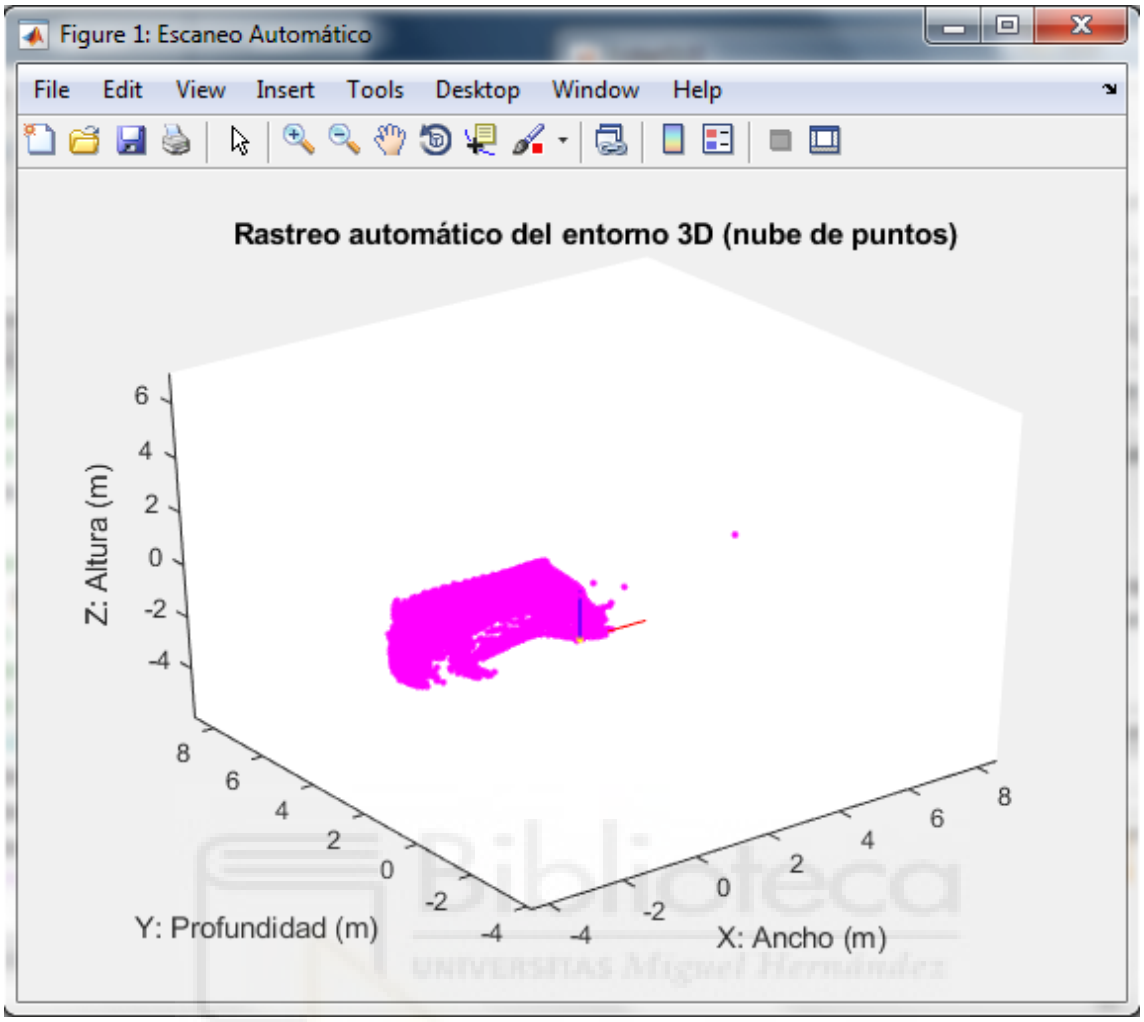


Ilustración 34: Escaneo automático del entorno 3D con el dispositivo lídar de bajo rendimiento

La función "`plot3`" de MatLab no ofrece una visualización de los datos muy intuitiva. Por tanto, se utiliza la librería PCD para transformar estos datos en una nube de puntos PCD y guardarlos convenientemente, pudiendo mostrarse del siguiente modo:

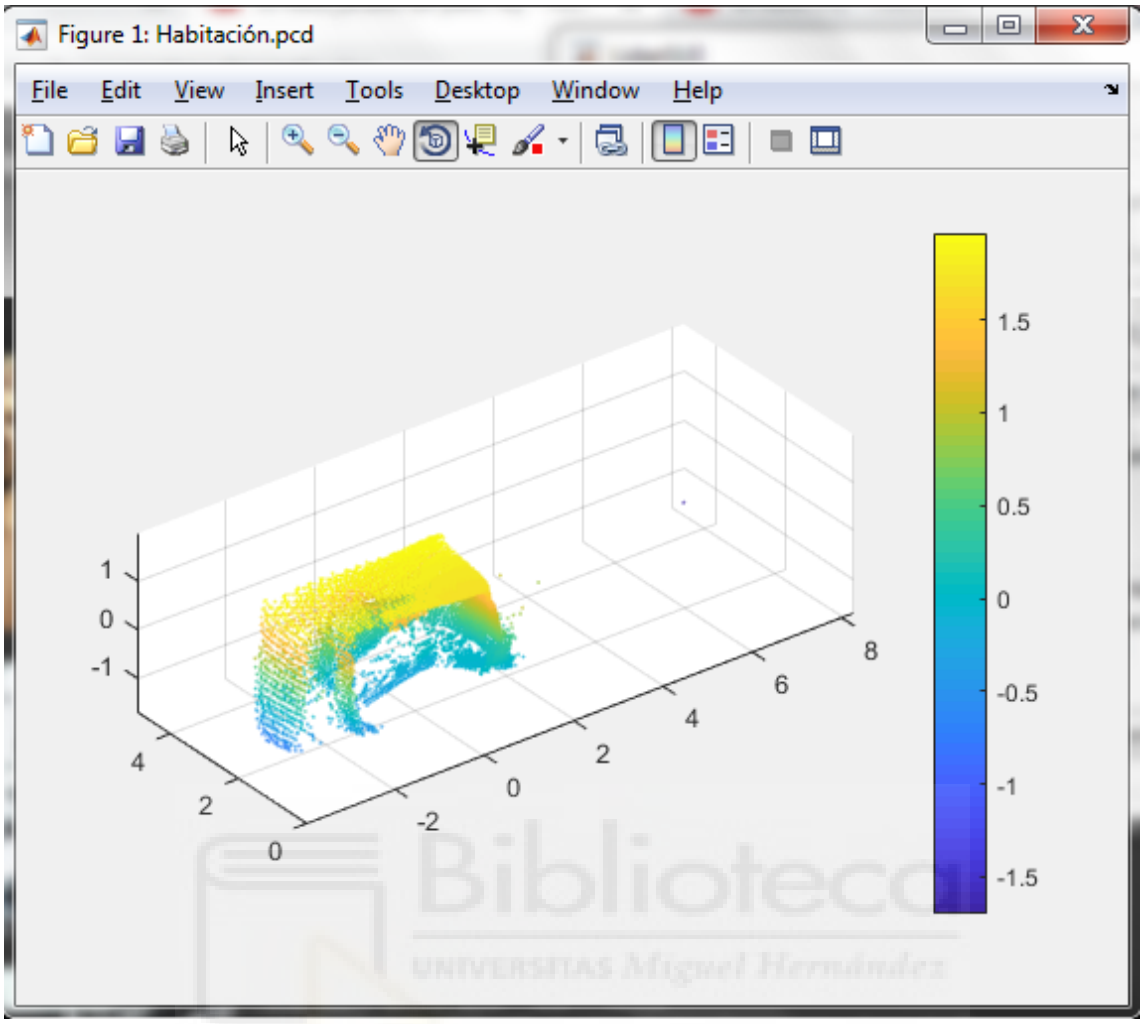


Ilustración 35: Nube de puntos de la habitación con el dispositivo lídar de bajo rendimiento tras la exportación de los datos a formato PCD

Ahora los datos pueden percibirse fácilmente en una escala de colores que indica la altura (eje Z) a la que se localiza cada punto. Posteriormente, con tal de eliminar las falsas mediciones producidas durante el escaneo, se procede al filtrado de puntos, dando lugar al siguiente resultado:

```
Se ha iniciado el filtrado de puntos con 18281 valores
Tiempo empleado por el filtro: 768.035789 segundos
Se han filtrado 11 valores, quedando un total de 18270 puntos
```

Tras este proceso, la nube de puntos se ajusta más a la realidad habiendo eliminado los puntos más aislados y dando lugar al siguiente gráfico, el cual ya no tiene ningún valor lejano al resto:

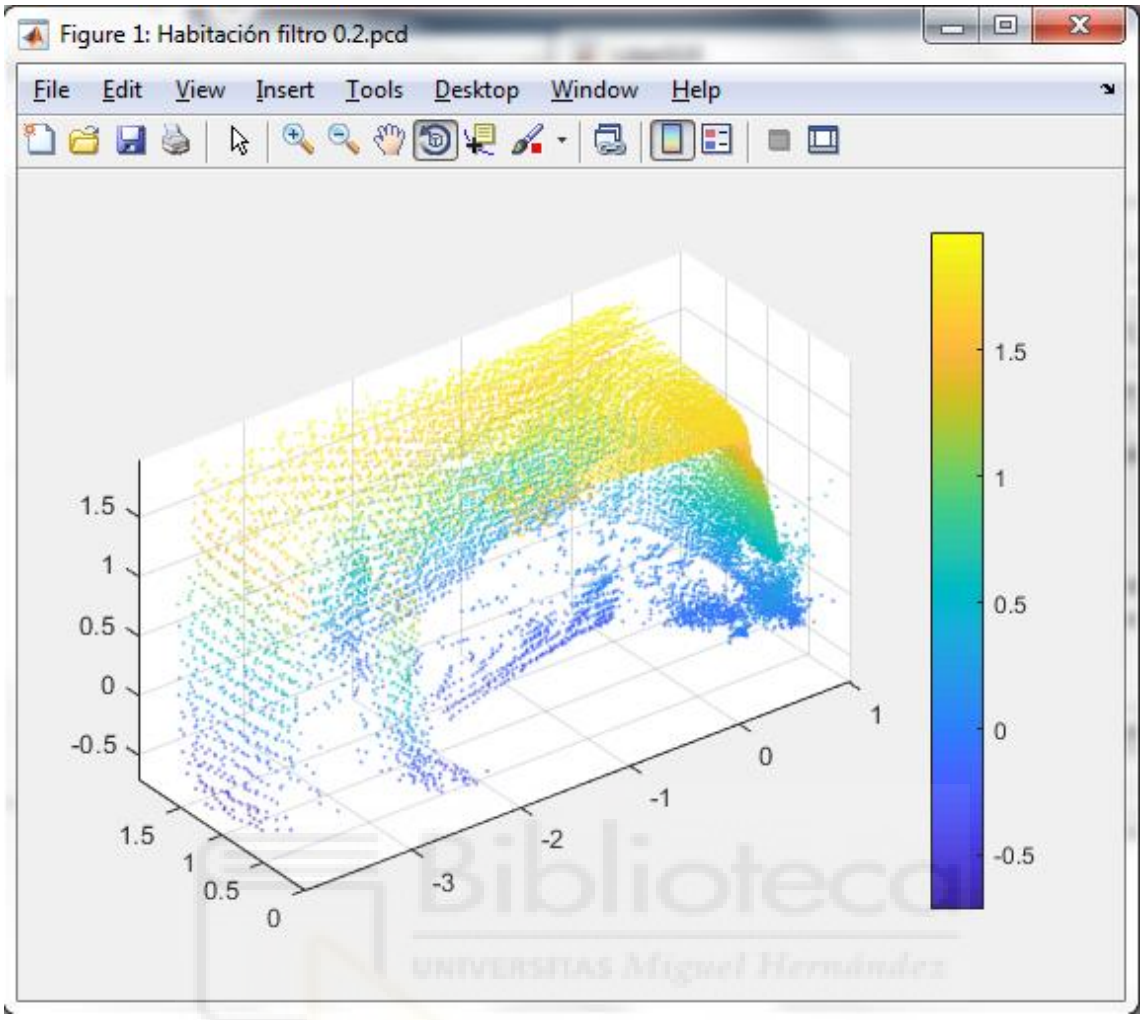


Ilustración 36: Nube de puntos de la habitación con el dispositivo lídar de bajo rendimiento tras el filtrado de puntos con coeficiente 0,2

6.1.2. Precisión

Dada la evolución del error según la longitud a la que se encuentra el lídar del obstáculo, tal y como se explica en el apartado 4.1.1, es necesario realizar distintas pruebas de precisión a diferentes longitudes para averiguar cómo influye la distancia con respecto al resultado de los datos. A continuación, se procede a realizar las correspondientes pruebas de precisión del dispositivo, utilizando para ello un muestreo de 100 puntos. La primera, se realiza a una distancia de 40 cm del origen de coordenadas, dando lugar a los siguientes resultados:

```
Resultados de la prueba de precisión:
MEDIA: 0.460000 metros
DESVIACIÓN TÍPICA: 0.028498 metros
```

Se puede percibir que la media difiere en 6 cm de la distancia real y que la desviación típica es de cerca de 3 cm. Posteriormente, se realiza la misma

prueba a 2 m de distancia del origen de coordenadas, dando como resultado los siguientes datos:

```
Resultados de la prueba de precisión:
MEDIA: 1.665800 metros
DESVIACIÓN TÍPICA: 0.059179 metros
```

Por tanto, ahora la media es más distante del valor real, siendo la diferencia de 33,5 cm, mientras que la desviación típica también ha crecido, siendo de cerca de 6 cm. Y finalmente, se realiza la misma prueba a 5 m, dando el siguiente resultado:

```
Resultados de la prueba de precisión:
MEDIA: 5.122900 metros
DESVIACIÓN TÍPICA: 0.027761 metros
```

Por tanto, se puede deducir que, teniendo una diferencia de 12 cm de la media frente al valor real y una desviación de cerca de 3 cm, el lídar presenta una precisión poco apropiada para la aplicación de escaneo y mapeado.

A continuación, se presenta una gráfica que muestra los resultados en cuanto a la prueba de precisión. Al parecer, en este caso la tendencia de los datos no es del todo lineal, debido a la baja precisión del dispositivo.

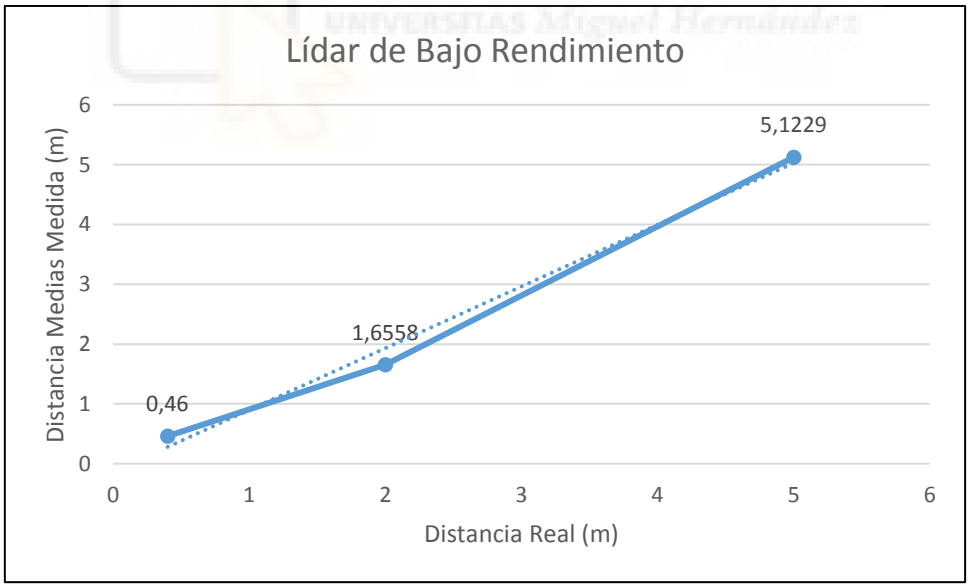


Ilustración 37: Gráfica que muestra la tendencia de la media de las medidas del lídar de bajo rendimiento

6.2. Telémetro Láser LÍDAR-Lite 3 de Alto Rendimiento (LLV3HP)

6.2.1. Escaneo

Para finalizar, se ha utilizado el dispositivo lídar de alto rendimiento. Primero se realiza un escaneo a una pared plana para determinar posibles curvaturas de la nube de puntos o desviaciones angulares del conjunto, utilizando parámetros de azimut de 85º a 95º, elevación de 100º a 110º e incrementos angulares de 1º, dando lugar al siguiente resultado.

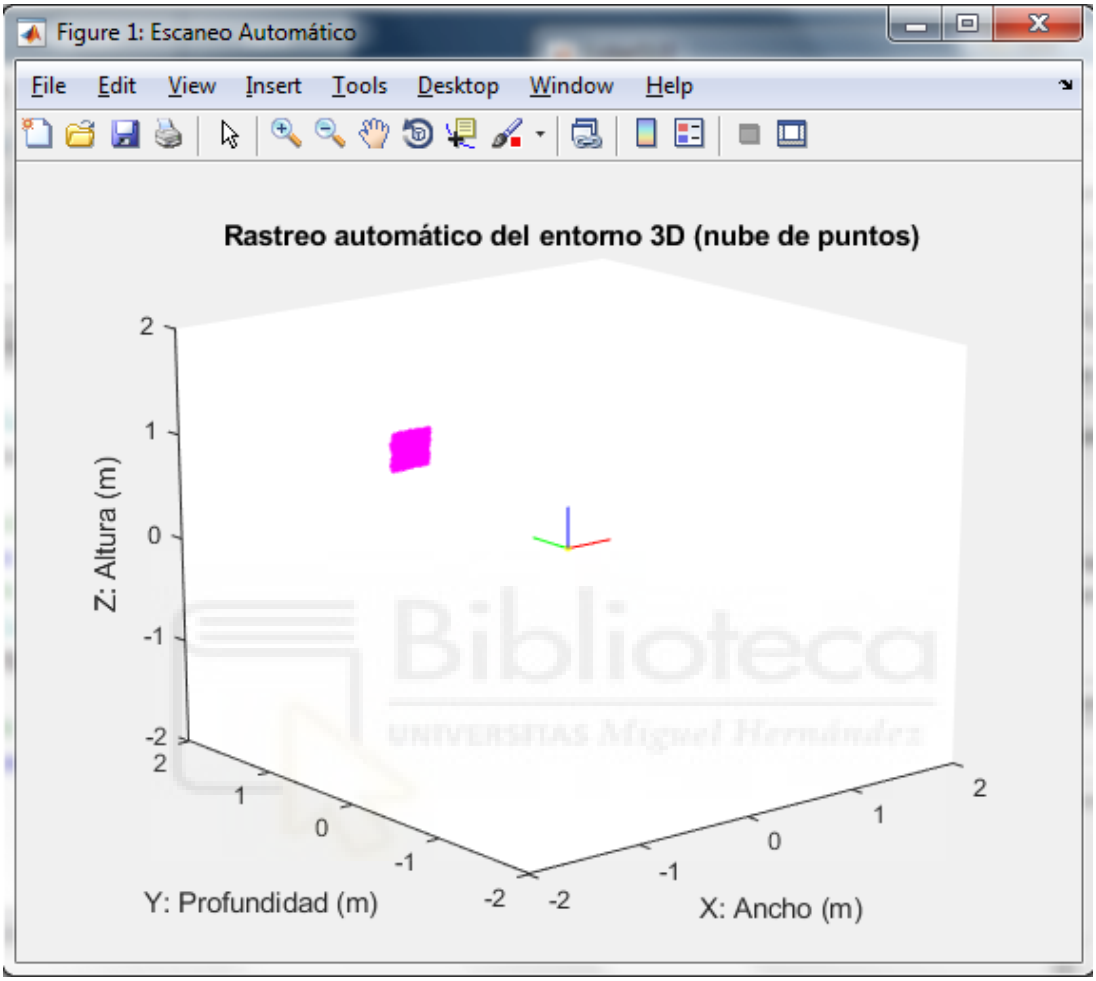


Ilustración 38: Escaneo de una pared con lídar de alto rendimiento

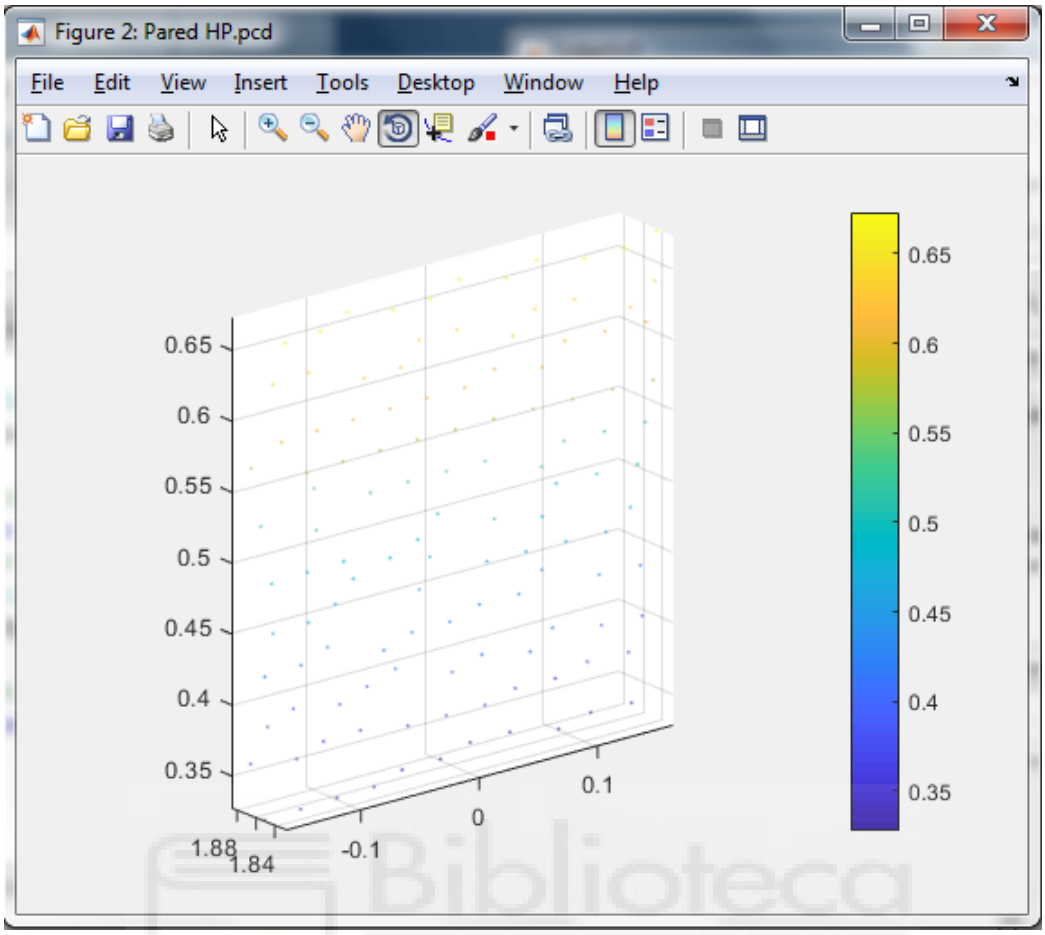


Ilustración 39: Archivo PCD del escaneo de una pared con lídar de alto rendimiento

La nube de puntos muestra una cierta inclinación en el sentido negativo del Eje Y, lo cual indica una descalibración de los servomotores respecto de los ejes de coordenadas del lídar.

