UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

DISEÑO DE UN SISTEMA DE POSICIONAMIENTO DE DRONES MEDIANTE LIDAR Y RASPBERRY PI

AUTOR: ÁNGEL ESPÍN CARO

TUTOR: ÁNGEL RODAS JORDÁ

COTUTOR: ISRAEL QUINTANILLA GARCÍA

Curso Académico: 2019-20

Agradecimientos

Ante todo, agradecer al profesor Ángel Rodas por su guía en la realización de este proyecto. También agradecer a mi familia por su apoyo al leerse el documento en busca de mejoras, aun cuando el contenido del mismo se escapa de sus campos de trabajo. En especial a mi padre, por su ayuda en las cuestiones técnicas y seguimiento constante de los progresos. Y a mi pareja por estar siempre a mi lado.

Por último, agradecer a todos los profesores y mentores que se han esforzado por dotarme de los conocimientos para llegar a ser un ingeniero de calidad.

Resumen

Este documento pretende abordar el diseño de un sistema con capacidad para dotar a un dron de auto posicionamiento. Empieza por un breve resumen de las tecnologías para el posicionamiento existentes, así como de sus usos. Seguidamente, describe la metodología empleada para la selección de la tecnología a utilizar para la obtención de datos del entorno a un dron. También se recogen las implementaciones en una Raspberry Pi de los diversos sensores comparados y de un sistema para la interpretación y comunicación de los datos. Y finaliza con la propuesta del diseño de un dron para incorporar el sensor de la tecnología LiDAR seleccionada.

Resum

Aquest document pretén abordar el disseny d'un sistema amb capacitat de dotar a un dron de auto posicionar-se. Comença amb un breu resum de sobre les tecnologies de posicionament existents, així com els seus usos. Seguidament es descriuen la metodologia utilitzada per a la selecció de la tecnologia a utilitzar per a la obtenció de dades del entorn a un dron. També es recolleixen les implementacions en una Raspberry Pi dels diversos sensors comparats y d'un sistema per a la interpretació y comunicació de les dades. I finalitza amb la proposta de disseny d'un dron per a incorporar el sensor de la tecnologia LiDAR seleccionada.

Abstract

This document is intended to address the design of a system with the capacity to give a drone the ability to auto positioning. It begins with a brief abstract of the current positioning technologies, as well as its uses. It follows with the description of the methodology used to select the technology that is going to use a drone to get the data from the environment. Also, this document gathers the implementations in a Raspberry Pi of the various compared sensors. And ends with the proposal of a design of a drone to incorporate the selected LiDAR technology sensor.

Índice

1.- Motivación

La tecnología emergente de los drones y su inclusión en muchos procesos me pareció una gran oportunidad de investigar el sector y aportar innovaciones. Desde mi punto de vista especializado en la electrónica y automática, la perspectiva de tener la oportunidad de implementar un sistema de adquisición de datos que se pudiera utilizar en un dron resultaba inspiradora, y más si el sistema se puede utilizar posteriormente para dotar de una forma solida y flexible a un dron de un sistema de posicionamiento.

Un dispositivo capaz de obtener información de un entorno complejo puede suponer un gran apoyo para el trabajo en zonas peligrosas o poco accesibles para el ser humano: edificios, ruinas, terreno accidentado, incendios, etc. Los sectores que se pueden aprovechar de este tipo de innovaciones son muy variados tanto industriales, como militares o de entretenimiento.

La gran mayoría de sistemas utilizan un posicionamiento basado en coordenadas GPS y no suelen tener en cuenta el entorno, por lo que su uso está limitado a espacios abierto, con buena señal de GPS y siempre bajo una supervisión permanente del usuario. Por eso resulta crucial solventar el problema de la obtención de información del entorno en los sistemas autónomos.

Frente a cómo solventar la detección del entorno, hay una gran variedad de sistemas en el mercado. El reto será implementar un sistema preciso y más económico que los sistemas más vanguardistas de visión artificial. Estos sistemas requieren de gran capacidad de cómputo y de equipos especializados que van más allá de la capacidad de un microcontrolador o un ordenador de placa reducida, como es la Raspberry Pi, al alcance de cualquiera.

Al proyecto podré aportar mi experiencia como director de proyecto de los robot aspirador Conga de la empresa Cecotec, donde supervise la creación de diversos dispositivos con tecnologías similares a las que se van a aplicar en este proyecto. Y podre completar mi experiencia añadiendo a mis conocimientos información sobre sensores que no había tenido en cuenta, como pueden ser los ultrasonidos.

Es un gran incentivo trasladar mis conocimientos prácticos a un proyecto propio, con grandes posibilidades.

2.- Objetivos

El objetivo de este proyecto es crear un sistema que permita realizar el posicionamiento de un dron. Es decir, crear las herramientas para que un dron sea capaz de detectar obstáculos.

Para ello dividiremos el proyecto en diversas partes:

- Una investigación sobre las tipologías de sensor que incorporar al sistema de percepción del entorno. Este objetivo, al ser el aspecto fundamental del proyecto lo dividiremos en:
    - Creación de un sistema de ensayo de diferentes tecnologías.
    - Ensayo con ultrasonidos.
    - Ensayo con láser.
    - Ensayo con infrarrojos.
    - Elección de tecnología.
- La implementación de un sistema LiDAR.
- La implementación de todos los sistemas de adquisición en un microprocesador Raspberry Pi.
- La programación en Python del sistema de comunicación del dron con un terminal de tierra vía Wi-Fi, para la obtención de datos.
- Propuesta de montaje de un dron con autopiloto para soportar el sensor escogido.

3.- Estado del arte

Para poder plantear la realización del proyecto necesitamos conocer qué tecnologías existen y cuál es su estado de madurez.

3.1.- Sistemas autónomos

Como su nombre indica son sistemas que no dependen de nadie más que de sí mismos para desarrollar sus funciones, no dependen de ningún sistema superior que los supervise o guíe.

El mayor problema que resolver de estos sistemas es su percepción del entorno para poder moverse e interactuar con él, es decir, el mapeo y localización simultáneos (SLAM), que es como se denomina al problema de estimar una trayectoria al desplazarse en un entorno desconocido al mismo tiempo que se realiza un mapa de este.

El ruido presente en los sistemas sensoriales, los inevitables errores y aproximaciones cometidos en los modelos empleados, y la dificultad representativa de los entornos a medida que éstos aumentan en complejidad, hacen que la tarea de resolver el mencionado problema sea ardua. Tal complejidad tiene una doble vertiente:

Desde un punto de vista conceptual se impone la necesidad de razonar en un mundo a veces confuso, en ocasiones dinámico y cambiante, aprendido mediante sensores que distan mucho de ser perfectos. En estas condiciones se busca la manera de obtener y manipular datos acerca del entorno, extraer aquel conocimiento que sea sustancial para la tarea de su representación