A continuación, se pretende realizar un escaneo de una habitación. Para ello, se han utilizado los parámetros por defecto de 0º a 180º de azimut y de 80º a 180º de elevación, con incrementos angulares en ambos grados de libertad de 1º. Tras la finalización del escaneo se reciben los datos siguientes:

```
Comando recibido: Manual
Tiempo de escaneo: 552.974180 segundos
Iniciando modo manual
```

Dando así lugar a un dibujo del entorno tridimensional con un menor número de datos erróneos perceptibles, mostrándose éstos como puntos aislados del resto sin ninguna continuidad.

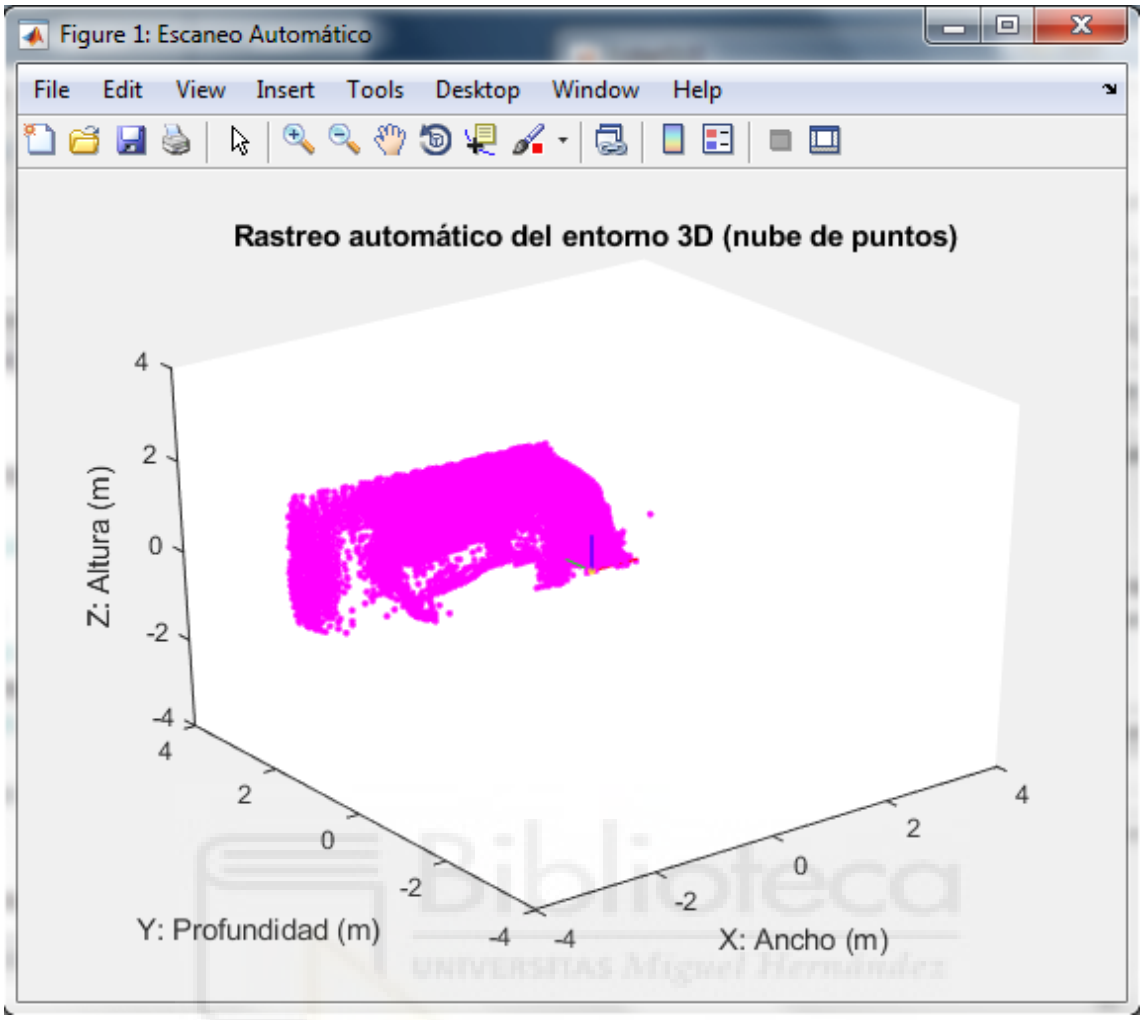


Ilustración 40: Escaneo automático del entorno 3D con el dispositivo lídar de alto rendimiento

Como la función "`plot3`" de MatLab no ofrece una visualización de los datos muy intuitiva, se utiliza la librería PCD para transformar estos datos en una nube de puntos PCD y guardarlos convenientemente, pudiendo mostrarse del siguiente modo:

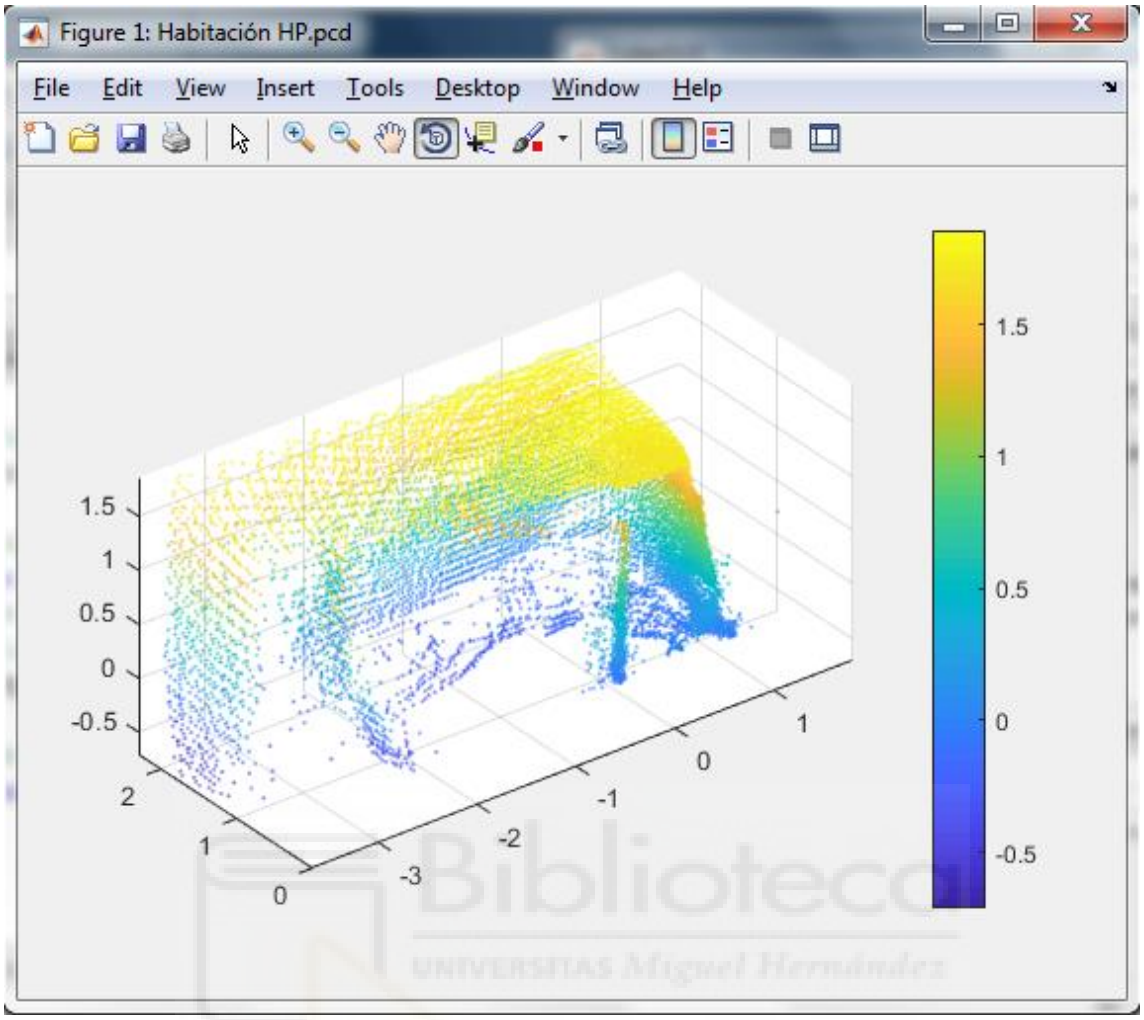


Ilustración 41: Nube de puntos de la habitación con el dispositivo lídar de alto rendimiento tras la exportación de los datos a formato PCD

Ahora los datos se muestran en una escala de colores que indica la altura (eje Z) a la que se localiza cada punto y puede percibirse que éstos datos, muestreados por el lídar de alto rendimiento, tienen menos errores que con el otro dispositivo, siendo más fieles a la realidad y teniendo menos puntos aislados. Posteriormente, con tal de eliminar las falsas mediciones producidas durante el escaneo, se procede al filtrado de puntos, dando lugar al siguiente resultado:

```
Se ha iniciado el filtrado de puntos con 18281 valores
Tiempo empleado por el filtro: 797.061706 segundos
Se han filtrado 8 valores, quedando un total de 18273 puntos
```

Tras este proceso, la nube de puntos se reduce, habiendo eliminado los puntos más aislados y dando lugar al siguiente gráfico, el cual es muy parecido al anterior ya que el resultado del muestreo era bastante acertado:

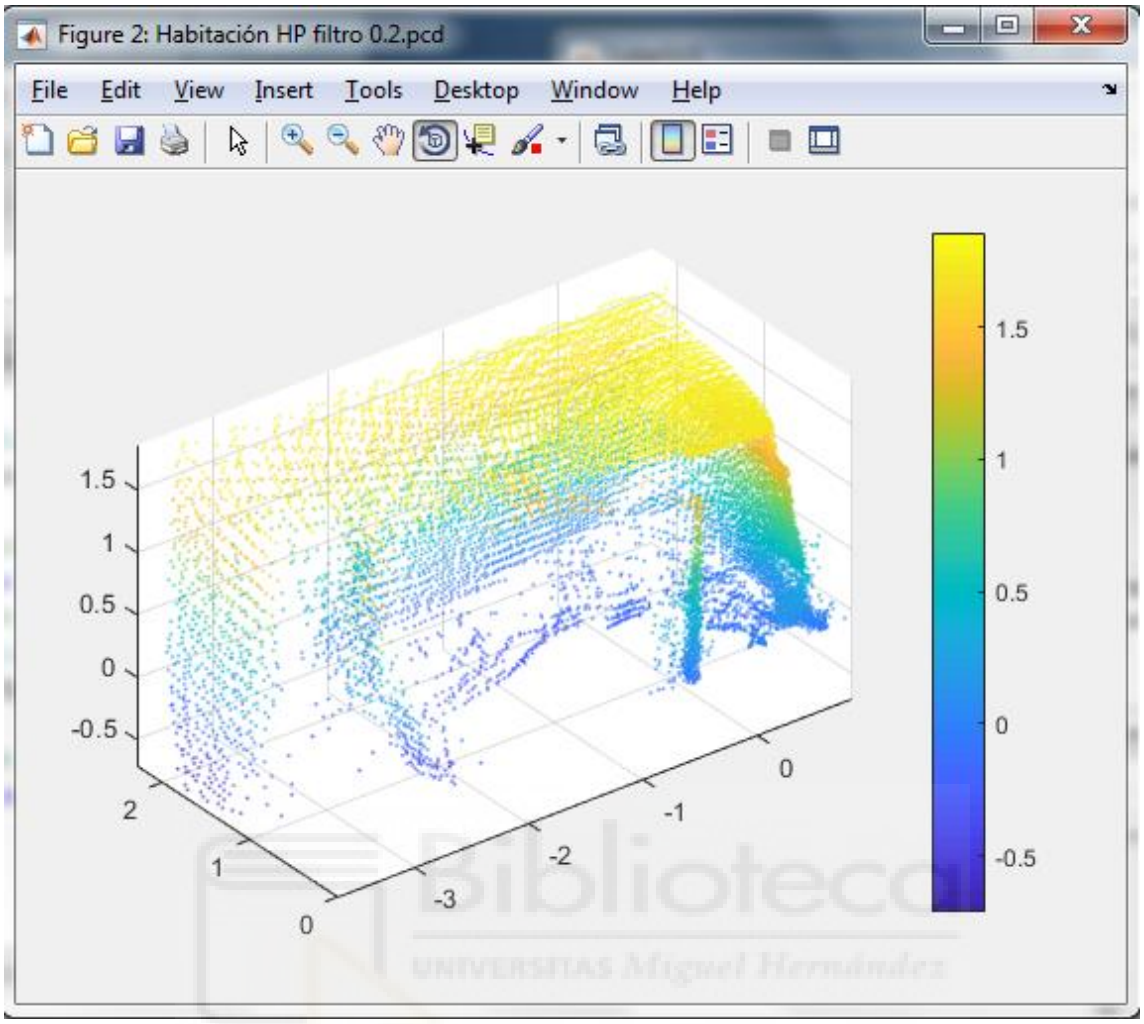


Ilustración 42: Nube de puntos de la habitación con el dispositivo lídar de alto rendimiento tras el filtrado de puntos con coeficiente 0,2

Por tanto, en vista a estos últimos resultados se puede deducir que el lídar de alto rendimiento es mucho más preciso que el de bajo rendimiento ya que las paredes reconstruidas por éste son más finas que las del otro y registra menos valores aislados. Por ende, este dispositivo reproduce resultados más fieles a la realidad.

6.2.2. Precisión

Dada la evolución del error según la longitud a la que se encuentra el lídar del obstáculo, tal y como se explica en el apartado 4.1.2, es necesario realizar distintas pruebas de precisión a diferentes longitudes para averiguar cómo influye la distancia con respecto al resultado de los datos. A continuación, se procede a realizar las correspondientes pruebas de precisión del dispositivo, utilizando para ello un muestreo de 100 puntos. La primera, se realiza a una distancia de 40 cm del origen de coordenadas, dando lugar a los siguientes resultados:

```
Resultados de la prueba de precisión:
MEDIA: 0.430300 metros
DESVIACIÓN TÍPICA: 0.032810 metros
```

Se puede percibir que la media difiere en 3 cm de la distancia real y que la desviación típica es de cerca de 3 cm, pero un poco superior al dispositivo anterior. Posteriormente, se realiza la misma prueba a 2 m de distancia del origen de coordenadas, dando como resultado los siguientes datos:

```
Resultados de la prueba de precisión:
MEDIA: 2.030900 metros
DESVIACIÓN TÍPICA: 0.007823 metros
```

Por tanto, ahora la media es en proporción mucho más cercana al valor real, siendo la diferencia de 3 cm igualmente, mientras que la desviación típica ha decrecido, siendo de cerca de 8 mm. Y finalmente, se realiza la misma prueba a 5 m, dando el siguiente resultado:

```
Resultados de la prueba de precisión:
MEDIA: 4.943400 metros
DESVIACIÓN TÍPICA: 0.018825 metros
```

Por tanto, se puede deducir que, teniendo una diferencia de 6 cm de la media frente al valor real y una desviación de cerca de 2 cm, el lídar presenta una precisión bastante apropiada para la aplicación de escaneo.

A continuación, se presenta una gráfica que muestra los resultados en cuanto a la prueba de precisión. Al parecer, en este caso la tendencia de los datos sí que es lineal, debido a la alta precisión del dispositivo.

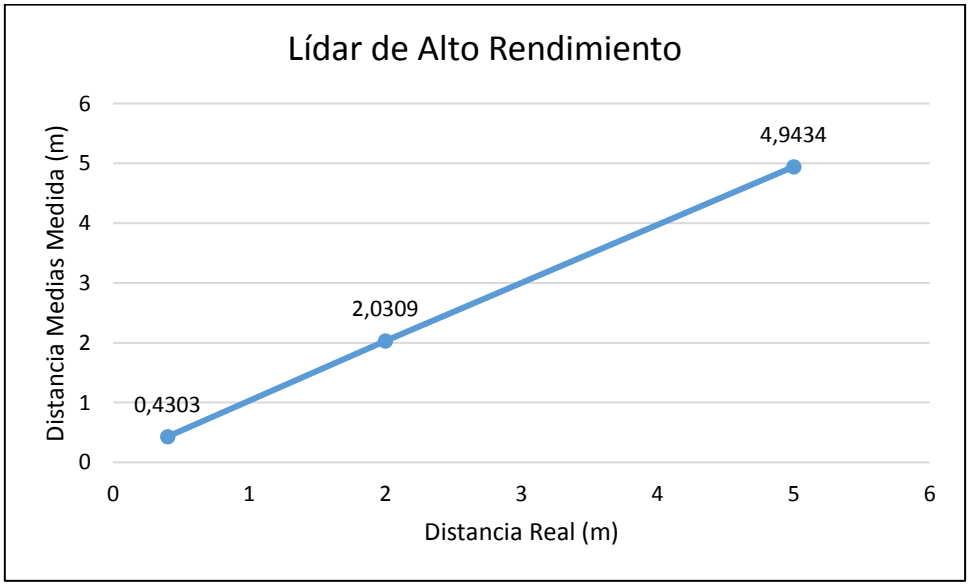


Ilustración 43: Gráfica que muestra la tendencia de la media de las medidas del lídar de alto rendimiento

7. Conclusiones y Líneas Futuras

En este trabajo se ha implementado el montaje y construcción de un sistema lídar de 2 grados de libertad, además de desarrollar un algoritmo de control para la placa Arduino Uno R3 capaz de controlar cada componente de forma independiente. Para ello, también, se han desarrollado algunos objetivos específicos.

Primero, se ha creado un interfaz gráfico interactivo en MatLab que permite controlar y parametrizar el funcionamiento del sistema lídar. Por otro lado, se ha implantado la librería Point Cloud de MatLab, que permite visualizar, cargar y guardar nubes de puntos escaneadas en archivos PCD.

Tras haber realizado todo esto, se ha elaborado un estudio de diferentes dispositivos lídar respecto de su precisión y el análisis cualitativo del resultado de sus escaneos.

7.1. Conclusión

En vista a los resultados obtenidos a través de este proyecto, se puede concluir que los datos obtenidos a través del lídar de bajo rendimiento difieren demasiado de la realidad y, por lo tanto, no es apto para la aplicación de escaneo que propone el presente proyecto. Por otro lado, el lídar de alto rendimiento proporciona resultados mucho más aproximados a la realidad y podría ser utilizado para dicha aplicación, siempre que no se requiera de una alta precisión.

A este error de medición, se le debe sumar también el error de posición de los servomotores, pues tal y como se ha explicado en el apartado 4.3, los servomotores no disponen de ningún dispositivo que verifique la posición real de éstos y, por tanto, no se puede establecer ningún algoritmo de control que ayude a corregir dicha desviación angular.

También, cabe destacar que debido a la acumulación de todos estos errores de medición, tanto por parte de los servomotores como por parte de los dispositivos lídar, la nubes de puntos resultantes en el apartado 6 corresponden a nubes de puntos ligeramente desplazadas angularmente respecto del origen de coordenadas, es decir, giradas en torno a los ejes de movimiento de los servomotores.

Finalmente, ha de hacerse una mención en torno al filtro, y es que tarda demasiado tiempo en procesar la nube de puntos entera, más incluso que el propio escaneo cuando la nube de datos es muy grande.

7.2. Trabajos Futuros

En vistas a mejorar y desarrollar nuevas funcionalidades para el presente proyecto, han de tenerse varios conceptos en mente. Para empezar, la necesidad fundamental para mejorar la precisión del sistema actual sería definir un método de calibración propio, ya sea para la implementación en conjunto de todo el sistema (matrices de transformación homogéneas) o bien para corregir el error parte por parte, es decir, de cada componente por separado. Otra opción sería abrir una investigación sobre métodos aplicados a sistemas similares que ya se estén utilizando a día de hoy, es decir, un estudio del estado del arte en cuanto a calibración de sistemas lídar y su posterior implementación en el presente proyecto.

Respecto a la opción de corregir el error para cada componente por separado, debería considerarse la opción de añadir codificadores rotacionales u otro tipo de sensor para conocer la posición real de los servomotores y reducir su error de posición, eliminando así los desplazamientos angulares de la nube de puntos mencionados en el apartado 7.1.

Otra idea es utilizar un mejor soporte más estable y pesado, con una base de alta fricción con la superficie (goma o similar) para que no pueda moverse durante el escaneo. También, se recomienda que sea ajustable para que pueda adquirir deferentes inclinaciones y así corregir la propia inclinación de la base sobre la que se sitúe, pudiendo así quedar los ejes de coordenadas X e Y paralelos al suelo y el Z perpendicular a éste. Para ello, basta con añadir un inclinómetro de burbuja de aire.

Otra opción que se propone es eliminar el botón “Dibujar” y utilizar únicamente el botón “Mostrar” del PCD, ya que el comando “`plot3`” es visualmente más incómodo y no es necesario guardar un PCD para utilizar la función “`pcshow`” del botón “Mostrar”. Para ello, habría que añadir un paso intermedio transformando los valores del escaneo a formato `pointcloud`, y adaptar la función del filtrado usando el atributo `Location` de los datos y después trasponerlo.

Por último, se recomienda añadir un algoritmo de calibración de los datos para conocer la localización y orientación real de los ejes de coordenadas del sistema. La idea consiste en establecer un patrón fácilmente reconocible en una nube de puntos y utilizarlo para calcular la orientación y posición (pose) de los ejes de coordenadas de la estructura lídar. Mediante este procedimiento, se pretende saber si la inclinación del soporte es la correcta y cuál es la desviación de los centros angulares (0º) de los dos servomotores. Es preciso realizar esta calibración cada vez que se desplace la estructura lídar o cambie de orientación, para así establecer la debida corrección.

8. Bibliografía

[1] Wikipedia, «Radar,» https://es.wikipedia.org/wiki/Radar, 2019.

[2] Wikpedia, «Sónar,» https://es.wikipedia.org/wiki/Sonar, 2019.

[3] Wikipedia, «Láser,» https://es.wikipedia.org/wiki/L%C3%A1ser, 2019.

[4] Wikipedia, «Radiación infrarroja,» https://es.wikipedia.org/wiki/Radiaci%C3%B3n_infrarroja, 2019.

[5] Sigsa, «Lidar,» http://www.sigsa.info/geotecnologias/lidar.

[6] Wikipedia, «Lidar,» https://en.wikipedia.org/wiki/Lidar#History_and_etymology, 2019.

[7] L. Jingyun, S. Qiao, F. Zhe y J. Yudong, «TOF Lidar Development in Autonomous Vehicle,» https://ieeexplore-ieee-org.publicaciones.umh.es/document/8529992, 2018.

[8] Velodyne, «Lídar HDL-64E,» https://velodynelidar.com/hdl-64e.html.

[9] SureStar, «Lídar C-Fans-32/128,» https://www.isurestar.com/en/cfans-91.html.

[10] Benewake, «CE30 Solid-state LiDAR(3D Solid-state LiDAR),» http://en.benewake.com/product/detail/5c34571eadd0b639f4340ce5.html.

[11] Continental Automotive, «High Resolution 3D Flash LiDAR,» https://www.continental-automotive.com/en-gl/Passenger-Cars/Chassis-Safety/Advanced-Driver-Assistance-Systems/Lidars/High-Resolution-3D-Flash-Lidar.

[12] LeddarTech, «Lídar Leddar M16,» https://leddartech.com/lidar/m16-multi-segment-sensor-module/.

[13] Wikipedia, «Time of flight,» https://en.wikipedia.org/wiki/Time_of_fligh8, 2018.

[14] C. Edoardo y P. Preethi, «Direct time-of-flight (D-TOF) image sensor for LiDAR applications,» https://aqua.epfl.ch/research/single-photon-sensor-architectures/page-157535-en-html/, 2019.

[15] D. Peters, «Arduino-based LIDAR Scanner,» http://www.qcontinuum.org/lidar.

[16] Arduino, «Especificaciones Arduino Uno Rev 3,» https://store.arduino.cc/arduino-uno-rev3.

[17] Lynxmotion, «Lynx B - Pan and Tilt Assembly Instructions,» http://www.lynxmotion.com/images/html/build153.htm, 2011.

9. Anexos

9.1. Código de Arduino

```
// LidarScanner.ino Arduino sketch
// http://www.gcontinuum.org/lidar

// Load sketch into Arduino software, available from:
// https://www.arduino.cc/

// This sketch controls X and Y servos to pan/tilt a LIDAR detector,
// either manually (by pressing Serial buttons to control XY location),
// or automatically (scanning horizontally and vertically).
// XYZ coordinates are output to the serial port to be received and
// displayed on computer by LidarScanner.pde Processing sketch.

// This sketch requires library "LIDAR-Lite v3" by Garmin.
// Select menu "Sketch", "Include Library", "Manage Libraries...",
// and in textbox "Filter your search...", enter "lidar".

// Freetronics Serial shield uses D3 to control backlight brightness,
// but digital output needed for servo control, so disable the
// backlight control by cutting strap marked "D3" on Serial shield.

#include <Servo.h>
#include <LIDARLite.h>
#include <LiquidCrystal.h>

Servo servoX;
Servo servoY;
LIDARLite lidar;
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);
// Minimum and maximum servo angle in degrees
// Modify to avoid hitting limits imposed by pan/tilt bracket geometry
int minAzimut = 0;
int maxAzimut = 180;
int minElevacion = 80;
int maxElevacion = 180;

int buttonPin = A0;
int buttonValue = 0;
int buttonThreshold = 50;
int lastAzimut = 0;
int lastElevacion = 0;
int loopCount = 0;
int radius = 0;
int lastRadius = 0;
int n_max_escaneos = 0;
int n_escaneos = 0;
int tiempoEspera = 0;
String cadena;
unsigned int retardo = 5;
boolean selectButtonPressed = false;
boolean scanning = false;
boolean scanDirection = false;
int scanIncrementAzimut = 1;
int scanIncrementElevacion = 1;
int Azimut = 90;
int Elevacion = 90;
float pi = 3.14159265;
float deg2rad = pi / 180.0;
```

```
float CentroOptico = 6.8; // cm

void setup() {
  // Se ajusta el LED de la placa para que permanezca apagado
  pinMode(13, OUTPUT);
  digitalWrite(13, LOW);

  lcd.begin(16, 2);
  lidar.begin(0, true);
  lidar.configure(0);
  servoX.attach(2);
  servoY.attach(3);
  // Posición frontal
  servoX.write(Azimut);
  servoY.write(Elevacion);
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }
  updateModeDisplay();
}

void loop() {

  buttonValue = analogRead(buttonPin);

  if (scanning) { // Modo de escaneo
    if (abs(buttonValue - 741) < buttonThreshold) { // Botón SELECT pulsado
      if (!selectButtonPressed) {
        // switch to manual scan mode
        selectButtonPressed = true;
        scanning = false;
        updateModeDisplay();
      }
    } else {
      selectButtonPressed = false;
    }
    if (scanDirection) {
      Azimut += scanIncrementAzimut;
    } else {
      Azimut -= scanIncrementAzimut;
    }
    if (Azimut > maxAzimut || Azimut < minAzimut) {
      // hit limit azimut limit, reverse auto scan direction
      scanDirection = !scanDirection;
      Elevacion += scanIncrementElevacion;
      if (Elevacion > maxElevacion) { // Sobrepasa el límite de elevación
        // completed auto scan, return to manual scan mode
        scanning = false;
        updateModeDisplay();
      }
    }
    if (n_max_escaneos != 0 && n_escaneos == n_max_escaneos) { // Prueba de precisión completada
      scanning = false;
      updateModeDisplay();
    }
  } else { // Modo manual
    if (abs(buttonValue - 741) < buttonThreshold) { // Botón SELECT pulsado
```

```
    if (!selectButtonPressed) {
      // switch to auto scan mode
      selectButtonPressed = true;
      iniciar_escaneo();
    }
  } else if (abs(buttonValue - 505) < buttonThreshold) { // Botón LEFT pulsado
    // manual scan left
    if (Azimut > minAzimut) {
      Azimut -= 1;
    }
  } else if (abs(buttonValue - 329) < buttonThreshold) { // Botón DOWN pulsado
    // manual scan down
    if (Elevacion > minElevacion) {
      Elevacion -= 1;
    }
  } else if (abs(buttonValue - 145) < buttonThreshold) { // Botón UP pulsado
    // manual scan up
    if (Elevacion < maxElevacion) {
      Elevacion += 1;
    }
  } else if (abs(buttonValue - 0) < buttonThreshold) { // Botón RIGHT pulsado
    // manual scan right
    if (Azimut < maxAzimut) {
      Azimut += 1;
    }
  } else {
    selectButtonPressed = false;
  }
}
// Lectura de parámetros a través del puerto serie
if (Serial.available() > 0) {
  cadena = Serial.readStringUntil(';');
  n_max_escaneos = cadena.toInt();
  if (n_max_escaneos < 2) {
    n_max_escaneos = 0;
  }
  if (n_max_escaneos == 0) { // Escaneo normal
    cadena = Serial.readStringUntil(';');
    minAzimut = cadena.toInt();
    cadena = Serial.readStringUntil(';');
    maxAzimut = cadena.toInt();
    cadena = Serial.readStringUntil(';');
    minElevacion = cadena.toInt();
    cadena = Serial.readStringUntil(';');
    maxElevacion = cadena.toInt();
    cadena = Serial.readStringUntil(';');
    scanIncrementAzimut = cadena.toInt();
    if (scanIncrementAzimut < 0) {
      scanIncrementAzimut = 0;
    }
    cadena = Serial.readStringUntil(';');
    scanIncrementElevacion = cadena.toInt();
    if (scanIncrementElevacion < 0) {
      scanIncrementElevacion = 0;
    }
  } else { // Modo de prueba
    minAzimut = 90;
    maxAzimut = 90;
```

```
      minElevacion = 90;
      maxElevacion = 90;
      scanIncrementAzimut = 0;
      scanIncrementElevacion = 0;
    }
    cadena = Serial.readString();
    tiempoEspera = cadena.toInt();
    if (tiempoEspera < 0) {
      tiempoEspera = 0;
    }
    iniciar_escaneo();
  }

  // Ajustar movimiento dentro de los límites
  Azimut = min(max(Azimut, minAzimut), maxAzimut);
  Elevacion = min(max(Elevacion, minElevacion), maxElevacion);
  bool moved = moveServos();
  displayPosition();

  loopCount += 1;
  if (loopCount % 100 == 0) {
    // recalibrate scanner every 100 loops
    radius = lidar.distance();
  } else {
    radius = lidar.distance(false);
  }
  n_escaneos += 1;
  if (abs(radius - lastRadius) > 2)
  { // Actualiza el LCD cuando hay suficiente cambio en la distancia
    lastRadius = radius;
    lcd.setCursor(8, 0);
    lcd.println("D:" + String(radius / 100.0, 2) + "  ");
  }
  // Si el motor se mueve o se ha medido un dato
  if (scanning || moved)
    float azimuth = Azimut * deg2rad;
    float elevation = Elevacion * deg2rad;
    double x = (radius + CentroOptico) * sin(elevation) * -cos(azimuth);
    double y = (radius + CentroOptico) * sin(elevation) * sin(azimuth);
    double z = (radius + CentroOptico) * -cos(elevation);
    lcd.println(String(x, 5) + " " + String(y, 5) + " " + String(z, 5));
    Serial.println(String(x, 5) + ";" + String(y, 5) + ";" + String(z, 5));
  }
}

void iniciar_escaneo() {
  scanning = true;
  Azimut = minAzimut;
  Elevacion = minElevacion;
  scanDirection = true;
  n_escaneos = 0;
  delay(tiempoEspera);
  updateModeDisplay();
}

bool moveServos() {
  bool moved = false;
  static int lastAzimut;
```

```
  static int lastElevacion;
  int movimientoX = 0;
  int movimientoY = 0;
  if (Azimut != lastAzimut) {
    movimientoX = abs(Azimut - lastAzimut);
    servoX.write(Azimut);
    lastAzimut = Azimut;
    moved = true;
  }
  if (Elevacion != lastElevacion) {
    movimientoY = abs(Elevacion - lastElevacion);
    servoY.write(Elevacion);
    lastElevacion = Elevacion;
    moved = true;
  }
  // Realiza un retardo relativo al mayor desplazamiento
  retardo = max(movimientoX, movimientoY) * 5;
  delay(retardo);
  return moved;
}

void displayPosition() {
  static int lastAzimut;
  static int lastElevacion;
  if (Azimut != lastAzimut) {
    lcd.setCursor(0, 0);
    lcd.println("X:" + String(Azimut) + "  ");
    lastAzimut = Azimut;
  }
  if (Elevacion != lastElevacion) {
    lcd.setCursor(0, 1);
    lcd.println("Y:" + String(Elevacion) + "  ");
    lastElevacion = Elevacion;
  }
}

void updateModeDisplay() {
  lcd.setCursor(8, 1);
  if (scanning) {
    lcd.println("Scanning");
    Serial.println("Scanning");
  } else {
    lcd.println("Manual  ");
    Serial.println("Manual");
  }
}
```

9.2. Código de MatLab

9.2.1. Init_lidar.m

```
clc
% Borrar posibles bucles paralelos abiertos
delete(gcp('nocreate'));

global Lidar

if ismac % Mac
    Lidar.puerto = '/dev/cu.usbmodem14101';
elseif ispc % Windows
    Lidar.puerto = 'COM6';
else
    error('Sistema Operativo no compatible')
end
% Buscar dispositivos conectados en el puerto "Lidar.puerto"
Lidar.arduino = instrfind({'Port'},{Lidar.puerto});

if ~isempty(Lidar.arduino) % Si hay alguno conectado
    fclose(Lidar.arduino);
    delete(Lidar.arduino)
    fprintf('Se ha cerrado un puerto de comunicación que había permanecido abierto\n')
end
Lidar.arduino = [];
% Abrir GUI
LidarGUI
% Bucle principal
while ~Lidar.cerrar
    % Esperar interacción con el GUI
    uiwait(LidarGUI)
    if Lidar.cerrar
        break
    end
    if ~Lidar.desconectar
    funciones.conexion()
    end
    % Bucle de comunicación (inhabilita el GUI)
    while ~Lidar.desconectar
        % Recibe los datos del buffer de entrada
        Lidar.comando = fscanf(Lidar.arduino,'%s\r\n');

        % Procesado de los datos
        switch Lidar.comando
            case 'Scanning'
                fprintf('\nIniciando escaneo automático\n')
                Lidar.escaneo = true;
                Lidar.rastreado = true;
                Lidar.nubePuntos = [];
                % Inicia contador
                Lidar.temporizador = true;
                tic
            case 'Manual'
                if Lidar.temporizador
                    fprintf('\nTiempo de escaneo: %f segundos\n',toc)
                    Lidar.temporizador = false;
                end
                fprintf('\nIniciando modo manual\n')
                Lidar.escaneo = false;
                % Comprueba si se ha cumplido con un escaneo
                if Lidar.rastreado
                    Lidar.desconectar = true;
                end
            otherwise
                % Si han llegado datos (buffer no vacío)
```

```
                if ~isempty(Lidar.comando)
                    % Recopila los datos X, Y y Z
                    Lidar.celda = textscan(Lidar.comando,'%f;%f;%f','CollectOutput',1);
                    Lidar.posicion = Lidar.celda{1};
                    if ~isempty(Lidar.posicion)
                        % Si son datos válidos
                        if isnumeric(Lidar.posicion) && ~any(isnan(Lidar.posicion))
                            fprintf('\nPosición X: %f\nPosición Y: %f\nPosición Z: %f\n'...
                                ,Lidar.posicion(1),Lidar.posicion(2),Lidar.posicion(3))
                            % Si está escaneando
                            if Lidar.escaneo
                                Lidar.nubePuntos = [Lidar.nubePuntos Lidar.posicion./100']; % Conversión cm -> m
                            end
                        else
                            warning('\nFormato de posición incorrecto')
                            disp(Lidar.posicion)
                        end
                    end
                end
            end
        end
        % Desconexión automática
        if ~Lidar.desconectado
        funciones.desconexion()
        end
        % Si se está realizando la prueba de precisión
        if Lidar.prueba
            Lidar.prueba = false;
            funciones.precision()
        else
            % Si el filtro automático está activado
            if Lidar.filtro
                Lidar.nubePuntos = funciones.filtrar(Lidar.nubePuntos);
            end
            funciones.dibujar()
        end
    end
    Lidar.cerrado = true;
    close(LidarGUI)
```

9.2.2. LidarGUI.m

```
function varargout = LidarGUI(varargin)
% LIDARGUI MATLAB code for LidarGUI.fig
%      LIDARGUI, by itself, creates a new LIDARGUI or raises the existing
%      singleton*.
%
%      H = LIDARGUI returns the handle to a new LIDARGUI or the handle to
%      the existing singleton*.
%
%      LIDARGUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in LIDARGUI.M with the given input arguments.
%
%      LIDARGUI('Property','Value',...) creates a new LIDARGUI or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before LidarGUI_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to LidarGUI_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See alLidar.SO: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help LidarGUI

% Last Modified by GUIDE v2.5 06-Aug-2019 18:19:10

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @LidarGUI_OpeningFcn, ...
                   'gui_OutputFcn',  @LidarGUI_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before LidarGUI is made visible.
function LidarGUI_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to LidarGUI (see VARARGIN)
global Lidar

Lidar.desconectar = true;
Lidar.desconectado = true;
Lidar.cerrado = false;
Lidar.cerrar = false;
Lidar.rastreado = false;
Lidar.temporizador = false;
```

```
% Límites
Lidar.limites.X.max = 180;
Lidar.limites.X.min = 0;
Lidar.limites.Y.max = 180;
Lidar.limites.Y.min = 80;

% Choose default command line output for LidarGUI
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes LidarGUI wait for user response (see UIRESUME)


% --- Outputs from this function are returned to the command line.
function varargout = LidarGUI_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in BotonConexion.
function BotonConexion_Callback(hObject, eventdata, handles)
% hObject    handle to BotonConexion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Lidar
Lidar.desconectar = false;
guidata(hObject, handles);
uiresume(gcbf)

% --- Executes on button press in CasillaFiltro.
function CasillaFiltro_Callback(hObject, eventdata, handles)
% hObject    handle to CasillaFiltro (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Lidar
if get(hObject,'Value') == get(hObject,'Max')
    Lidar.filtro = true;
elseif get(hObject,'Value') == get(hObject,'Min')
    Lidar.filtro = false;
else
    error('Valor en CasillaFiltro incorrecto')
end
guidata(hObject, handles);

% --- Executes on button press in BotonCerrar.
function BotonCerrar_Callback(hObject, eventdata, handles)
% hObject    handle to BotonLidar.cerrar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Lidar
Lidar.cerrar = true;
guidata(hObject, handles);
uiresume(gcbf)
% close(LidarGUI);

function AzimutMin_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to AzimutMin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Lidar
numero = str2double(get(hObject,'String'));
if ~isempty(numero) % Si se convierte a número (str2double)
    numero = real(numero);
    if numero < Lidar.limites.X.min
        numero = Lidar.limites.X.min;
    elseif numero > Lidar.limites.X.max
        numero = Lidar.limites.X.max;
    end

    if numero > Lidar.AzimutMax
        numero = Lidar.AzimutMax;
    end
    Lidar.AzimutMin = floor(numero);
end
set(hObject,'String',num2str(Lidar.AzimutMin));
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function AzimutMin_CreateFcn(hObject, eventdata, handles)
% hObject    handle to AzimutMin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
global Lidar
if isfield(Lidar,'AzimutMin')
    set(hObject,'String',num2str(Lidar.AzimutMin));
else
    Lidar.AzimutMin = str2double(get(hObject,'String'));
end
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
guidata(hObject, handles);

function AzimutMax_Callback(hObject, eventdata, handles)
% hObject    handle to AzimutMax (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Lidar
numero = str2double(get(hObject,'String'));
if ~isempty(numero) % Si se convierte a número (str2double)
    numero = real(numero);
    if numero < Lidar.limites.X.min
        numero = Lidar.limites.X.min;
    elseif numero > Lidar.limites.X.max
        numero = Lidar.limites.X.max;
    end

    if numero < Lidar.AzimutMin
        numero = Lidar.AzimutMin;
    end
    Lidar.AzimutMax = floor(numero);
end
set(hObject,'String',num2str(Lidar.AzimutMax));
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
```

```
function AzimutMax_CreateFcn(hObject, eventdata, handles)
% hObject    handle to AzimutMax (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
global Lidar
if isfield(Lidar,'AzimutMax')
    set(hObject,'String',num2str(Lidar.AzimutMax));
else
    Lidar.AzimutMax = str2double(get(hObject,'String'));
end
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
guidata(hObject, handles);

function ElevacionMin_Callback(hObject, eventdata, handles)
% hObject    handle to ElevacionMin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Lidar
numero = str2double(get(hObject,'String'));
if ~isempty(numero) % Si se convierte a número (str2double)
    numero = real(numero);
    if numero < Lidar.limites.Y.min
        numero = Lidar.limites.Y.min;
    elseif numero > Lidar.limites.Y.max
        numero = Lidar.limites.Y.max;
    end
    if numero > Lidar.ElevacionMax
        numero = Lidar.ElevacionMax;
    end
    Lidar.ElevacionMin = floor(numero);
end
set(hObject,'String',num2str(Lidar.ElevacionMin));
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function ElevacionMin_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ElevacionMin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
global Lidar
if isfield(Lidar,'ElevacionMin')
    set(hObject,'String',num2str(Lidar.ElevacionMin));
else
    Lidar.ElevacionMin = str2double(get(hObject,'String'));
end
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
guidata(hObject, handles);

function ElevacionMax_Callback(hObject, eventdata, handles)
% hObject    handle to ElevacionMax (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Lidar
```

```
numero = str2double(get(hObject,'String'));
if ~isempty(numero) % Si se convierte a número (str2double)
    numero = real(numero);
    if numero < Lidar.limites.Y.min
        numero = Lidar.limites.Y.min;
    elseif numero > Lidar.limites.Y.max
        numero = Lidar.limites.Y.max;
    end

    if numero < Lidar.ElevacionMin
        numero = Lidar.ElevacionMin;
    end
    Lidar.ElevacionMax = floor(numero);
end
set(hObject,'String',num2str(Lidar.ElevacionMax));
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function ElevacionMax_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ElevacionMax (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
global Lidar
if isfield(Lidar,'ElevacionMax')
    set(hObject,'String',num2str(Lidar.ElevacionMax));
else
    Lidar.ElevacionMax = str2double(get(hObject,'String'));
end
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
guidata(hObject, handles);

function PasoAzimut_Callback(hObject, eventdata, handles)
% hObject    handle to PasoAzimut (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Lidar
numero = str2double(get(hObject,'String'));
if ~isempty(numero) % Si se convierte a número (str2double)
    numero = real(numero);
    if numero < 1
        numero = 1;
    elseif numero > 180
        numero = 180;
    end
    Lidar.PasoAzimut = floor(numero);
end
set(hObject,'String',num2str(Lidar.PasoAzimut));
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function PasoAzimut_CreateFcn(hObject, eventdata, handles)
% hObject    handle to PasoAzimut (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
global Lidar
if isfield(Lidar,'PasoAzimut')
    set(hObject,'String',num2str(Lidar.PasoAzimut));
else
```

```
    Lidar.PasoAzimut = str2double(get(hObject,'String'));
end

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function CasillaFiltro_CreateFcn(hObject, eventdata, handles)
global Lidar
if isfield(Lidar,'filtro')
    if Lidar.filtro
        set(hObject,'Value',get(hObject,'Max'));
    else
        set(hObject,'Value',get(hObject,'Min'));
    end
else
    if get(hObject,'Value') == get(hObject,'Max')
        Lidar.filtro = true;
    elseif get(hObject,'Value') == get(hObject,'Min')
        Lidar.filtro = false;
    else
        error('Valor en BotonFiltro incorrecto')
    end
end
% hObject    handle to CasillaFiltro (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function BotonConexion_CreateFcn(hObject, eventdata, handles)
% hObject    handle to BotonConexion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

function tiempoEspera_Callback(hObject, eventdata, handles)
% hObject    handle to tiempoEspera (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Lidar
numero = str2double(get(hObject,'String'));
if ~isempty(numero) % Si se convierte a número (str2double)
    numero = real(numero);
    if numero < 0
        numero = 0;
    elseif numero > 32.767
        numero = 32.767;
    end
    Lidar.tiempoEspera = round(numero,3);
end
set(hObject,'String',num2str(Lidar.tiempoEspera));
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function tiempoEspera_CreateFcn(hObject, eventdata, handles)
% hObject    handle to tiempoEspera (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
global Lidar
if isfield(Lidar,'tiempoEspera')
    set(hObject,'String',num2str(Lidar.tiempoEspera));
else
    Lidar.tiempoEspera = str2double(get(hObject,'String'));
end
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function PasoElevacion_Callback(hObject, eventdata, handles)
% hObject    handle to PasoElevacion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Lidar
numero = str2double(get(hObject,'String'));
if ~isempty(numero) % Si se convierte a número (str2double)
    numero = real(numero);
    if numero < 1
        numero = 1;
    elseif numero > 180
        numero = 180;
    end
    Lidar.PasoElevacion = floor(numero);
end
set(hObject,'String',num2str(Lidar.PasoElevacion));
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function PasoElevacion_CreateFcn(hObject, eventdata, handles)
% hObject    handle to PasoElevacion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
global Lidar
if isfield(Lidar,'PasoElevacion')
    set(hObject,'String',num2str(Lidar.PasoElevacion));
else
    Lidar.PasoElevacion = str2double(get(hObject,'String'));
end

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function coeficienteFiltro_Callback(hObject, eventdata, handles)
% hObject    handle to coeficienteFiltro (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Lidar
numero = str2double(get(hObject,'String'));
if ~isempty(numero) % Si se convierte a número (str2double)
    numero = real(numero);
    if numero < 0.001
        numero = 0.001;
    elseif numero > 32767 % 2^15 - 1
        numero = 32767;
    end
    Lidar.coefFiltro = numero;
```

```
end
set(hObject,'String',num2str(Lidar.coefFiltro));
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function coeficienteFiltro_CreateFcn(hObject, eventdata, handles)
% hObject    handle to coeficienteFiltro (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
global Lidar
if isfield(Lidar,'coefFiltro')
    set(hObject,'String',num2str(Lidar.coefFiltro));
else
    Lidar.coefFiltro = str2double(get(hObject,'String'));
end
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function BotonFiltro_Callback(hObject, eventdata, handles)
global Lidar
% hObject    handle to coeficienteFiltro (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
Lidar.nubePuntos = funciones.filtrar(Lidar.nubePuntos);
guidata(hObject, handles);

function BotonFiltro_CreateFcn(hObject, eventdata, handles)
% hObject    handle to coeficienteFiltro (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in BotonPrueba.
function BotonPrueba_Callback(hObject, eventdata, handles)
% hObject    handle to BotonPrueba (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Lidar
Lidar.prueba = true;
Lidar.desconectar = false;
guidata(hObject, handles);
uiresume(gcbf)

% --- Executes during object creation, after setting all properties.
function BotonPrueba_CreateFcn(hObject, eventdata, handles)
% hObject    handle to BotonPrueba (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
global Lidar
Lidar.prueba = false;

function nPuntosPrueba_Callback(hObject, eventdata, handles)
% hObject    handle to nPuntosPrueba (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Lidar
numero = str2double(get(hObject,'String'));
if ~isempty(numero) % Si se convierte a número (str2double)
    numero = real(numero);
    if numero < 2
```

```
        numero = 2;
    elseif numero > 32767 % 2^15 - 1
        numero = 32767;
    end
    Lidar.coefFiltro = floor(numero);
end
set(hObject,'String',num2str(Lidar.coefFiltro));
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function nPuntosPrueba_CreateFcn(hObject, eventdata, handles)
% hObject    handle to nPuntosPrueba (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
global Lidar
if isfield(Lidar,'puntosPrueba')
    set(hObject,'String',num2str(Lidar.puntosPrueba));
else
    Lidar.puntosPrueba = str2double(get(hObject,'String'));
end
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in botonGuardarPCD.
function botonGuardarPCD_Callback(hObject, eventdata, handles)
% hObject    handle to botonGuardarPCD (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Lidar
if isfield(Lidar,'nubePuntos')
    if ~isempty(Lidar.nubePuntos)
        Lidar.nubePCD.datos = pointCloud(Lidar.nubePuntos');
        Lidar.nubePCD.nombre = Lidar.nombreArchivo;
        pcwrite(Lidar.nubePCD.datos,[Lidar.nubePCD.nombre '.pcd']);
        fprintf('\nGuardado con éxito\n')
    else
        fprintf('\nNo hay datos que guardar\nRealiza un escaneo para guardar la información\n')
    end
else
    fprintf('\nNo hay datos que guardar\nRealiza un escaneo para guardar la información\n')
end

function nombrePCD_Callback(hObject, eventdata, handles)
% hObject    handle to nombrePCD (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Lidar
Lidar.nombreArchivo = get(hObject,'String');
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function nombrePCD_CreateFcn(hObject, eventdata, handles)
% hObject    handle to nombrePCD (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
global Lidar
if isfield(Lidar,'nombreArchivo')
    set(hObject,'String',Lidar.nombreArchivo);
else
```

```
    Lidar.nombreArchivo = get(hObject,'String');
end
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in botonMostrarPCD.
function botonMostrarPCD_Callback(hObject, eventdata, handles)
global Lidar
% hObject    handle to botonMostrarPCD (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if isfield(Lidar,'nubePCD')
    if ~isempty(Lidar.nubePCD.datos)
        figure('Name',[Lidar.nubePCD.nombre '.pcd'])
        pcshow(Lidar.nubePCD.datos);
    else
        fprintf('\nNo hay datos que mostrar\n"Carga" un archivo .pcd o "Guarda"')
        fprintf('los datos de un escaneo para visualizar la información\n')
    end
else
    fprintf('\nNo hay datos que mostrar\n"Carga" un archivo .pcd o "Guarda"')
    fprintf('los datos de un escaneo para visualizar la información\n')
end

% --- Executes during object creation, after setting all properties.
function botonGuardarPCD_CreateFcn(hObject, eventdata, handles)
% hObject    handle to botonGuardarPCD (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% --- Executes on button press in BotonDibujar.
function BotonDibujar_Callback(hObject, eventdata, handles)
% hObject    handle to BotonDibujar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
funciones.dibujar()

% --- Executes on button press in BotonCargarPCD.
function BotonCargarPCD_Callback(hObject, eventdata, handles)
% hObject    handle to BotonCargarPCD (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Lidar
Lidar.nubePCD.datos = pcread([Lidar.nombreArchivo '.pcd']);
Lidar.nubePCD.nombre = Lidar.nombreArchivo;
Lidar.nubePuntos = Lidar.nubePCD.datos.Location';
fprintf('\nCargado con éxito\n')
```

9.2.3. Funciones.m

```
classdef funciones
    methods(Static)
        function Puntos = filtrar(Puntos)
            % Filtro de puntos por cercanía relativa a otros
            % Si hay por lo menos dos puntos
            if ~isempty(Puntos) && size(Puntos,2) > 1
                n_puntos_inicial = size(Puntos,2);
                fprintf('\nSe ha iniciado el filtrado de puntos con %d valores\n',n_puntos_inicial)
                % Se inicia el temporizador
                tic
                % Busca encontrar para cada punto de la nube almenos un
                % punto cercano (distancia euclídea < radio)

                % Compara puntos en un doble bucle
                i = 1;
                while i <= size(Puntos,2)
                    j = 1;
                    while j <= size(Puntos,2)
                        % Comprueba si son distantes
                        if funciones.distancia(Puntos(:,i),Puntos(:,j))
                            j = j + 1;
                        else
                            break
                        end
                    end
                    % Si se sobrepasa el tamaño de la nube, significa que
                    % no a encontrado ningún punto cercano y por tanto es
                    % aislado
                    if j > size(Puntos,2)
                        % Se elimina dicho punto
                        Puntos(:,i:(size(Puntos,2)-1)) = Puntos(:,(i+1):size(Puntos,2));
                        Puntos(:,size(Puntos,2)) = [];
                        i = i - 1;
                    end
                    i = i + 1;
                end
                fprintf('Tiempo empleado por el filtro: %f segundos\n',toc)
                n_puntos_final = size(Puntos,2);
                fprintf('Se han filtrado %d valores, quedando un total de %d puntos\n',...
                    n_puntos_inicial-n_puntos_final,n_puntos_final)
            end
        end

        function conexion()
            % Conecta el entorno de MatLab con el lídar
            global Lidar
            % Procesado
            Lidar.escaneo = false;
            Lidar.nubePuntos = [];

            Lidar.arduino = serial(Lidar.puerto,'BaudRate',9600,'DataBits',8,...
                'Parity','none','StopBits',1,'Terminator','CR');
            Lidar.arduino.BreakInterruptFcn = @notificar;
            Lidar.arduino.ErrorFcn = @notificar;
            fopen(Lidar.arduino);
            fprintf('Se ha abierto el puerto de comunicación\n')
            get(Lidar.arduino)

            % Esperar al establecimiento de conexión
            % recibiendo el primer comando
            while true
                Lidar.comando = fscanf(Lidar.arduino,'%s\r\n');
```

```
        if ~isempty(Lidar.comando)
            break
        end
    end
    fprintf('\nPrimer comando recibido: %s\nListo para el intercambio de datos\n',Lidar.comando)

    % Enviar parámetros
    if Lidar.prueba % Si se está haciendo la prueba de precisión
        % El tiempo de espera se envía en milisegundos
        fprintf(Lidar.arduino,[num2str(Lidar.puntosPrueba) ';'...
            num2str(Lidar.tiempoEspera*1000)]);
        fprintf('Iniciando prueba de precisión\n')
    else
        fprintf(Lidar.arduino,['0;' num2str(Lidar.AzimutMin) ';'...
           num2str(Lidar.AzimutMax) ';' num2str(Lidar.ElevacionMin)...
           ';' num2str(Lidar.ElevacionMax) ';' num2str(Lidar.PasoAzimut)...
           ';' num2str(Lidar.PasoElevacion) ';' num2str(Lidar.tiempoEspera*1000)]);
    end

    if Lidar.tiempoEspera > 0
        fprintf('Iniciando cuenta atrás de %f segundos para el escaneo\n',Lidar.tiempoEspera)
    end

    % Conexión
    Lidar.desconectado = false;
end

function desconexion()
    % Desconecta el entorno de MatLab del lídar
    global Lidar
    fclose(Lidar.arduino);
    delete(Lidar.arduino)
    Lidar.arduino = [];
    fprintf('\nDesconexión completada\n')
    Lidar.desconectado = true;
end

function dibujar()
    % Muestra la nube de puntos escaneada
    global Lidar
    % Si nubePuntos es un campo de la estructura Lidar
    if isfield(Lidar,'nubePuntos')
        % Si la nube de puntos no está vacía
        if ~isempty(Lidar.nubePuntos)
            % Tamaño del escenario 3D
            maximo = ceil(max(max(Lidar.nubePuntos)));
            minimo = floor(min(min(Lidar.nubePuntos)));
            margen = max(abs(maximo),abs(minimo));
            % Tamaño de los ejes del lídar
            escala_ejes = 0.2*margen;
            fprintf('\nGenerando imagen 3D\n')
            tic
            figure('Name','Escaneo Automático')
            plot3(Lidar.nubePuntos(1,:),Lidar.nubePuntos(2,:),Lidar.nubePuntos(3,:),'m.')
            % Representar ejes de coordenadas
            hold on
            plot3([0 escala_ejes],[0 0],[0 0],'r-') % Eje X
            plot3([0 0],[0 escala_ejes],[0 0],'g-') % Eje Y
            plot3([0 0],[0 0],[0 escala_ejes],'b-') % Eje Z
            plot3(0,0,0,'y.') % Origen
            hold off
            title('Rastreo automático del entorno 3D (nube de puntos)')
```

```
                    xlabel('X: Ancho (m)')
                    ylabel('Y: Profundidad (m)')
                    zlabel('Z: Altura (m)')
                    axis([-margen margen -margen margen -margen margen])
                    % Perspectiva cónica
                    set(gca,'Projection','perspective')
                    fprintf('Tiempo empleado para el dibujo: %f segundos\n',toc)
                else
                    fprintf('\nNo se dispone de puntos para mostrar\nRealiza un escaneo primero\n')
                end
            else
                fprintf('\nNo se dispone de puntos para mostrar\nRealiza un escaneo primero\n')
            end
        end

        function precision()
            % Muestra los datos de precisión del lídar
            global Lidar
            fprintf('\nResultados de la prueba de precisión:\n')
            fprintf('MEDIA: %f metros\n',mean(Lidar.nubePuntos(2,:)))
            fprintf('DESVIACIÓN TÍPICA: %f metros\n',std(Lidar.nubePuntos(2,:)))
        end

        function distante = distancia(punto1,punto2)
            % Comprueba si dos puntos son distantes (se aplica al filtro)
            global Lidar
            % Si los puntos son iguales se consideran distantes
            if punto1 == punto2
                distante = true;
            else
                distante = sqrt(sum((punto1 - punto2).^2)) > Lidar.coefFiltro*sqrt(sum(punto1.^2));
            end
        end

        function notificar(obj,event)
            % Notifica el error producido durante la conexión
            global Lidar
            % Desconecta y cierra el entorno
            Lidar.desconectar = true;
            Lidar.cerrar = true;
            fprintf('Tipo de evento de cierre: %s\n',event.Type)
            fprintf('Mensaje:\n%s\n',event.Data.Message)
        end
    end
end
```

9.3. Fichas Técnicas de los Componentes

Lidar Lite v3 Operation Manual and Technical Specifications

Laser Safety

⚠ WARNING

This device requires no regular maintenance. In the event that the device becomes damaged or is inoperable, repair or service must be handled by authorized, factory-trained technicians only. Attempting to repair or service the unit on your own can result in direct exposure to laser radiation and the risk of permanent eye damage. For repair or service, contact your dealer or Garmin® for more information. This device should not be modified or operated without its housing or optics. Operating this device without a housing and optics, or operating this device with modified housing or optics that expose the laser source, may result in direct exposure to laser radiation and the risk of permanent eye damage. Removal or modification of the diffuser in front of the laser optic may result in the risk of permanent eye damage.

Use of controls or adjustments or performance of procedures other than those specified in this documentation may result in hazardous radiation exposure. Garmin is not responsible for injuries caused through the improper use or operation of this product.

⚠ CAUTION

This device emits laser radiation. This Laser Product is designated Class 1 during all procedures of operation. This designation means that the laser is safe to look at with the unaided eye, however it is advisable to avoid looking into the beam when operating the device and to turn off the module when not in use.

Documentation Revision Information

Rev	Date	Changes
0A	09/2016	Initial release

Table of Contents

Lidar Lite v3 Operation Manual and Technical Specifications 1
Laser Safety 1
Documentation Revision Information 1
Specifications 2
Physical 2
Electrical 2
Performance 2
Interface 2
Laser 2
Connections 2
Wiring Harness 2
Connector 2
Connector Port Identification 2
I2C Connection Diagrams 3
Standard I2C Wiring 3
Standard Arduino I2C Wiring 3
PWM Wiring 3
PWM Arduino Wiring 3
Operational Information 4
Technology 4
Theory of Operation 4
Interface 4
Initialization 4
Power Enable Pin 4
I2C Interface 4
Mode Control Pin 4
Settings 4
I2C Protocol Information 6
I2C Protocol Operation 7
Register Definitions 7
Control Register List 7
Detailed Control Register Definitions 8
Frequently Asked Questions 12
Must the device run on 5 Vdc? Can it run on 3.3 Vdc instead? 12
What is the spread of the laser beam? 12
How do distance, target size, aspect, and reflectivity effect returned signal strength? 12
How does the device work with reflective surfaces? 12
Diffuse Reflective Surfaces 12
Specular Surfaces 12
How does liquid affect the signal? 13

Specifications

Physical

Specification	Measurement
Size (LxWxH)	20 × 48 × 40 mm (0.8 × 1.9 × 1.6 in.)
Weight	22 g (0.78 oz.)
Operating temperature	-20 to 60°C (-4 to 140°F)

Electrical

Specification	Measurement
Power	5 Vdc nominal 4.5 Vdc min., 5.5 Vdc max.
Current consumption	105 mA idle 135 mA continuous operation

Performance

Specification	Measurement
Range (70% reflective target)	40 m (131 ft)
Resolution	+/- 1 cm (0.4 in.)
Accuracy < 5 m	±2.5 cm (1 in.) typical*
Accuracy ≥ 5 m	±10 cm (3.9 in.) typical Mean ±1% of distance maximum Ripple ±1% of distance maximum
Update rate (70% Reflective Target)	270 Hz typical 650 Hz fast mode** >1000 Hz short range only
Repetition rate	~50 Hz default 500 Hz max

*Nonlinearity present below 1 m (39.4 in.)

**Reduced sensitivity

Interface

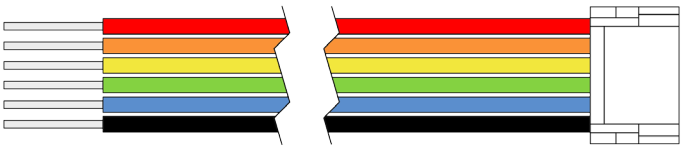
Specification	Measurement
User interface	I2C PWM External trigger
I2C interface	Fast-mode (400 kbit/s) Default 7-bit address 0x62 Internal register access & control
PWM interface	External trigger input PWM output proportional to distance at 10 µs/cm

Laser

Specification	Measurement
Wavelength	905 nm (nominal)
Total laser power (peak)	1.3 W
Mode of operation	Pulsed (256 pulse max. pulse train)
Pulse width	0.5 µs (50% duty Cycle)
Pulse train repetition frequency	10-20 KHz nominal
Energy per pulse	<280 nJ
Beam diameter at laser aperture	12 × 2 mm (0.47 × 0.08 in.)
Divergence	8 mRadian

Connections

Wiring Harness



Wire Color	Function
Red	5 Vdc (+)
Orange	Power enable (internal pull-up)
Yellow	Mode control
Green	I2C SCL
Blue	I2C SDA
Black	Ground (-)

There are two basic configurations for this device:

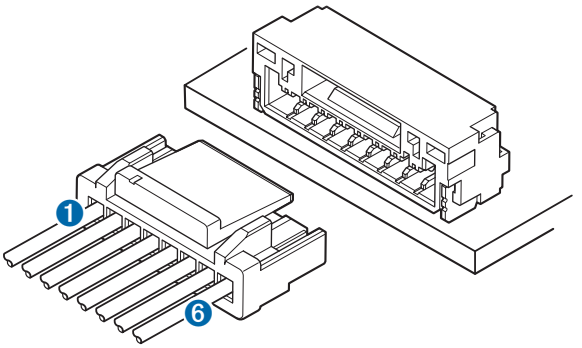
- **I2C (Inter-Integrated Circuit)**—a serial computer bus used to communicate between this device and a microcontroller, such as an Arduino board ("I2C Interface", page 4).
- **PWM (Pulse Width Modulation)**—a bi-directional signal transfer method that triggers acquisitions and returns distance measurements using the mode-control pin ("Mode Control Pin", page 4).

Connector

You can create your own wiring harness if needed for your project or application. The needed components are readily available from many suppliers.

Part	Description	Manufacturer	Part Number
Connector housing	6-position, rectangular housing, latch-lock connector receptacle with a 1.25 mm (0.049 in.) pitch.	JST	GHR-06V-S
Connector terminal	26-30 AWG crimp socket connector terminal (up to 6)	JST	SSHL-002T-P0.2
Wire	UL 1061 26 AWG stranded copper	N/A	N/A

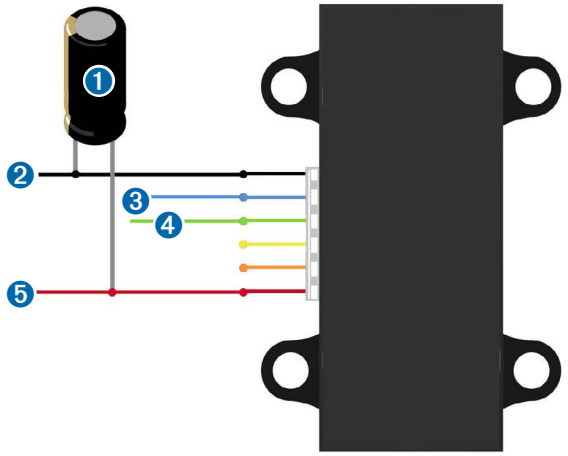
Connector Port Identification



Item	Pin	Function
1	1	5 Vdc (+)
	2	Power enable (internal pull-up)
	3	Mode control
	4	I2C SCL
	5	I2C SDA
6	6	Ground (-)

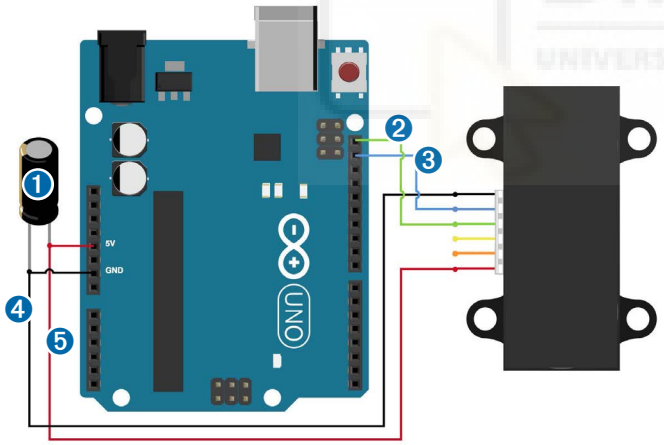
I2C Connection Diagrams

Standard I2C Wiring



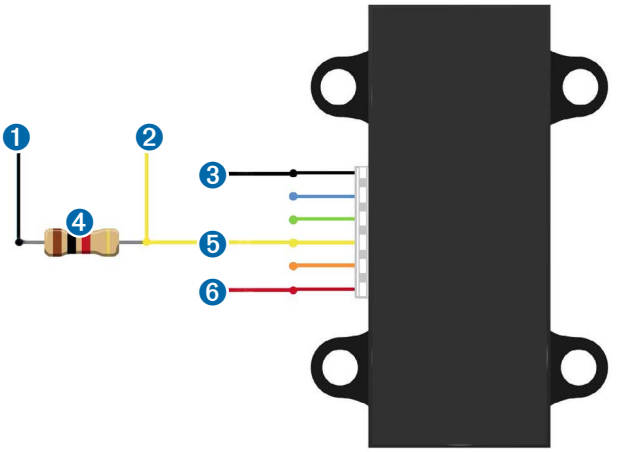
Item	Description	Notes
1	680µF electrolytic capacitor	You must observe the correct polarity when installing the capacitor.
2	Power ground (-) connection	Black wire
3	I2C SDA connection	Blue wire
4	I2C SCA connection	Green wire
5	5 Vdc power (+) connection	Red wire The sensor operates at 4.75 through 5.5 Vdc, with a max. of 6 Vdc.

Standard Arduino I2C Wiring



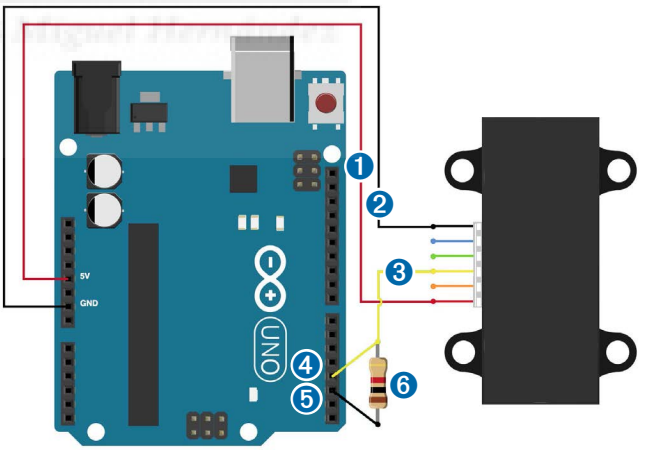
Item	Description	Notes
1	680µF electrolytic capacitor	You must observe the correct polarity when installing the capacitor.
2	I2C SCA connection	Green wire
3	I2C SDA connection	Blue wire
4	Power ground (-) connection	Black wire
5	5 Vdc power (+) connection	Red wire The sensor operates at 4.75 through 5.5 Vdc, with a max. of 6 Vdc.

PWM Wiring



Item	Description	Notes
1	Trigger pin on microcontroller	Connect the other side of the resistor to the trigger pin on your microcontroller.
2	Monitor pin on microcontroller	Connect one side of the resistor to the mode-control connection on the device, and to a monitoring pin on your microcontroller.
3	Power ground (-) connection	Black Wire
4	1kΩ resistor	
5	Mode-control connection	Yellow wire
6	5 Vdc power (+) connection	Red wire The sensor operates at 4.75 through 5.5 Vdc, with a max. of 6 Vdc.

PWM Arduino Wiring



Item	Description	Notes
1	5 Vdc power (+) connection	Red wire The sensor operates at 4.75 through 5.5 Vdc, with a max. of 6 Vdc.
2	Power ground (-) connection	Black Wire
3	Mode-control connection	Yellow wire
4	Monitor pin on microcontroller	Connect one side of the resistor to the mode-control connection on the device, and to a monitoring pin on your microcontroller.
5	Trigger pin on microcontroller	Connect the other side of the resistor to the trigger pin on your microcontroller.
6	1kΩ resistor	

Operational Information

Technology

This device measures distance by calculating the time delay between the transmission of a Near-Infrared laser signal and its reception after reflecting off of a target. This translates into distance using the known speed of light. Our unique signal processing approach transmits a coded signature and looks for that signature in the return, which allows for highly effective detection with eye-safe laser power levels. Proprietary signal processing techniques are used to achieve high sensitivity, speed, and accuracy in a small, low-power, and low-cost system

Theory of Operation

To take a measurement, this device first performs a receiver bias correction routine, correcting for changing ambient light levels and allowing maximum sensitivity.

Then the device sends a reference signal directly from the transmitter to the receiver. It stores the transmit signature, sets the time delay for “zero” distance, and recalculates this delay periodically after several measurements.

Next, the device initiates a measurement by performing a series of acquisitions. Each acquisition is a transmission of the main laser signal while recording the return signal at the receiver. If there is a signal match, the result is stored in memory as a correlation record. The next acquisition is summed with the previous result. When an object at a certain distance reflects the laser signal back to the device, these repeated acquisitions cause a peak to emerge, out of the noise, at the corresponding distance location in the correlation record.

The device integrates acquisitions until the signal peak in the correlation record reaches a maximum value. If the returned signal is not strong enough for this to occur, the device stops at a predetermined maximum acquisition count.

Signal strength is calculated from the magnitude of the signal record peak and a valid signal threshold is calculated from the noise floor. If the peak is above this threshold the measurement is considered valid and the device will calculate the distance, otherwise it will report 1 cm. When beginning the next measurement, the device clears the signal record and starts the sequence again.

Interface

Initialization

On power-up or reset, the device performs a self-test sequence and initializes all registers with default values. After roughly 22 ms distance measurements can be taken with the I2C interface or the Mode Control Pin.

Power Enable Pin

The enable pin uses an internal pullup resistor, and can be driven low to shut off power to the device.

I2C Interface

This device has a 2-wire, I2C-compatible serial interface (refer to I2C-Bus Specification, Version 2.1, January 2000, available from Philips Semiconductor). It can be connected to an I2C bus as a slave device, under the control of an I2C master device. It supports 400 kHz Fast Mode data transfer.

The I2C bus operates internally at 3.3 Vdc. An internal level shifter allows the bus to run at a maximum of 5 Vdc. Internal 3k ohm pullup resistors ensure this functionality and allow for a simple connection to the I2C host.

The device has a 7-bit slave address with a default value of 0x62. The effective 8-bit I2C address is 0xC4 write and 0xC5 read. The device will not respond to a general call. Support is not provided for 10-bit addressing.

Setting the most significant bit of the I2C address byte to one triggers automatic incrementing of the register address with successive reads or writes within an I2C block transfer. This is commonly used to read the two bytes of a 16-bit value within one transfer and is used in the following example.

The simplest method of obtaining measurement results from the I2C interface is as follows:

1. Write 0x04 to register 0x00.
2. Read register 0x01. Repeat until bit 0 (LSB) goes low.
3. Read two bytes from 0x8f (High byte 0x0f then low byte 0x10) to obtain the 16-bit measured distance in centimeters.

A list of all available control resisters is available on page 7.

For more information about the I2C protocol, see I2C Protocol Operation (page 7).

Mode Control Pin

The mode control pin provides a means to trigger acquisitions and return the measured distance via Pulse Width Modulation (PWM) without having to use the I2C interface.

The idle state of the mode control pin is high impedance (High-Z). Pulling the mode control pin low will trigger a single measurement, and the device will respond by driving the line high with a pulse width proportional to the measured distance at 10 µs/cm. A 1k ohm termination resistance is required to prevent bus contention.

The device drives the mode control pin high at 3.3 Vdc. Diode isolation allows the pin to tolerate a maximum of 5 Vdc.

As shown in the diagram PWM Arduino Wiring (page 3), a simple triggering method uses a 1k ohm resistor in series with a host output pin to pull the mode control pin low to initiate a measurement, and a host input pin connected directly to monitor the low-to-high output pulse width.

If the mode control pin is held low, the acquisition process will repeat indefinitely, producing a variable frequency output proportional to distance.

The mode control pin behavior can be modified with the ACQ_CONFIG_REG (0x04) I2C register as detailed in 0x04 (page 8).

Settings

The device can be configured with alternate parameters for the distance measurement algorithm. This can be used to customize performance by enabling configurations that allow choosing between speed, range and sensitivity. Other useful features are also detailed in this section. See the full register map (Control Register List (page 7) for additional settings not mentioned here.

Receiver Bias Correction

Address	Name	Description	Initial Value
0x00	ACQ_COMMAND	Device command	--

- Write 0x00: Reset device, all registers return to default values
- Write 0x03: Take distance measurement without receiver bias correction
- Write 0x04: Take distance measurement with receiver bias correction

Faster distance measurements can be performed by omitting the receiver bias correction routine. Measurement accuracy and sensitivity are adversely affected if conditions change (e.g. target distance, device temperature, and optical noise). To achieve good performance at high measurement rates receiver bias correction must be performed periodically. The recommended method is to do so at the beginning of every 100 sequential measurement commands.

Maximum Acquisition Count

Address	Name	Description	Initial Value
0x02	SIG_COUNT_VAL	Maximum acquisition count	0x80

The maximum acquisition count limits the number of times the device will integrate acquisitions to find a correlation record peak (from a returned signal), which occurs at long range or with low target reflectivity. This controls the minimum measurement rate and maximum range. The unit-less relationship is roughly as follows: rate = 1/n and range = n^(1/4), where n is the number of acquisitions.

Measurement Quick Termination Detection

Address	Name	Description	Initial Value
0x04	ACQ_CONFIG_REG	Acquisition mode control	0x08

You can enable quick-termination detection by clearing bit 3 in this register. The device will terminate a distance measurement early if it anticipates that the signal peak in the correlation record will reach maximum value. This allows for faster and slightly less accurate operation at strong signal strengths without sacrificing long range performance.

Detection Sensitivity

Address	Name	Description	Initial Value
0x1c	THRESHOLD_ BYPASS	Peak detection threshold bypass	0x00

The default valid measurement detection algorithm is based on the peak value, signal strength, and noise in the correlation record. This can be overridden to become a simple threshold criterion by setting a non-zero value. Recommended non-default values are 0x20 for higher sensitivity with more frequent erroneous measurements, and 0x60 for reduced sensitivity and fewer erroneous measurements.

Burst Measurements and Free Running Mode

Address	Name	Description	Initial Value
0x04	ACQ_CONFIG_REG	Acquisition mode control	0x08
0x11	OUTER_LOOP_ COUNT	Burst measurement count control	0x00
0x45	MEASURE_DELAY	Delay between automatic measurements	0x14

The device can be configured to take multiple measurements for each measurement command or repeat indefinitely for free running mode.

OUTER_LOOP_COUNT (0x11) controls the number of times the device will retrigger itself. Values 0x00 or 0x01 result in the default one measurement per command. Values 0x02 to 0xfe directly set the repetition count. Value 0xff will enable free running mode after the host device sends an initial measurement command.

The default delay between automatic measurements corresponds to a 10 Hz repetition rate. Set bit 5 in ACQ_CONFIG_REG (0x04) to use the delay value in MEASURE_DELAY (0x45) instead. A delay value of 0x14 roughly corresponds to 100Hz.

The delay is timed from the completion of each measurement. The means that measurement duration, which varies with returned signal strength, will affect the repetition rate. At low repetition rates (high delay) this effect is small, but for lower delay values it is recommended to limit the maximum acquisition count if consistent frequency is desired.

Velocity

Address	Name	Description	Initial Value
0x09	VELOCITY	Velocity measurement output	--

The velocity measurement is the difference between the current measurement and the previous one, resulting in a signed (2's complement) 8-bit number in cm. Positive velocity is away from the device. This can be combined with free running mode for a constant measurement frequency. The default free running frequency of 10 Hz therefore results in a velocity measurement in .1 m/s.

Configurable I2C Address

Address	Name	Description	Initial Value
0x16	UNIT_ID_HIGH	Serial number high byte	Unique
0x17	UNIT_ID_LOW	Serial number low byte	Unique
0x18	I2C_ID_HIGH	Write serial number high byte for I2C address unlock	--
0x19	I2C_ID_LOW	Write serial number low byte for I2C address unlock	--
0x1a	I2C_SEC_ADDR	Write new I2C address after unlock	--
0x1e	I2C_CONFIG	Default address response control	0x00

The I2C address can be changed from its default value. Available addresses are 7-bit values with a '0' in the least significant bit (even hexadecimal numbers).

To change the I2C address, the unique serial number of the unit must be read then written back to the device before setting the new address. The process is as follows:

1. Read the two byte serial number from 0x96 (High byte 0x16 and low byte 0x17).
2. Write the serial number high byte to 0x18.
3. Write the serial number low byte to 0x19.
4. Write the desired new I2C address to 0x1a.
5. Write 0x08 to 0x1e to disable the default address.

This can be used to run multiple devices on a single bus, by enabling one, changing its address, then enabling the next device and repeating the process.

The I2C address will be restored to default after a power cycle.

Power Control

Address	Name	Description	Initial Value
0x65	POWER_CONTROL	Power state control	0x80

NOTE: The most effective way to control power usage is to utilize the enable pin to deactivate the device when not in use.

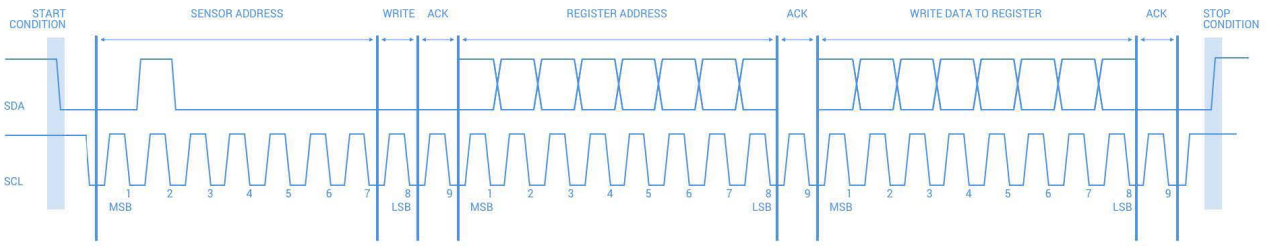
Another option is to set bit 0 in this register which disables the receiver circuit, saving roughly 40mA. After being re-enabled, the receiver circuit stabilizes by the time a measurement can be performed. Setting bit 2 puts the device in sleep mode until the next I2C transaction, saving 20mA. **Since the wake-up time is only around 2 m/s shorter than the full power-on time, and both will reset all registers, it is recommended to use the enable pin instead.**

I2C Protocol Information

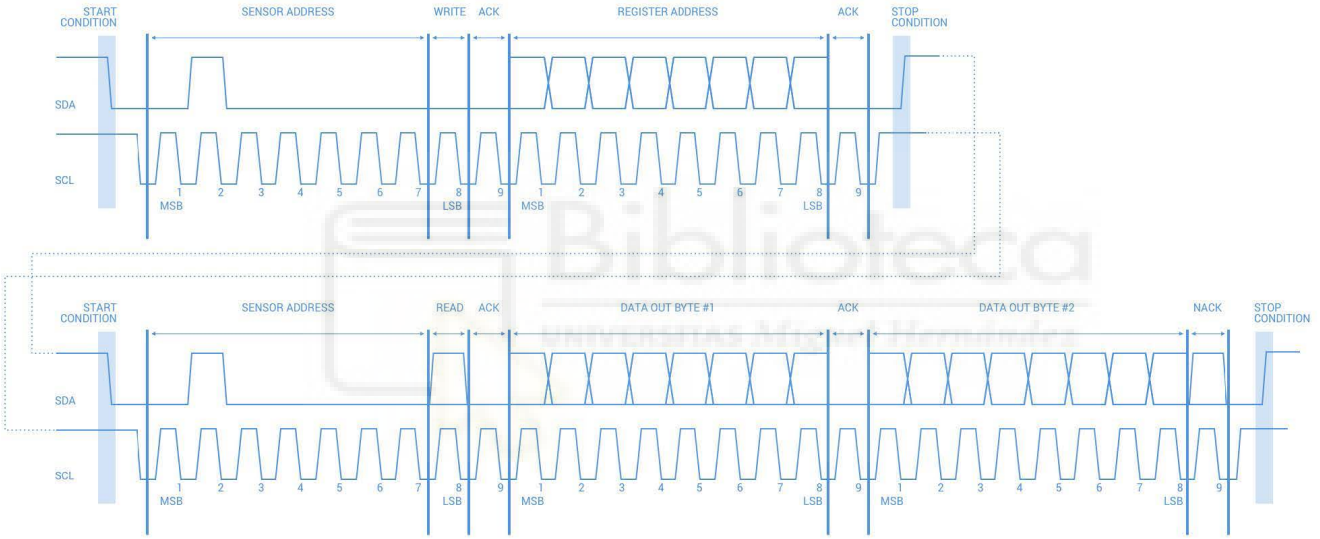
This device has a 2-wire, I2C-compatible serial interface (refer to I2C-Bus Specification, Version 2.1, January 2000, available from Philips Semiconductor). It can be connected to an I2C bus as a slave device, under the control of an I2C master device. It supports standard 400 kHz data transfer mode. Support is not provided for 10-bit addressing.

The Sensor module has a 7-bit slave address with a default value of 0x62 in hexadecimal notation. The effective 8 bit I2C address is: 0xC4 write, 0xC5 read. The device will not presently respond to a general call.

Write



Read



Notes:

- This device does not work with repeated START conditions. It must first receive a STOP condition before a new START condition.
- The ACK and NACK items are responses from the master device to the slave device.
- The last NACK in the read is technically optional, but the formal I2C protocol states that the master shall not acknowledge the last byte.

I2C Protocol Operation

The I2C serial bus protocol operates as follows:

1. The master initiates data transfer by establishing a start condition, which is when a high-to-low transition on the SDA line occurs while SCL is high. The following byte is the address byte, which consists of the 7-bit slave address followed by a read/write bit with a zero state indicating a write request. A write operation is used as the initial stage of both read and write transfers. If the slave address corresponds to the module's address the unit responds by pulling SDA low during the ninth clock pulse (this is termed the acknowledge bit). At this stage, all other devices on the bus remain idle while the selected device waits for data to be written to or read from its shift register.
2. Data is transmitted over the serial bus in sequences of nine clock pulses (eight data bits followed by an acknowledge bit). The transitions on the SDA line must occur during the low period of SCL and remain stable during the high period of SCL.
3. An 8 bit data byte following the address loads the I2C control register with the address of the first control register to be read along with flags indicating if auto increment of the addressed control register is desired with successive reads or writes; and if access to the internal micro or external correlation processor register space is requested. Bit locations 5:0 contain the control register address while bit 7 enables the automatic incrementing of control register with successive data blocks. Bit position 6 selects correlation memory external to the microcontroller if set. (Presently an advanced feature)
4. If a read operation is requested, a stop bit is issued by the master at the completion of the first data frame followed by the initiation of a new start condition, slave address with the read bit set (one state). The new address byte is followed by the reading of one or more data bytes succession. After the slave has acknowledged receipt of a valid address, data read operations proceed by the master releasing the I2C data line SDA with continuing clocking of SCL. At the completion of the receipt of a data byte, the master must strobe the acknowledge bit before continuing the read cycle.
5. For a write operation to proceed, Step 3 is followed by one or more 8 bit data blocks with acknowledges provided by the slave at the completion of each successful transfer. At the completion of the transfer cycle a stop condition is issued by the master terminating operation.

Register Definitions

Control Register List

Address	R/W	Name	Description	Intial Value	Details
0x00	W	ACQ_COMMAND	Device command	--	page 8
0x01	R	STATUS	System status	--	page 8
0x02	R/W	SIG_COUNT_VAL	Maximum acquisition count	0x80	page 8
0x04	R/W	ACQ_CONFIG_REG	Acquisition mode control	0x08	page 8
0x09	R	VELOCITY	Velocity measurement output	--	page 8
0x0c	R	PEAK_CORR	Peak value in correlation record	--	page 8
0x0d	R	NOISE_PEAK	Correlation record noise floor	--	page 8
0x0e	R	SIGNAL_STRENGTH	Received signal strength	--	page 9
0x0f	R	FULL_DELAY_HIGH	Distance measurement high byte	--	page 9
0x10	R	FULL_DELAY_LOW	Distance measurement low byte	--	page 9
0x11	R/W	OUTER_LOOP_COUNT	Burst measurement count control	0x01	page 9
0x12	R/W	REF_COUNT_VAL	Reference acquisition count	0x05	page 9
0x14	R	LAST_DELAY_HIGH	Previous distance measurement high byte	--	page 9
0x15	R	LAST_DELAY_LOW	Previous distance measurement low byte	--	page 9
0x16	R	UNIT_ID_HIGH	Serial number high byte	Unique	page 9
0x17	R	UNIT_ID_LOW	Serial number low byte	Unique	page 9
0x18	W	I2C_ID_HIGH	Write serial number high byte for I2C address unlock	--	page 9
0x19	W	I2C_ID_LOW	Write serial number low byte for I2C address unlock	--	page 9
0x1a	R/W	I2C_SEC_ADDR	Write new I2C address after unlock	--	page 9
0x1c	R/W	THRESHOLD_BYPASS	Peak detection threshold bypass	0x00	page 9
0x1e	R/W	I2C_CONFIG	Default address response control	0x00	page 9
0x40	R/W	COMMAND	State command	--	page 10
0x45	R/W	MEASURE_DELAY	Delay between automatic measurements	0x14	page 10
0x4c	R	PEAK_BCK	Second largest peak value in correlation record	--	page 10
0x52	R	CORR_DATA	Correlation record data low byte	--	page 10
0x53	R	CORR_DATA_SIGN	Correlation record data high byte	--	page 10
0x5d	R/W	ACQ_SETTINGS	Correlation record memory bank select	--	page 10
0x65	R/W	POWER_CONTROL	Power state control	0x80	page 10

Detailed Control Register Definitions

NOTE: Unless otherwise noted, all registers contain one byte and are read and write.

0x00

R/W	Name	Description	Initial Value
W	ACQ_COMMAND	Device command	--

Bit	Function
7:0	Write 0x00: Reset FPGA, all registers return to default values Write 0x03: Take distance measurement without receiver bias correction Write 0x04: Take distance measurement with receiver bias correction

0x01

R/W	Name	Description	Initial Value
R	STATUS	System status	--

Bit	Function
6	Process Error Flag 0: No error detected 1: System error detected during measurement
5	Health Flag 0: Error detected 1: Reference and receiver bias are operational
4	Secondary Return Flag 0: No secondary return detected 1: Secondary return detected in correlation record
3	Invalid Signal Flag 0: Peak detected 1: Peak not detected in correlation record, measurement is invalid
2	Signal Overflow Flag 0: Signal data has not overflowed 1: Signal data in correlation record has reached the maximum value before overflow. This occurs with a strong received signal strength
1	Reference Overflow Flag 0: Reference data has not overflowed 1: Reference data in correlation record has reached the maximum value before overflow. This occurs periodically
0	Busy Flag 0: Device is ready for new command 1: Device is busy taking a measurement

0x02

R/W	Name	Description	Initial Value
R/W	SIG_COUNT_VAL	Maximum acquisition count	0x80

Bit	Function
7:0	Maximum number of acquisitions during measurement

0x04

R/W	Name	Description	Initial Value
R/W	ACQ_CONFIG_REG	Acquisition mode control	0x08

Bit	Function
6	0: Enable reference process during measurement 1: Disable reference process during measurement
5	0: Use default delay for burst and free running mode 1: Use delay from MEASURE_DELAY (0x45) for burst and free running mode
4	0: Enable reference filter, averages 8 reference measurements for increased consistency 1: Disable reference filter
3	0: Enable measurement quick termination. Device will terminate distance measurement early if it anticipates that the signal peak in the correlation record will reach maximum value. 1: Disable measurement quick termination.
2	0: Use default reference acquisition count of 5. 1: Use reference acquisition count from REF_COUNT_VAL (0x12).
1:0	Mode Select Pin Function Control 00: Default PWM mode. Pull pin low to trigger measurement, device will respond with an active high output with a duration of 10us/cm. 01: Status output mode. Device will drive pin active high while busy. Can be used to interrupt host device. 10: Fixed delay PWM mode. Pulling pin low will not trigger a measurement. 11: Oscillator output mode. Nominal 31.25 kHz output. The accuracy of the silicon oscillator in the device is generally within 1% of nominal. This affects distance measurements proportionally and can be measured to apply a compensation factor.

0x09

R/W	Name	Description	Initial Value
R	VELOCITY	Velocity measurement output	--

Bit	Function
7:0	Velocity measurement output. The difference between the current measurement and the previous one, signed (2's complement) value in centimeters.

0x0c

R/W	Name	Description	Initial Value
R	PEAK_CORR	Peak value in correlation record	--

Bit	Function
7:0	The value of the highest peak in the correlation record.

0x0d

R/W	Name	Description	Initial Value
R	NOISE_PEAK	Correlation record noise floor	--

Bit	Function
7:0	A measure of the noise in the correlation record. Will be slightly above the third highest peak.

0x0e

R/W	Name	Description	Initial Value
R	SIGNAL_STRENGTH	Received signal strength	--

Bit	Function
7:0	Received signal strength calculated from the value of the highest peak in the correlation record and how many acquisitions were performed.

0x0f

R/W	Name	Description	Initial Value
R	FULL_DELAY_HIGH	Distance measurement high byte	--

Bit	Function
7:0	Distance measurement result in centimeters, high byte.

0x10

R/W	Name	Description	Initial Value
R	FULL_DELAY_LOW	Distance measurement low byte	--

Bit	Function
7:0	Distance measurement result in centimeters, low byte.

0x11

R/W	Name	Description	Initial Value
R/W	OUTER_LOOP_COUNT	Burst measurement count control	0x01

Bit	Function
7:0	0x00-0x01: One measurement per distance measurement command. 0x02-0xfe: Repetition count per distance measurement command. 0xff: Indefinite repetitions after initial distance measurement command. See ACQ_CONFIG_REG (0x04) and MEASURE_DELAY (0x45) for non-default automatic repetition delays.

0x12

R/W	Name	Description	Initial Value
R/W	REF_COUNT_VAL	Reference acquisition count	0x05

Bit	Function
7:0	Non-default number of reference acquisitions during measurement. ACQ_CONFIG_REG (0x04) bit 2 must be set.

0x14

R/W	Name	Description	Initial Value
R	LAST_DELAY_HIGH	Previous distance measurement high byte	--

Bit	Function
7:0	Previous distance measurement result in centimeters, high byte.

0x15

R/W	Name	Description	Initial Value
R	LAST_DELAY_LOW	Previous distance measurement low byte	--

Bit	Function
7:0	Previous distance measurement result in centimeters, low byte.

0x16

R/W	Name	Description	Initial Value
R	UNIT_ID_HIGH	Serial number high byte	Unique

Bit	Function
7:0	Unique serial number of device, high byte.

0x17

R/W	Name	Description	Initial Value
R	UNIT_ID_LOW	Serial number low byte	Unique

Bit	Function
7:0	Unique serial number of device, high byte.

0x18

R/W	Name	Description	Initial Value
W	I2C_ID_HIGH	Write serial number high byte for I2C address unlock	--

Bit	Function
7:0	Write the value in UNIT_ID_HIGH (0x16) here as part of enabling a non-default I2C address. See I2C_ID_LOW (0x19) and I2C_SEC_ADDR (0x1a).

0x19

R/W	Name	Description	Initial Value
W	I2C_ID_LOW	Write serial number low byte for I2C address unlock	--

Bit	Function
7:0	Write the value in UNIT_ID_LOW (0x17) here as part of enabling a non-default I2C address. See I2C_ID_HIGH (0x18) and I2C_SEC_ADDR (0x1a).

0x1a

R/W	Name	Description	Initial Value
R/W	I2C_SEC_ADDR	Write new I2C address after unlock	--

Bit	Function
7:0	Non-default I2C address. Available addresses are 7-bit values with a '0' in the least significant bit (even hexadecimal numbers). I2C_ID_HIGH (0x18) and I2C_ID_LOW (0x19) must have the correct value for the device to respond to the non-default I2C address.

0x1c

R/W	Name	Description	Initial Value
R/W	THRESHOLD_BYPASS	Peak detection threshold bypass	0x00

Bit	Function
7:0	0x00: Use default valid measurement detection algorithm based on the peak value, signal strength, and noise in the correlation record. 0x01-0xff: Set simple threshold for valid measurement detection. Values 0x20-0x60 generally perform well.

0x1e

R/W	Name	Description	Initial Value
R/W	I2C_CONFIG	Default address response control	0x00

Bit	Function

3	0: Device will respond to I2C address 0x62. Device will also respond to non-default address if configured successfully. See I2C_ID_HIGH (0x18), I2C_ID_LOW (0x19), and I2C_SEC_ADDR (0x1a). 1: Device will only respond to non-default I2C address. It is recommended to configure the non-default address first, then use the non-default address to write to this register, ensuring success.

0x40

R/W	Name	Description	Initial Value
R/W	COMMAND	State command	--

Bit	Function
2:0	000: Test mode disable, resume normal operation 111: Test mode enable, allows download of correlation record Select correlation memory bank in ACQ_SETTINGS (0x5d) prior to enabling test mode. Once test mode is enabled, read CORR_DATA (0x52) and CORR_DATA_SIGN (0x53) in one transaction (read from 0xd2). The memory index is incremented automatically and successive reads produce sequential data.

0x45

R/W	Name	Description	Initial Value
R/W	MEASURE_DELAY	Delay between automatic measurements	0x14

Bit	Function
7:0	Non-default delay after completion of measurement before automatic retrigger, in burst and continuous modes. ACQ_CONFIG_REG (0x04) bit 5 must be set. Value 0xc8 corresponds to 10 Hz repetition rate and 0x14 to roughly 100 Hz.

0x4c

R/W	Name	Description	Initial Value
R	PEAK_BCK	Second largest peak value in correlation record	--

Bit	Function
7:0	The value of the second highest peak in the correlation record.

0x52

R/W	Name	Description	Initial Value
R	CORR_DATA	Correlation record data low byte	--

Bit	Function
7:0	Correlation record data low byte. See CORR_DATA_SIGN (0x53), ACQ_SETTINGS (0x5d), and COMMAND (0x40).

0x53

R/W	Name	Description	Initial Value
R	CORR_DATA_SIGN	Correlation record data high byte	--

Bit	Function
7:0	Correlation record data high byte. Correlation record data is a 2's complement 9-bit value, and must be sign extended to be formatted as a 16-bit 2's complement value. Thus when repacking the two bytes obtained for the I2C transaction, set the high byte to 0xff if the LSB of the high byte is one.

0x5d

R/W	Name	Description	Initial Value
R/W	ACQ_SETTINGS	Correlation record memory bank select	--

Bit	Function
7:6	11: Access correlation memory bank. Write prior to test mode enable, see COMMAND (0x40).

0x65

R/W	Name	Description	Initial Value
R/W	POWER_CONTROL	Power state control	0x80

Bit	Function
2	1: Device Sleep, wakes upon I2C transaction. Registers are reinitialized, wakeup time similar to full reset using enable pin. 0: Device awake
0	1: Disable receiver circuit 0: Enable receiver circuit. Receiver circuit stabilizes by the time a measurement can be performed.

Frequently Asked Questions

Must the device run on 5 Vdc? Can it run on 3.3 Vdc instead?

The device requires 5 Vdc to run properly, so this specification is recommended and supported.

What is the spread of the laser beam?

At very close distances (less than 1 m) the beam diameter is about the size of the aperture (lens). For distances greater than 1 m, you can estimate the beam diameter using this equation:

Distance/100 = beam diameter at that distance (in whatever units you measured the distance).

The actual spread is ~8 milli-radians or ~1/2 degree.

How do distance, target size, aspect, and reflectivity effect returned signal strength?

The device transmits a focused infrared beam that reflects off of a target, and a portion of that reflected signal returns to the receiver. The distance is calculated by taking the difference between the moment of signal transmission to the moment of signal reception. Successfully receiving a reflected signal is heavily influenced by several factors. These factors include:

- Target Distance

  The relationship of distance (D) to returned signal strength is an inverse square. So, with increase in distance, returned signal strength decreases by 1/D^2 or the square root of the distance.

- Target Size

  The relationship of a target's Cross Section (C) to returned signal strength is an inverse power of four. The device transmits a focused near-infrared laser beam that spreads at a rate of approximately 0.5º as distance increases. Up to 1 m it is approximately the size of the lens. Beyond 1 m, the approximate beam spread in degrees can be estimated by dividing the distance by 100, or ~8 milliradians. When the beam overfills (is larger than) the target, the signal returned decreases by 1/C^4 or the fourth root of the target's cross section.

- Aspect

  The aspect of the target, or its orientation to the sensor, affects the observable cross section and, therefore, the amount of returned signal decreases as the aspect of the target varies from the normal.

- Reflectivity

  Reflectivity characteristics of the target's surface also affect the amount of returned signal. In this case, we concern ourselves with reflectivity of near infrared wavelengths ("How does the device work with reflective surfaces?", page 12).

In summary, a small target can be very difficult to detect if it is distant, poorly reflective, and its aspect is away from the normal. In such cases, the returned signal strength may be improved by attaching infrared reflectors to the target, increasing the size of the target, modifying its aspect, or reducing distance from the sensor.

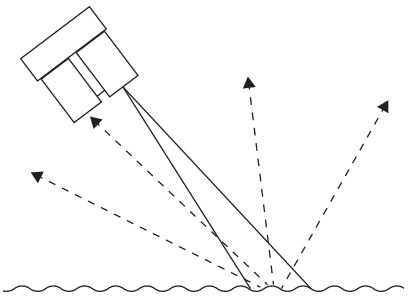
How does the device work with reflective surfaces?

Reflective characteristics of an object's surface can be divided into three categories (in the real world, a combination of characteristics is typically present):

- Diffuse Reflective
- Specular
- Retro-reflective

Diffuse Reflective Surfaces

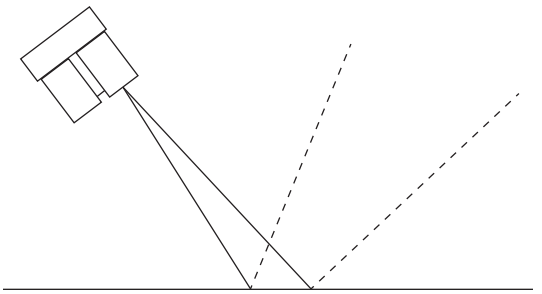
Purely diffuse surfaces are found on materials that have a textured quality that causes reflected energy to disperse uniformly. This tendency results in a relatively predictable percentage of the dispersed laser energy finding its way back to the device. As a result, these materials tend to read very well.



Materials that fall into this category are paper, matte walls, and granite. It is important to note that materials that fit into this category due to observed reflection at visible light wavelengths may exhibit unexpected results in other wavelengths. The near infrared range used by the device may detect them as nearly identical. For example, a black sheet of paper may reflect a nearly identical percentage of the infrared signal back to the receiver as a white sheet.

Specular Surfaces

Specular surfaces, are found on materials that have a smooth quality that reflect energy instead of dispersing it. It is difficult or impossible for the device to recognize the distance of many specular surfaces. Reflections off of specular surfaces tend to reflect with little dispersion which causes the reflected beam to remain small and, if not reflected directly back to the receiver, to miss the receiver altogether. The device may fail to detect a specular object in front of it unless viewed from the normal.



Examples of specular surfaces are mirrors and glass viewed off-axis.

How does liquid affect the signal?

There are a few considerations to take into account if your application requires measuring distances to, or within, liquid:

- Reflectivity and other characteristics of the liquid itself
- Reflectivity characteristics of particles suspended in the liquid
- Turbidity
- Refractive characteristics of the liquid

Reflectivity of the liquid is important when measuring distance to the surface of a liquid or if measuring through liquid to the bottom of a container("How does the device work with reflective surfaces?", page 12).

It is important to note that measuring distance with the device depends on reflected energy from the transmitted signal being detected by the receiver in the sensor. For that reason, the surface condition of the liquid may play an important role in the overall reflectivity and detectability of the liquid. In the case of a flat, highly reflective liquid surface, the laser's reflected energy may not disperse adequately to allow detection unless viewed from the normal. By contrast, small surface ripples may create enough dispersion of the reflected energy to allow detection of the liquid without the need to position the sensor so that the transmitted beam strikes the liquid's surface from the normal.

Reflectivity of suspended particles is a characteristic that may help or hinder depending on the application.

Turbidity, or the clarity of a liquid created by the presence or absence of suspended particles, can similarly help or hinder measurement efforts. If the application requires detecting the surface of the liquid, then suspended particles may help by reflecting more of the transmitted beam back to the receiver, increasing detectability and permitting measurements to be taken.

It is important to note that, attempting to measure through suspended particles in a liquid will only be successful if the transmitted beam is allowed to reflect off of the desired target without first being absorbed or reflected by the suspended particles.

When the near infrared energy transmitted by the device transitions from the atmosphere to a liquid, the energy may be bent, or refracted, and absorbed in addition to being dispersed. The degree to which the transmitted beam is refracted and absorbed is defined by its refraction index. That being said, the most important criteria impacting successful measurement through a liquid is the amount of dispersion of the transmitted beam and whether any of the dispersed beam makes its way back to the receiver on the device.

Remember that electromagnetic energy travels slower through a liquid and may affect accuracy of the final measurement output.

For the latest free software updates (excluding map data) throughout the life of your Garmin products, visit the Garmin Web site at www.garmin.com.

GARMIN®

Garmin International, Inc.
1200 East 151st Street, Olathe, Kansas 66062, USA

Garmin (Europe) Ltd.
Liberty House, Hounsdown Business Park, Southampton, Hampshire, SO40 9LR UK

Garmin Corporation
No. 68, Zhangshu 2nd Road, Xizhi Dist., New Taipei City, 221, Taiwan (R.O.C.)

www.garmin.com

September 2016 190-02088-00_0A Printed in Taiwan

GARMIN

LIDAR-Lite v3HP Operation Manual and Technical Specifications

Laser Safety

⚠ WARNING

This device requires no regular maintenance. In the event that the device becomes damaged or is inoperable, repair or service must be handled by authorized, factory-trained technicians only. Attempting to repair or service the unit on your own can result in direct exposure to laser radiation and the risk of permanent eye damage. For repair or service, contact your dealer or Garmin® for more information. This device has a protective housing which, when in place, prevents human access to laser radiation in excess of the accessible emission limit (AEL) for Class 1 laser products. This device should not be modified or operated without its housing or optics. Operating this device without a housing and optics, or operating this device with a modified housing or optics that expose the laser source, may result in direct exposure to laser radiation and the risk of permanent eye damage. Removal or modification of the diffuser in front of the laser optic may result in the risk of permanent eye damage.

⚠ CAUTION

This device emits laser radiation. Use of controls or adjustments or performance of procedures other than those specified herein may result in hazardous radiation exposure.

This laser product is designated Class 1 during all procedures of operation. When the ranging feature of the device is activated, a laser emitter of a ranging module may emit laser radiation and the device should not be aimed toward anyone. Avoid looking toward the laser emitter or into the laser radiation (beam) when operating the device. It is advisable to turn off the ranging module when it is not in use. This device must be used only according to the directions and procedures described in this documentation.

Do not leave this device within the reach of children.

NOTICE

CLASS 1 LASER PRODUCT
Classified EN/IEC 60825-1 2014

This product is in conformity with performance standards for laser products under 21 CFR 1040, except with respect to those characteristics authorized by Variance Number FDA-2016-V-2943 effective September 27, 2016.

Table of Contents

LIDAR-Lite v3HP Operation Manual and Technical Specifications 1
Laser Safety 1

Specifications 2
Physical 2
Water Resistance 2
Electrical 2
Performance 2
Interface 2
Laser 2

Connections 2
Wiring Harness 2
I2C Connection Diagrams 2
- Standard I2C Wiring 2
- Standard Arduino I2C Wiring 3
- PWM Wiring 3
- PWM Arduino Wiring 3

Operational Information 4
Technology 4
Theory of Operation 4
Interface 4
- Initialization 4
- Power Enable Pin 4
- I2C Interface 4
- Mode Control Pin 4
- Settings 4

I2C Protocol Information 6
I2C Protocol Operation 7
- Read Operation 7
- Write Operation 7

Register Definitions 7
- Control Register List 7
- Detailed Control Register Definitions 8

Frequently Asked Questions 10
How do I use the device for fast-scanning applications? 10
Does the device operate only on 5 Vdc? 10
What is the spread of the laser beam? 10
How do distance, target size, aspect, and reflectivity affect returned signal strength? 10
How does the device work with reflective surfaces? 11
- Diffuse Reflective Surfaces 11
- Specular Surfaces 11

How does liquid affect the signal? 11

Specifications

Physical

Specification	Measurement
Size (LxWxH)	20 × 48 × 40 mm (0.8 × 1.9 × 1.6 in.)
Weight	22 g (0.78 oz.)
Operating temperature	-20 to 60°C (-4 to 140°F)

Water Resistance

Body of this device is rated IPX7, and can wthstand incidental exposure to water of up to 1 meter for up to 30 minutes.

IMPORTANT: The bare wire portion of the wiring harness is not water resistant, and can act as a path for water to enter the device. All bare-wire connections must either be made in a water-tight location or properly sealed.

Water may enter under the transmitting lens. This could affect performance, but will not affect IPX7 water resistance.

Electrical

Specification	Measurement
Power	5 Vdc nominal 4.5 Vdc min., 5.5 Vdc max.
Current consumption	65 mA idle 85 mA during an acquisition

Performance

Specification	Measurement
Range (70% reflective target)	40 m (131 ft)
Resolution	+/- 1 cm (0.4 in.)
Accuracy < 2 m	±5 cm (2 in.) typical*
Accuracy ≥ 2 m	±2.5 cm (1 in.) typical Mean ±1% of distance maximum Ripple ±1% of distance maximum
Update rate (70% Reflective Target)	Greater than 1 kHz typical Reduced sensitivity at high update rates

*Nonlinearity present below 1 m (39.4 in.)

Interface

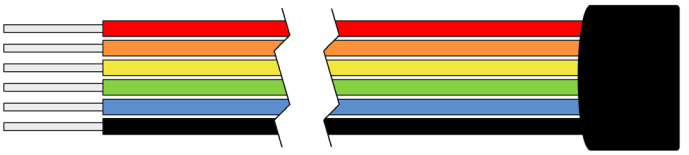
Specification	Measurement
User interface	I2C PWM External trigger
I2C interface	Fast-mode (400 kbit/s) Default 7-bit address 0x62 Internal register access & control
PWM interface	External trigger input PWM output proportional to distance at 10 µs/cm

Laser

Specification	Measurement
Wavelength	905 nm (nominal)
Total laser power (peak)	1.3 W
Mode of operation	Pulsed (256 pulse max. pulse train)
Pulse width	0.5 µs (50% duty cycle)
Pulse train repetition frequency	10-20 kHz nominal
Energy per pulse	<280 nJ
Beam diameter at laser aperture	12 × 2 mm (0.47 × 0.08 in.)
Divergence	8 mRad

Connections

Wiring Harness



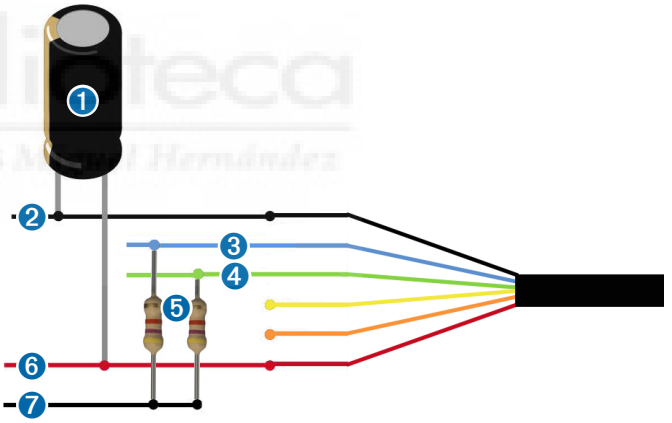
Wire Color	Function
Red	5 Vdc (+)
Orange	Power enable (internal pull-up)
Yellow	Mode control
Green	I2C SCL
Blue	I2C SDA
Black	Ground (-)

There are two basic configurations for this device:

- **I2C (Inter-Integrated Circuit)**—a serial computer bus used to communicate between this device and a microcontroller, such as an Arduino board (I2C Interface, page 4).
- **PWM (Pulse Width Modulation)**—a bi-directional signal transfer method that triggers acquisitions and returns distance measurements using the mode-control pin (Mode Control Pin, page 4).

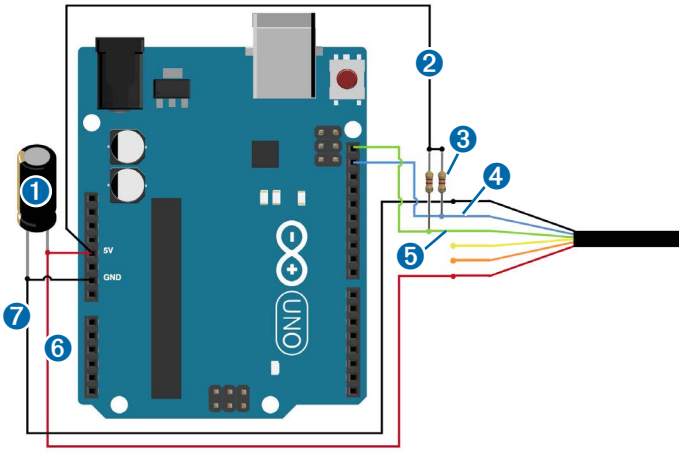
I2C Connection Diagrams

Standard I2C Wiring



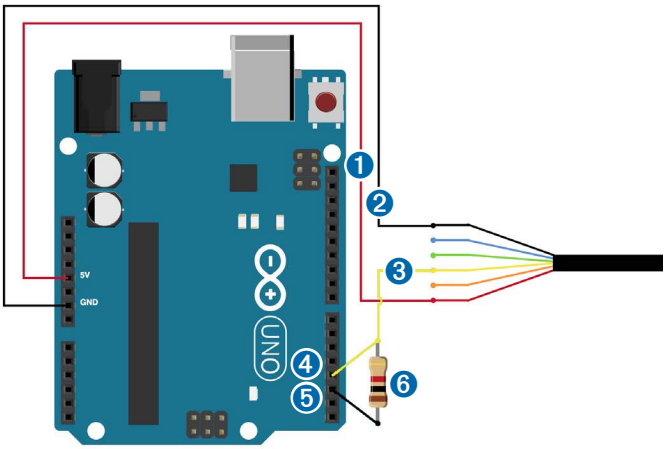
Item	Description	Notes
1	680µF electrolytic capacitor	You must observe the correct polarity when installing the capacitor.
2	Power ground (-) connection	Black wire
3	I2C SDA connection	Blue wire
4	I2C SCL connection	Green wire
5	4.7kΩ pull-up resistor (not required in all applications)	In installations with long cable extensions or with multiple devices on the I2C bus, you must install a 1kΩ to 10kΩ pull-up resistor on each I2C wire to account for cable capacitance. It is recommended to start with 4.7kΩ resistors and adjust if necessary.
6	5 Vdc power (+) connection	Red wire The sensor operates at 4.75 through 5.5 Vdc, with a max. of 6 Vdc.
7	Logic rail connection	The pull-up resistors connected to both I2C wires must connect to the logic rail on your microcontroller board.

Standard Arduino I2C Wiring



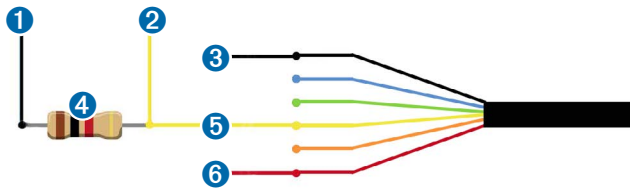
Item	Description	Notes
1	680µF electrolytic capacitor	You must observe the correct polarity when installing the capacitor.
2	Pull-up resistor connection (not required in all applications)	In installations with long cable extensions or with multiple devices on the I2C bus, you must connect the pull-up resistors on the SDA and SCL wires to the logic rail on your microcontroller board. On an Arduino board, this is the 5v pin.
3	4.7kΩ pull-up resistor (not required in all applications)	In installations with long cable extensions or with multiple devices on the I2C bus, you must install a 1kΩ to 10kΩ pull-up resistor on each I2C wire to account for cable capacitance. It is recommended to start with 4.7kΩ resistors and adjust if necessary.
4	I2C SDA connection	Blue wire
5	I2C SCL connection	Green wire
6	5 Vdc power (+) connection	Red wire The sensor operates at 4.75 through 5.5 Vdc, with a max. of 6 Vdc.
7	Power ground (-) connection	Black wire

PWM Arduino Wiring



Item	Description	Notes
1	5 Vdc power (+) connection	Red wire The sensor operates at 4.75 through 5.5 Vdc, with a max. of 6 Vdc.
2	Power ground (-) connection	Black Wire
3	Mode-control connection	Yellow wire
4	Monitor pin on microcontroller	Connect one side of the resistor to the mode-control connection on the device, and to a monitoring pin on your microcontroller board.
5	Trigger pin on microcontroller	Connect the other side of the resistor to the trigger pin on your microcontroller board.
6	1kΩ resistor	

PWM Wiring



Item	Description	Notes
1	Trigger pin on microcontroller	Connect the other side of the resistor to the trigger pin on your microcontroller.
2	Monitor pin on microcontroller	Connect one side of the resistor to the mode-control connection on the device, and to a monitoring pin on your microcontroller.
3	Power ground (-) connection	Black Wire
4	1kΩ resistor	
5	Mode-control connection	Yellow wire
6	5 Vdc power (+) connection	Red wire The sensor operates at 4.75 through 5.5 Vdc, with a max. of 6 Vdc.

Operational Information

Technology

This device measures distance by calculating the time delay between the transmission of a Near-Infrared laser signal and its reception after reflecting off of a target. This translates into distance using the known speed of light.

Theory of Operation

To take a measurement, this device first performs a receiver adjustment routine, correcting for changing ambient light levels and allowing maximum sensitivity.

The device sends a reference signal directly from the transmitter to the receiver. It stores the transmit signature, sets the time delay for “zero” distance, and recalculates this delay periodically after several measurements.

Next, the device initiates a measurement by performing a series of acquisitions. Each acquisition is a transmission of the main laser signal while recording the return signal at the receiver. If there is a signal match, the result is stored in memory as a correlation record. The next acquisition is summed with the previous result. When an object at a certain distance reflects the laser signal back to the device, these repeated acquisitions cause a peak to emerge, out of the noise, at the corresponding distance location in the correlation record.

The device integrates acquisitions until the signal peak in the correlation record reaches a maximum value. If the returned signal is not strong enough for this to occur, the device stops at a predetermined maximum acquisition count.

Signal strength is calculated from the magnitude of the signal record peak and a valid signal threshold is calculated from the noise floor. If the peak is above this threshold, the measurement is considered valid and the device will calculate the distance. Otherwise, it will report 1 cm. When beginning the next measurement, the device clears the signal record and starts the sequence again.

Interface

Initialization

On power-up or reset, the device performs a self-test sequence and initializes all registers with default values. After roughly 22 ms, distance measurements can be taken with the I2C interface or the Mode Control Pin.

Power Enable Pin

The enable pin uses an internal pullup resistor, and can be driven low to shut off power to the device.

I2C Interface

This device has a 2-wire, I2C-compatible serial interface (refer to I2C-Bus Specification, Version 2.1, January 2000, available from Philips Semiconductor). It can be connected to an I2C bus as a slave device, under the control of an I2C master device. It supports 400 kHz Fast Mode data transfer.

The I2C bus operates internally at 3.3 Vdc. An internal level shifter allows the bus to run at a maximum of 5 Vdc. Internal 3kΩ pullup resistors ensure this functionality and allow for a simple connection to the I2C host.

The device has a 7-bit slave address with a default value of 0x62. The effective 8-bit I2C address is 0xC4 write and 0xC5 read. The device will not respond to a general call. Support is not provided for 10-bit addressing.

The most significant bit of the register is the byte that follows the I2C address in a normal transaction. Setting this most significant bit of the I2C address byte to one triggers automatic incrementing of the register address with successive reads or writes within an I2C block transfer. This is commonly used to read the two bytes of a 16-bit value within one transfer and is used in the following example.

The simplest method of obtaining measurement results from the I2C interface is as follows:

1 Write 0x04 to register 0x00.

2 Read register 0x01. Repeat until bit 0 (LSB) goes low.

3 Read two bytes from 0x8f (High byte 0x0f then low byte 0x10) to obtain the 16-bit measured distance in centimeters.

A list of all available control resisters is available on page 7.

For more information about the I2C protocol, see I2C Protocol Operation (page 7).

Mode Control Pin

The mode control pin provides a means to trigger acquisitions and return the measured distance via Pulse Width Modulation (PWM) without having to use the I2C interface.

The idle state of the mode control pin is high impedance (High-Z). Pulling the mode control pin low will trigger a single measurement, and the device will respond by driving the line high with a pulse width proportional to the measured distance at 10 µs/cm. A 1kΩ termination resistance is required to prevent bus contention.

The device drives the mode control pin high at 3.3 Vdc. Diode isolation allows the pin to tolerate a maximum of 5 Vdc.

As shown in the diagram PWM Arduino Wiring (page 3), a simple triggering method uses a 1kΩ resistor in series with a host output pin to pull the mode control pin low to initiate a measurement, and a host input pin connected directly to monitor the low-to-high output pulse width.

If the mode control pin is held low, the acquisition process will repeat indefinitely, producing a variable frequency output proportional to distance.

The mode control pin behavior can be modified with the ACQ_CONFIG_REG (0x04) I2C register as detailed in 0x04 (page 8).

Settings

The device can be configured with alternate parameters for the distance measurement algorithm. This can be used to customize performance by enabling configurations that allow choosing between speed, range, and sensitivity. Other useful features are also detailed in this section. See the full Control Register List (page 7) for additional settings.

Acquisition Command

Address	Name	Description	Initial Value
0x00	ACQ_COMMAND	Device command	--

- Writing any non-zero value initiates an acquisition.

Maximum Acquisition Count

Address	Name	Description	Initial Value
0x02	SIG_COUNT_VAL	Maximum acquisition count	0xFF

The maximum acquisition count limits the number of times the device will integrate acquisitions to find a correlation record peak (from a returned signal), which occurs at long range or with low target reflectivity. This controls the minimum measurement rate and maximum range. The unit-less relationship is roughly as follows: rate = 1/n and range = n^(1/4), where n is the number of acquisitions.

Measurement Quick Termination Detection

Address	Name	Description	Initial Value
0x04	ACQ_CONFIG_REG	Acquisition mode control	0x08

You can enable quick-termination detection by clearing bit 3 in this register (starting with the LSB in this register as bit 0). The device will terminate a distance measurement early if it anticipates that the signal peak in the correlation record will reach maximum value. This allows for faster and slightly less accurate operation at strong signal strengths without sacrificing long range performance.

Detection Sensitivity

Address	Name	Description	Initial Value
0x1c	THRESHOLD_ BYPASS	Peak detection threshold bypass	0x00

The default valid measurement detection algorithm is based on the peak value, signal strength, and noise in the correlation record. This can be overridden to become a simple threshold criterion by setting a non-zero value. Recommended non-default values are 0x20 for higher sensitivity with more

frequent erroneous measurements, and 0x60 for reduced sensitivity and fewer erroneous measurements.

Configurable I2C Address

Address	Name	Description	Initial Value
0x16	UNIT_ID_HIGH	Serial number high byte	Unique
0x17	UNIT_ID_LOW	Serial number low byte	Unique
0x18	I2C_ID_HIGH	Write serial number high byte for I2C address unlock	--
0x19	I2C_ID_LOW	Write serial number low byte for I2C address unlock	--
0x1a	I2C_SEC_ADDR	Write new I2C address after unlock	--
0x1e	I2C_CONFIG	Default address response control	0x00

The I2C address can be changed from its default value. Available addresses are 7-bit values with a '0' in the least significant bit (even hexadecimal numbers).

To change the I2C address, the unique serial number of the unit must be read then written back to the device before setting the new address. The process is as follows:

1. Read the two byte serial number from 0x96 (high byte 0x16 and low byte 0x17).
2. Write the serial number high byte to 0x18.
3. Write the serial number low byte to 0x19.
4. Write the desired new I2C address to 0x1a.
5. Write 0x08 to 0x1e to disable the default address.

This can be used to run multiple devices on a single bus, by enabling one, changing its address, then enabling the next device and repeating the process.

The I2C address will be restored to default after a power cycle.

Power Control

Address	Name	Description	Initial Value
0x65	POWER_CONTROL	Power state control	0

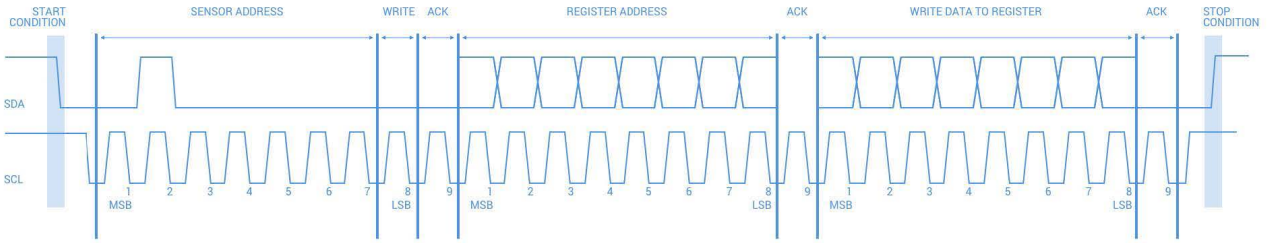
Setting bit 1 in this register disables the receiver circuit, saving roughly 40 mA. After being re-enabled, the receiver circuit stabilizes by the time a measurement can be performed.

NOTE: The most effective way to control power usage is to utilize the enable pin to deactivate the device when not in use.

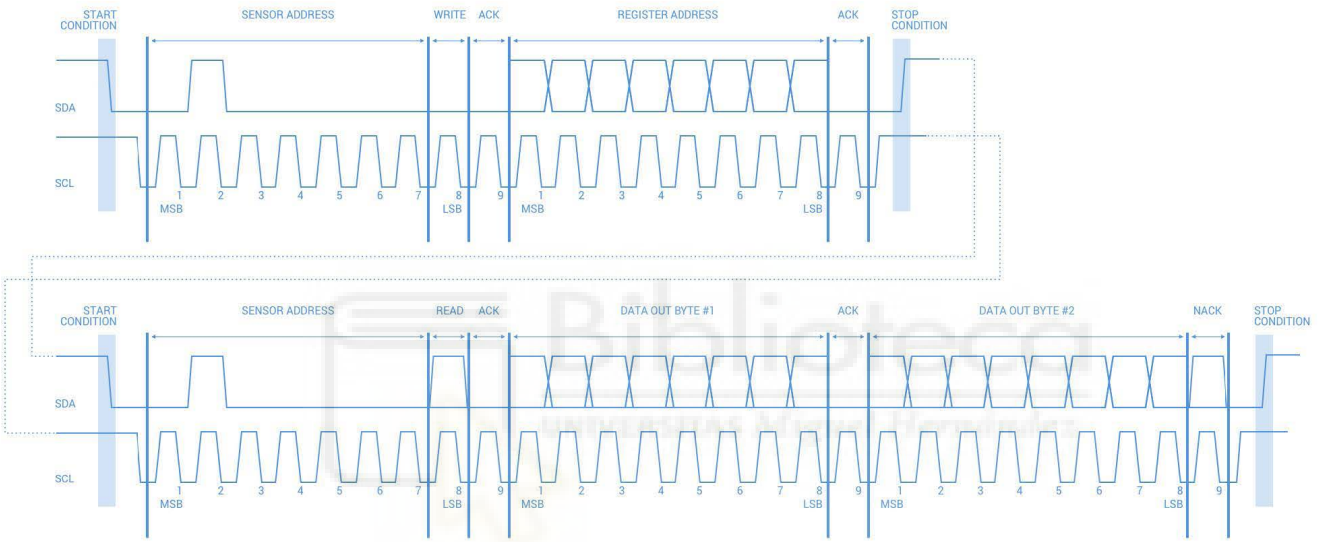
I2C Protocol Information

The sensor module has a 7-bit slave address with a default value of 0x62 in hexadecimal notation. The effective 8 bit I2C address is: 0xC4 write, 0xC5 read. The device will not respond to a general call.

Write



Read



Notes:

- The ACK and NACK items are responses from the master device to the slave device.
- The last NACK in the read is optional, but the formal I2C protocol states that the master shall not acknowledge the last byte.

I2C Protocol Operation

This protocol description uses the term master to refer to the Arduino controller, and uses the term LIDAR device to refer to the LIDAR-Lite v3HP device acting as a slave on the I2C bus.

When working with the I2C serial bus protocol, the device operates as follows:

1. The master initiates data transfer by establishing a start condition, which consists of a high-to-low transition on the SDA line while SCL is high.
2. The master sends an address byte, which consists of the 7-bit slave address.
3. The master sends a read/write bit with a zero state indicating a write request.

   A write operation is used as the initial stage of both read and write transfers.
4. If the slave address corresponds to the LIDAR device address, the LIDAR device responds by pulling SDA low during the ninth clock pulse.

   This operation is considered the acknowledge bit.

   At this stage, all other devices on the bus remain idle while the selected LIDAR device waits for data to be written to or read from its shift register.
5. Data transmits over the serial bus in sequences of nine clock pulses (eight data bits followed by an acknowledge bit).

   These transmissions must occur on the SDA line during the low period of SCL and remain stable during the high period of SCL.
6. The master sends an 8-bit data byte following the slave address, which loads the I2C control register on the LIDAR device with the address of the first control register to be accessed.

   Note: If the high bit (Bit 7) is set, it enables automatic incrementing for successive reads/writes.
7. The master requests a read operation from the LIDAR device or sends a write operation to the LIDAR device.

Read Operation

After the master establishes communication with the LIDAR device, obtaining a reading from the LIDAR device operates as follows.

1. The first data frame sets the address of the desired read register. The master sends a stop bit at the completion of the first data frame.
2. The master initiates a new start condition, which consists of the slave address with the read bit set (one state).
3. The master reads one or more data bytes in succession.
   - A The LIDAR device sends an acknowledge bit to the master when it receives a valid address.
   - B The master releases the SDA data line with continued clocking of the SCL line.
   - C The master strobes the acknowledge bit and continues the read cycle.
4. After the read cycle is done, the master sends a stop condition to complete the operation.

Write Operation

After the master establishes communication with the LIDAR device, writing to the LIDAR device operates as follows.

1. The master sends one or more 8-bit data blocks to the LIDAR device.
   - A The LIDAR device sends an acknowledge bit to the master when it receives and writes a valid data byte.
   - B The master releases the SDA data line with continued clocking of the SCL line.
   - C The master strobes the acknowledge bit and continues the write cycle, if necessary.
3. After the write cycle is done, the master sends a stop condition to complete the operation.

Register Definitions

Control Register List

Address	R/W	Name	Description	Intial Value	Details
0x00	W	ACQ_COMMAND	Device command	--	page 8
0x01	R	STATUS	System status	--	page 8
0x02	R/W	SIG_COUNT_VAL	Maximum acquisition count	0xFF	page 8
0x04	R/W	ACQ_CONFIG_REG	Acquisition mode control	0x08	page 8
0x06	W	LEGACY_RESET_EN	Enables unit reset	--	page 8
0x0e	R	SIGNAL_STRENGTH	Received signal strength	--	page 8
0x0f	R	FULL_DELAY_HIGH	Distance measurement high byte	--	page 8
0x10	R	FULL_DELAY_LOW	Distance measurement low byte	--	page 8
0x12	R/W	REF_COUNT_VAL	Reference acquisition count	0x03	page 8
0x16	R	UNIT_ID_HIGH	Serial number high byte	Unique	page 8
0x17	R	UNIT_ID_LOW	Serial number low byte	Unique	page 9
0x18	W	I2C_ID_HIGH	Write serial number high byte for I2C address unlock	--	page 9
0x19	W	I2C_ID_LOW	Write serial number low byte for I2C address unlock	--	page 9
0x1a	R/W	I2C_SEC_ADDR	Write new I2C address after unlock	--	page 9
0x1c	R/W	THRESHOLD_BYPASS	Peak detection threshold bypass	0x00	page 9
0x1e	R/W	I2C_CONFIG	Default address response control	0x00	page 9
0x26	R/W	PEAK STACK HIGH BYTE	Used for post processing of correlation peak data	--	page 9
0x27	R/W	PEAK STACK LOW BYTE	Used for post processing of correlation peak data	--	page 9
0x40	R/W	COMMAND	State command	--	page 9
0x48	R	HEALTH STATUS	Used to diagnose major hardware issues at initialization	--	page 10
0x52	R	CORR_DATA	Correlation record data low byte	--	page 10
0x53	R	CORR_DATA_SIGN	Correlation record data high byte	--	page 10
0x65	R/W	POWER_CONTROL	Power state control	0	page 10

Detailed Control Register Definitions

NOTE: Unless otherwise noted, all registers contain one byte and are read and write.

0x00

R/W	Name	Description	Initial Value
W	ACQ_COMMAND	Device command	--

Bit	Function
7:1	Write any non-zero value to start a measurement
0	Performs a hard reset by reloading the FPGA and returning all registers to default values This operation must be enabled by writing 1 to bit 0 on register 0x06. When reset the I2C lines go into a high-z state for up to 10 ms. This has the potential to cause legacy-microcontroller-interface code to crash.

0x01

R/W	Name	Description	Initial Value
R	STATUS	System status	--

Bit	Function
5	Health Flag 0: Error detected 1: Reference and receiver bias are operational
4	Device command regulation flag 0: device is not in DC regulation 1: device is in DC regulation
3	Peak detection flag 0: No signal detected 1: Peak detected
2	Reference Overflow Flag 0: Reference data has not overflowed 1: Reference data in correlation record has reached the maximum value before overflow (occurs periodically)
1	Signal Overflow Flag 0: Signal data has not overflowed 1: Signal data in correlation record has reached the maximum value before overflow (occurs with a strong received signal strength)

Additional returns can be evaluated using data downloaded from the peak stack registers, 0x26 and 0x27 (page 9).

0x02

R/W	Name	Description	Initial Value
R/W	SIG_COUNT_VAL	Maximum acquisition count	0xFF

Bit	Function
7:0	Maximum number of acquisitions during measurement

0x04

R/W	Name	Description	Initial Value
R/W	ACQ_CONFIG_REG	Acquisition mode control	0x08

Bit	Function
7	0: Record download resolution set at 9 bits (legacy) 1: Record download resolution set at 12 bits
6	0: Enable reference process during measurement 1: Disable reference process during measurement
5	0: DC compensation enabled 1: DC compensation disabled
4	0: Enable reference filter, averages multiple reference measurements for increased consistency 1: Disable reference filter
3	0: Enable measurement quick termination. Device will terminate distance measurement early if it anticipates that the signal peak in the correlation record will reach maximum value. 1: Disable measurement quick termination.
2	bit unused
1:0	Mode Select Pin Function Control 00: Default PWM mode. Pull pin low to trigger measurement, device will respond with an active high output with a duration of 10us/cm. 01: Status output mode. Device will drive pin active high while busy. Can be used to interrupt host device. 10: Fixed delay PWM mode. Pulling pin low will not trigger a measurement. 11: Oscillator output mode. Nominal 31.25 kHz output. The accuracy of the silicon oscillator in the device is generally within 1% of nominal. This affects distance measurements proportionally and can be measured to apply a compensation factor.

0x06

R/W	Name	Description	Initial Value
R	LEGACY_RESET_EN	Enables legacy unit reset	--

Bit	Function
0	Writing 1 to bit 0 enables the legacy reset operation using the 0x00 register.

0x0e

R/W	Name	Description	Initial Value
R	SIGNAL_STRENGTH	Received signal strength	--

Bit	Function
7:0	Received signal strength calculated from the value of the highest peak in the correlation record and how many acquisitions were performed.

0x0f

R/W	Name	Description	Initial Value
R	FULL_DELAY_HIGH	Distance measurement high byte	--

Bit	Function
7:0	Distance measurement result in centimeters, high byte.

0x10

R/W	Name	Description	Initial Value
R	FULL_DELAY_LOW	Distance measurement low byte	--

Bit	Function
7:0	Distance measurement result in centimeters, low byte.

0x12

R/W	Name	Description	Initial Value
R/W	REF_COUNT_VAL	Reference acquisition count	0x03

Bit	Function
7:0	Non-default number of reference acquisitions during measurement. ACQ_CONFIG_REG (0x04) bit 2 must be set.

0x16

R/W	Name	Description	Initial Value
R	UNIT_ID_HIGH	Serial number high byte	Unique

Bit	Function
7:0	Unique serial number of device, high byte.

0x17

R/W	Name	Description	Initial Value
R	UNIT_ID_LOW	Serial number low byte	Unique

Bit	Function
7:0	Unique serial number of device, high byte.

0x18

R/W	Name	Description	Initial Value
W	I2C_ID_HIGH	Write serial number high byte for I2C address unlock	--

Bit	Function
7:0	Write the value in UNIT_ID_HIGH (0x16) here as part of enabling a non-default I2C address. See I2C_ID_LOW (0x19) and I2C_SEC_ADDR (0x1a).

0x19

R/W	Name	Description	Initial Value
W	I2C_ID_LOW	Write serial number low byte for I2C address unlock	--

Bit	Function
7:0	Write the value in UNIT_ID_LOW (0x17) here as part of enabling a non-default I2C address. See I2C_ID_HIGH (0x18) and I2C_SEC_ADDR (0x1a).

0x1a

R/W	Name	Description	Initial Value
R/W	I2C_SEC_ADDR	Write new I2C address after unlock	--

Bit	Function
7:0	Non-default I2C address. Available addresses are 7-bit values with a '0' in the least significant bit (even hexadecimal numbers). I2C_ID_HIGH (0x18) and I2C_ID_LOW (0x19) must have the correct value for the device to respond to the non-default I2C address.

0x1c

R/W	Name	Description	Initial Value
R/W	THRESHOLD_ BYPASS	Peak detection threshold bypass	0x00

Bit	Function
7:0	0x00: Use default valid measurement detection algorithm based on the peak value, signal strength, and noise in the correlation record. 0x01-0xff: Set simple threshold for valid measurement detection. Values 0x20-0x60 generally perform well.

0x1e

R/W	Name	Description	Initial Value
R/W	I2C_CONFIG	Default address response control	0x00

Bit	Function
5	0: Disables the alternate status mode. 1: Enables an alternate indication status byte at STATUS register 0x01. NOTE: If bit 5 is enabled (1), the status word consists of all ones except for the bit position selected by bits [2:0] in this I2C CONFIG register (0x1e). This allows for the reading of the busy status of multiple units sharing the same active base address 0x62.
4	0: Disables the altrenative I2C address. 1: Enables the alternative I2C address.
3	0: Device will respond to I2C address 0x62. Device will also respond to non-default address if configured successfully. See I2C_ID_HIGH (0x18), I2C_ID_LOW (0x19), and I2C_SEC_ADDR (0x1a). 1: Device will only respond to non-default I2C address. It is recommended to configure the non-default address first, then use the non-default address to write to this register, ensuring success.
2:0	Defines the bit position(s) to remain set as 0 when bit 5 is enabled.

0x26

R/W	Name	Description	Initial Value
R/W	PEAK STACK HIGH BYTE	Registers read successive values from the peak stack register. Data from the stack register is used for post processing.	--

Bit	Function
10:8	For every 11-bit stack value, this resister (0x26) must be read first. Reading from this register latches the low order data into 0x27 and increments the stack pointer. Writing 0x01 to this register (0x26) resets the stack pointer to the first element.

0x27

R/W	Name	Description	Initial Value
R/W	PEAK STACK LOW BYTE	Registers read successive values from the peak stack register. Data from the stack register is used for post processing.	--

Bit	Function
7:0	Reading from 0x27 reads the low order data from this register.

0x40

R/W	Name	Description	Initial Value
R/W	TEST COMMAND	State command	--

Bit	Function
2:0	000: Test mode disable, resume normal operation 111: Test mode enable, allows download of correlation record Once test mode is enabled, read CORR_DATA (0x52) and CORR_DATA_ SIGN (0x53) in one transaction (read from 0xd2). The memory index is incremented automatically and successive reads produce sequential data.

0x48

R/W	Name	Description	Initial Value
R	HEALTH STATUS	Used to diagnose major hardware issues at system initialization.	--

Bit	Function
4:0	Reference value is within normal range.
3	Reference overflow occurred during the first acquisition.
2	An initial acquisition was completed at wake-up to set the initial reference value.
1	The receiver DC control command is within the normal range.
0	DC regulation was successful during wake-up.

0x52

R/W	Name	Description	Initial Value
R	CORR_DATA	Correlation record data low byte	--

Bit	Function
7:0	Correlation record data low byte. See CORR_DATA_SIGN (0x53), ACQ_ SETTINGS (0x5d), and COMMAND (0x40).

0x53

R/W	Name	Description	Initial Value
R	CORR_DATA_SIGN	Correlation record data high byte	--

Bit	Function
7:0	Correlation record data high byte. Correlation record data is a 2's complement 9-bit value, and must be sign extended to be formatted as a 16-bit 2's complement value. Thus when repacking the two bytes obtained for the I2C transaction, set the high byte to 0xff if the LSB of the high byte is one.

0x65

R/W	Name	Description	Initial Value
R/W	POWER_CONTROL	Power state control	0x80

Bit	Function
0	1: Disable receiver circuit 0: Enable receiver circuit. Receiver circuit stabilizes by the time a measurement can be performed.

Frequently Asked Questions

How do I use the device for fast-scanning applications?

Using the LIDAR-Lite v3HP device for fast-scanning applications may require you to change the program you used for "continuous" or "burst" mode functions with previous versions of the sensor.

1. Initiate new measurement command.
2. Immediately read the distance registers, obtaining the previous measurement results while the new measurement is occurring.

   Measurement data stored in the sensor is valid until a new measurement concludes.
3. Perform other actions while polling the status bit until it indicates an idle state.
4. Repeat steps 1 through 3.

NOTES:

- This method uses slightly more I2C overhead, but it allows more efficient polling if you know about your measurement time, which depends on maximum acquisition count settings. You also know exactly when that measurement begins.
- With this approach (and nothing else going on except relentless polling), the device has been able to reach >1.5 kHz with very small acquisition count settings.
- You can find sample Arduinio code for this in the Garmin GitHub repository at the following location: https://github.com/garmin/LIDARLite_v3_Arduino_Library/blob/master/examples/ShortRangeHighSpeed/ShortRangeHighSpeed.ino.

Does the device operate only on 5 Vdc?

The device requires 5 Vdc to function properly.

NOTICE

Connecting the device to a source greater or less than 5 Vdc is not supported, and may result in poor performance or may damage the device.

What is the spread of the laser beam?

At very close distances (less than 1 m), the beam diameter is about the size of the aperture (lens). For distances greater than 1 m, you can estimate the beam diameter using this equation:

Distance/100 = beam diameter at that distance (in whatever units you measured the distance).

The actual spread is ~8 milli radians or ~1/2 degree.

How do distance, target size, aspect, and reflectivity affect returned signal strength?

The device transmits a focused infrared beam that reflects off of a target, and a portion of that reflected signal returns to the receiver. The distance is calculated by taking the difference between the moment of signal transmission to the moment of signal reception. Successfully receiving a reflected signal is heavily influenced by several factors. These factors include:

- Target Distance

  The relationship of distance (D) to returned signal strength is an inverse square. With an increase in distance, the returned signal strength decreases by 1/D^2 or the square root of the distance.
- Target Size

  The relationship of a target's Cross Section (C) to returned signal strength is an inverse power of four. The device transmits a focused near-infrared laser beam that spreads at a rate of approximately 0.5º as distance increases. Up to 1 m, it is approximately the size of the lens. Beyond 1 m, the approximate beam spread in degrees can be estimated by dividing the distance by 100, or ~8 milliradians. When the beam overfills (is larger than) the target, the signal returned decreases by 1/C^4 or the fourth root of the target's cross section.

- Aspect

  The aspect of the target, or its orientation to the sensor, affects the observable cross section and, therefore, the amount of returned signal decreases as the aspect of the target varies from the normal.
- Reflectivity

  Reflectivity characteristics of the target's surface also affect the amount of returned signal (How does the device work with reflective surfaces?, page 11).

In summary, a small target can be very difficult to detect if it is distant, poorly reflective, and its aspect is away from the normal. In such cases, the returned signal strength may be improved by attaching infrared reflectors to the target, increasing the size of the target, modifying its aspect, or reducing distance from the sensor.

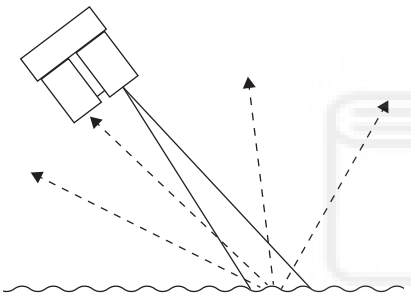
How does the device work with reflective surfaces?

Reflective characteristics of an object's surface can be divided into three categories:

- Diffuse Reflective
- Specular
- Retro-reflective

Diffuse Reflective Surfaces

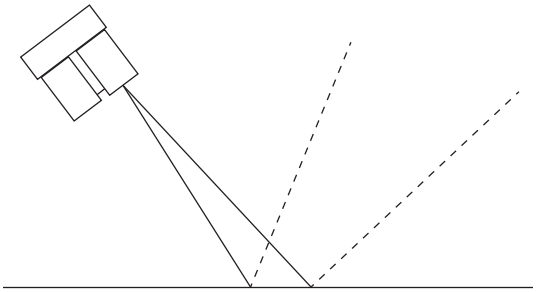
Purely diffuse surfaces are found on materials that have a textured quality that causes reflected energy to disperse uniformly. This tendency results in a relatively predictable percentage of the dispersed laser energy finding its way back to the device. As a result, these materials tend to read very well.



Materials that fall into this category are paper, matte walls, and granite. It is important to note that materials that fit into this category due to observed reflection at visible light wavelengths may exhibit unexpected results in other wavelengths. The near infrared range used by the device may detect them as nearly identical. For example, a black sheet of paper may reflect a nearly identical percentage of the infrared signal back to the receiver as a white sheet.

Specular Surfaces

Specular surfaces, are found on materials that have a smooth quality that reflect energy instead of dispersing it. It is difficult or impossible for the device to recognize the distance of many specular surfaces. Reflections off of specular surfaces tend to reflect with little dispersion which causes the reflected beam to remain small and, if not reflected directly back to the receiver, to miss the receiver altogether. The device may fail to detect a specular object in front of it unless viewed from the normal.



Examples of specular surfaces are mirrors and glass viewed off-axis.

How does liquid affect the signal?

There are a few considerations to take into account if your application requires measuring distances to, or within, liquid:

- Reflectivity and other characteristics of the liquid itself
- Reflectivity characteristics of particles suspended in the liquid
- Turbidity
- Refractive characteristics of the liquid

Reflectivity of the liquid is important when measuring distance to the surface of a liquid or if measuring through liquid to the bottom of a container (How does the device work with reflective surfaces?, page 11).

Measuring distance with the device depends on reflected energy from the transmitted signal being detected by the receiver in the sensor. For that reason, the surface condition of the liquid may play an important role in the overall reflectivity and detectability of the liquid. In the case of a flat, highly reflective liquid surface, the laser's reflected energy may not disperse adequately to allow detection unless viewed from the normal. By contrast, small surface ripples may create enough dispersion of the reflected energy to allow detection of the liquid without the need to position the sensor so that the transmitted beam strikes the liquid's surface from the normal.

Reflectivity of suspended particles is a characteristic that may help or hinder, depending on the application.

Turbidity, or the clarity of a liquid created by the presence or absence of suspended particles, can similarly help or hinder measurement efforts. If the application requires detecting the surface of the liquid, then suspended particles may help by reflecting more of the transmitted beam back to the receiver, increasing detectability and permitting measurements to be taken.

Attempting to measure through suspended particles in a liquid will only be successful if the transmitted beam is allowed to reflect off of the desired target without first being absorbed or reflected by the suspended particles.

When the near infrared energy transmitted by the device transitions from the atmosphere to a liquid, the energy may be bent, or refracted, and absorbed in addition to being dispersed. The degree to which the transmitted beam is refracted and absorbed is defined by its refraction index. That being said, the most important criteria impacting successful measurement through a liquid is the amount of dispersion of the transmitted beam and whether any of the dispersed beam makes its way back to the receiver on the device.

Electromagnetic energy travels slower through a liquid and may affect accuracy of the final measurement output.

GARMIN®

Garmin International, Inc.
1200 East 151st Street, Olathe, Kansas 66062, USA

Garmin (Europe) Ltd.
Liberty House, Hounsdown Business Park, Southampton, Hampshire, SO40 9LR UK

Garmin Corporation
No. 68, Zhangshu 2nd Road, Xizhi Dist., New Taipei City, 221, Taiwan (R.O.C.)

www.garmin.com

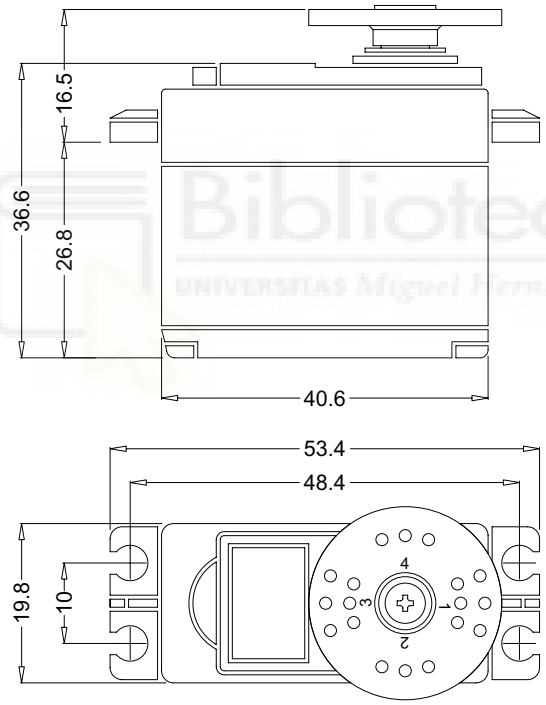
April 2018

190-02088-02_0A

ANNOUNCED SPECIFICATION OF HS-422 STANDARD DELUXE SERVO

1.TECHNICAL VALUES

CONTROL SYSTEM	: +PULSE WIDTH CONTROL 1500usec NEUTRAL	
OPERATING VOLTAGE RANGE	: 4.8V TO 6.0V	
OPERATING TEMPERATURE RANGE	: -20 T0 +60°C	
TEST VOLTAGE	: AT 4.8V	AT 6.0V
OPERATING SPEED	: 0.21sec/60° AT NO LOAD	0.16sec/60°AT NO LOAD
STALL TORQUE	: 3.3kg.cm(45.82oz.in)	4.1kg.cm(56.93oz.in)
OPERATING ANGLE	: 45°/ONE SIDE PULSE TRAVELING 400usec	
DIRECTION	: CLOCK WISE/PULSE TRAVELING 1500 TO 1900usec	
CURRENT DRAIN	: 8mA/IDLE AND 150mA/NO LOAD RUNNING	
DEAD BAND WIDTH	: 8usec	
CONNECTOR WIRE LENGTH	: 300mm(11.81in)	
DIMENSIONS	: 40.6x19.8x36.6mm(1.59x0.77x1.44in)	
WEIGHT	: 45.5g(1.6oz)	



2.FEATURES

3-POLE FERRITE MOTOR
LONG LIFE POTENTIOMETER
DUAL OILITE BUSHING
INDIRECT POTENTIOMETER DRIVE

3.APPLICATIONS

AIRCRAFT 20-60 SIZE
30 SIZE HELICOPTERS
STEERING AND THROTTLE SERVO FOR CARS
TRUCK AND BOATS

LCD & Keypad Shield Quickstart Guide

The 16x2 LCD And Keypad Shield is very simple to use because it's fully compatible with the Arduino "LiquidCrystal" library. You can initialise the LCD and display messages on it with just a few lines of code, but it also gives you the flexibility to do more advanced projects such as display menu items and select them using the buttons.

Power Requirements

The LCD & Keypad Shield requires a good 5V power supply to ensure the backlight fully illuminates and the display contrast is high, and if you power your Arduino from USB with the LCD Shield attached you may experience a voltage drop over the USB cable. If you have trouble with display contrast or backlight brightness, try attaching a power supply of around 7 to 9Vdc to the 2.1mm DC jack on the Arduino. A typical symptom in an undervoltage situation is that one line of the LCD will show pale rectangles in place of the characters, and the other line will show nothing at all. The Arduino may even continue running normally because it's quite happy at just 4V or so, but the LCD & Keypad Shield won't function.

Library Requirements

All the hard work of interfacing with the LCD Shield is handled by the LiquidCrystal library, which is included as part of the official Arduino distribution. You can check whether you have it installed by starting up the IDE and looking under Files -> Examples -> LiquidCrystal. If it exists, you're good to go.

Minimal Display Example

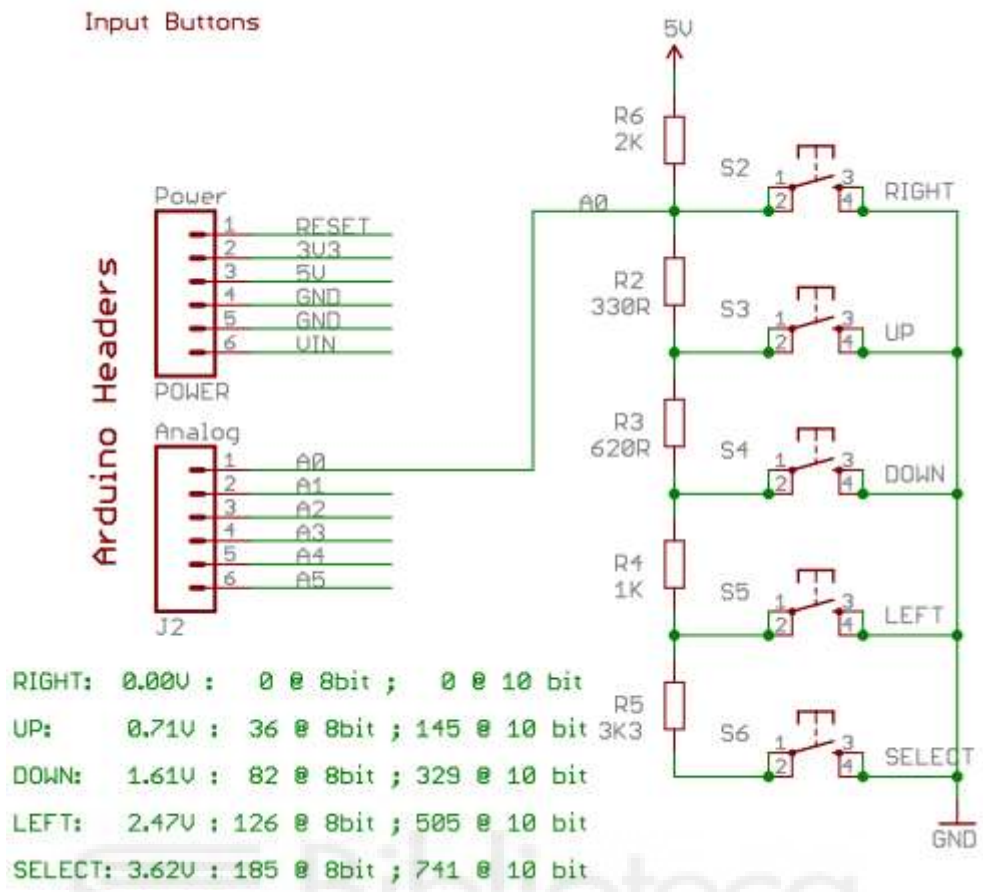
To start up the LCD and display a message, open a new sketch in the Arduino IDE and paste in the following code:

```
#include <Wire.h>
#include <LiquidCrystal.h>

LiquidCrystal lcd( 8, 9, 4, 5, 6, 7 );

void setup()
{
  lcd.begin(16, 2);
  lcd.print("hello, world!");
}

void loop()
{
   // your main loop code here...
}
```

Reading The Buttons

The LCD Shield includes 5 buttons designed for use as navigational or control input. The buttons are arranged in a handy pattern and referred to as UP, DOWN, LEFT, RIGHT, and SELECT, but of course it's totally up to your sketch to decide what to do when any particular button is pressed.

All the buttons are connected to a single analog input, A0, using a chain of resistors that causes a different reference voltage to be applied to A0 depending on which button is pressed. This section of the shield schematic shows the input buttons and associated resistors:



As you can see, if no button is being pressed the voltage on A0 will be pulled all the way up to 5V by the 2K resistor called R6. In that situation none of the other resistors have any effect at all, and the analog reading on A0 will be hard on the upper limit of 1023. Therefore if you perform an analogRead() call on A0 and it returns 1023 (or any value above about 1000) you know that no buttons are being pressed.

Now consider what happens if the "DOWN" button is pressed. Now A0 is being presented with a voltage that is divided between the 2K resistor that is trying to pull it up to 5V, and the 330R and 620R resistors in series (totaling 950R) that are trying to pull it down to 0V. The voltage presented to A0 in that case is about 1.61V, which means if you perform an analogRead() on A0 it will return a value of about 329. So if you read a value of about 329 from A0 you know the "DOWN" button is being pressed.

The same principle applies for the other buttons, with the voltages and equivalent analogRead() values shown on the schematic above.

This is a neat way to provide a whole set of input buttons while only using up one of the I/O pins on your Arduino, leaving more pins free for use in your project.

Complex Example

The extensive example below combines a number of techniques to demonstrate how to show messages on the LCD, read from the buttons, and change the display message depending on which buttons are pressed.

```
/*
  Example code for the Freetronics LCD & Keypad Shield:

    http://www.freetronics.com/products/lcd-keypad-shield

  by Marc Alexander, 7 September 2011
  This example code is in the public domain.

  ----------------------------------------------------------------

  This program demonstrates button detection, LCD text/number printing,
  and LCD backlight control on the Freetronics LCD & Keypad Shield, connected to an Arduino board.

  After powerup, the screen looks like this:
```

```
  ------------------
  |Freetronics 16x2|
  |Btn:         0 | <- This time value counts up the number of seconds since reset (overflows at 99)
  ------------------
```

When a button is pressed, a label appears for it:

```
  ------------------
  |Freetronics 16x2|
  |Btn:RIGHT     0 |
  ------------------
  Labels are LEFT, UP, DOWN, RIGHT and SELECT-FLASH.
  SELECT-FLASH makes the LCD backlight flash off and on when held down.
```

Pins used by LCD & Keypad Shield:

```
  A0: Buttons, analog input from voltage ladder
  D4: LCD bit 4
  D5: LCD bit 5
  D6: LCD bit 6
  D7: LCD bit 7
  D8: LCD RS
  D9: LCD E
  D3: LCD Backlight (high = on, also has pullup high so default is on)
```

ADC voltages for the 5 buttons on analog input pin A0:

```
  RIGHT:  0.00V :   0 @ 8bit ;   0 @ 10 bit
  UP:     0.71V :  36 @ 8bit ; 145 @ 10 bit
  DOWN:   1.61V :  82 @ 8bit ; 329 @ 10 bit
  LEFT:   2.47V : 126 @ 8bit ; 505 @ 10 bit
  SELECT: 3.62V : 185 @ 8bit ; 741 @ 10 bit
*/
/*-----------------------------------------------------------------------------
  Includes
-----------------------------------------------------------------------------*/
#include <Wire.h>
#include <LiquidCrystal.h>   // include LCD library
/*-----------------------------------------------------------------------------
  Defines
-----------------------------------------------------------------------------*/
// Pins in use
#define BUTTON_ADC_PIN          A0  // A0 is the button ADC input
#define LCD_BACKLIGHT_PIN        3  // D3 controls LCD backlight
// ADC readings expected for the 5 buttons on the ADC input
#define RIGHT_10BIT_ADC          0  // right
#define UP_10BIT_ADC           145  // up
#define DOWN_10BIT_ADC         329  // down
#define LEFT_10BIT_ADC         505  // left
#define SELECT_10BIT_ADC       741  // right
#define BUTTONHYSTERESIS        10  // hysteresis for valid button sensing window
//return values for ReadButtons()
#define BUTTON_NONE              0  //
#define BUTTON_RIGHT             1  //
#define BUTTON_UP                2  //
#define BUTTON_DOWN              3  //
#define BUTTON_LEFT              4  //
#define BUTTON_SELECT            5  //
//some example macros with friendly labels for LCD backlight/pin control, tested and can be swapped into the example code as
you like
#define LCD_BACKLIGHT_OFF()     digitalWrite( LCD_BACKLIGHT_PIN, LOW )
#define LCD_BACKLIGHT_ON()      digitalWrite( LCD_BACKLIGHT_PIN, HIGH )
#define LCD_BACKLIGHT(state)    { if( state ){digitalWrite( LCD_BACKLIGHT_PIN, HIGH );}else{digitalWrite(
LCD_BACKLIGHT_PIN, LOW );} }
/*-----------------------------------------------------------------------------
  Variables
-----------------------------------------------------------------------------*/
byte buttonJustPressed  = false;         //this will be true after a ReadButtons() call if triggered
```

```
byte buttonJustReleased = false;        //this will be true after a ReadButtons() call if triggered
byte buttonWas          = BUTTON_NONE;   //used by ReadButtons() for detection of button events
/*--------------------------------------------------------------------------------------
  Init the LCD library with the LCD pins to be used
--------------------------------------------------------------------------------------*/
LiquidCrystal lcd( 8, 9, 4, 5, 6, 7 );   //Pins for the freetronics 16x2 LCD shield. LCD: ( RS, E, LCD-D4, LCD-D5, LCD-D6, LCD-D7 )
/*--------------------------------------------------------------------------------------
  setup()
  Called by the Arduino framework once, before the main loop begins
--------------------------------------------------------------------------------------*/
void setup()
{
  //button adc input
  pinMode( BUTTON_ADC_PIN, INPUT );         //ensure A0 is an input
  digitalWrite( BUTTON_ADC_PIN, LOW );      //ensure pullup is off on A0
  //lcd backlight control
  digitalWrite( LCD_BACKLIGHT_PIN, HIGH );  //backlight control pin D3 is high (on)
  pinMode( LCD_BACKLIGHT_PIN, OUTPUT );     //D3 is an output
  //set up the LCD number of columns and rows:
  lcd.begin( 16, 2 );
  //Print some initial text to the LCD.
  lcd.setCursor( 0, 0 );   //top left
  //         1234567890123456
  lcd.print( "Freetronics 16x2" );
  //
  lcd.setCursor( 0, 1 );   //bottom left
  //         1234567890123456
  lcd.print( "Btn:" );
}
/*--------------------------------------------------------------------------------------
  loop()
  Arduino main loop
--------------------------------------------------------------------------------------*/
void loop()
{
  byte button;
  byte timestamp;

  //get the latest button pressed, also the buttonJustPressed, buttonJustReleased flags
  button = ReadButtons();
  //blank the demo text line if a new button is pressed or released, ready for a new label to be written
  if( buttonJustPressed || buttonJustReleased )
  {
    lcd.setCursor( 4, 1 );
    lcd.print( "           " );
  }
  //show text label for the button pressed
  switch( button )
  {
    case BUTTON_NONE:
    {
      break;
    }
    case BUTTON_RIGHT:
    {
      lcd.setCursor( 4, 1 );
      lcd.print( "RIGHT" );
      break;
    }
    case BUTTON_UP:
    {
      lcd.setCursor( 4, 1 );
      lcd.print( "UP" );
      break;
    }
    case BUTTON_DOWN:
    {
```

```
            lcd.setCursor( 4, 1 );
            lcd.print( "DOWN" );
            break;
          }
          case BUTTON_LEFT:
          {
            lcd.setCursor( 4, 1 );
            lcd.print( "LEFT" );
            break;
          }
          case BUTTON_SELECT:
          {
            lcd.setCursor( 4, 1 );
            lcd.print( "SELECT-FLASH" );

            //SELECT is a special case, it pulses the LCD backlight off and on for demo
            digitalWrite( LCD_BACKLIGHT_PIN, LOW );
            delay( 150 );
            digitalWrite( LCD_BACKLIGHT_PIN, HIGH );   //leave the backlight on at exit
            delay( 150 );

            /* an example of LCD backlight control via macros with nice labels
            LCD_BACKLIGHT_OFF();
            delay( 150 );
            LCD_BACKLIGHT_ON();   //leave the backlight on at exit
            delay( 150 );
            */

            /*
            // an example of LCD backlight control via a macro with nice label, called with a value
            LCD_BACKLIGHT(false);
            delay( 150 );
            LCD_BACKLIGHT(true);   //leave the backlight on at exit
            delay( 150 );
            */

            break;
          }
          default:
          {
            break;
          }
        }
        // print the number of seconds since reset (two digits only)
        timestamp = ( (millis() / 1000) % 100 );   //"% 100" is the remainder of a divide-by-100, which keeps the value as 0-99 even as
      the result goes over 100
        lcd.setCursor( 14, 1 );
        if( timestamp <= 9 )
          lcd.print( " " );   //quick trick to right-justify this 2 digit value when it's a single digit
        lcd.print( timestamp, DEC );
      /*
        //debug/test display of the adc reading for the button input voltage pin.
        lcd.setCursor(12, 0);
        lcd.print( "    " );         //quick hack to blank over default left-justification from lcd.print()
        lcd.setCursor(12, 0);        //note the value will be flickering/faint on the LCD
        lcd.print( analogRead( BUTTON_ADC_PIN ) );
      */
        //clear the buttonJustPressed or buttonJustReleased flags, they've already done their job now.
        if( buttonJustPressed )
          buttonJustPressed = false;
        if( buttonJustReleased )
          buttonJustReleased = false;
      }
      /*--------------------------------------------------------------------------------
        ReadButtons()
        Detect the button pressed and return the value
        Uses global values buttonWas, buttonJustPressed, buttonJustReleased.
      --------------------------------------------------------------------------------*/
```

```
byte ReadButtons()
{
  unsigned int buttonVoltage;
  byte button = BUTTON_NONE;   // return no button pressed if the below checks don't write to btn

  //read the button ADC pin voltage
  buttonVoltage = analogRead( BUTTON_ADC_PIN );
  //sense if the voltage falls within valid voltage windows
  if( buttonVoltage < ( RIGHT_10BIT_ADC + BUTTONHYSTERESIS ) )
  {
    button = BUTTON_RIGHT;
  }
  else if(   buttonVoltage >= ( UP_10BIT_ADC - BUTTONHYSTERESIS )
        && buttonVoltage <= ( UP_10BIT_ADC + BUTTONHYSTERESIS ) )
  {
    button = BUTTON_UP;
  }
  else if(   buttonVoltage >= ( DOWN_10BIT_ADC - BUTTONHYSTERESIS )
        && buttonVoltage <= ( DOWN_10BIT_ADC + BUTTONHYSTERESIS ) )
  {
    button = BUTTON_DOWN;
  }
  else if(   buttonVoltage >= ( LEFT_10BIT_ADC - BUTTONHYSTERESIS )
        && buttonVoltage <= ( LEFT_10BIT_ADC + BUTTONHYSTERESIS ) )
  {
    button = BUTTON_LEFT;
  }
  else if(   buttonVoltage >= ( SELECT_10BIT_ADC - BUTTONHYSTERESIS )
        && buttonVoltage <= ( SELECT_10BIT_ADC + BUTTONHYSTERESIS ) )
  {
    button = BUTTON_SELECT;
  }
  //handle button flags for just pressed and just released events
  if( ( buttonWas == BUTTON_NONE ) && ( button != BUTTON_NONE ) )
  {
    //the button was just pressed, set buttonJustPressed, this can optionally be used to trigger a once-off action for a button
press event
    //it's the duty of the receiver to clear these flags if it wants to detect a new button change event
    buttonJustPressed  = true;
    buttonJustReleased = false;
  }
  if( ( buttonWas != BUTTON_NONE ) && ( button == BUTTON_NONE ) )
  {
    buttonJustPressed  = false;
    buttonJustReleased = true;
  }

  //save the latest button value, for change event detection next time round
  buttonWas = button;

  return( button );
}
```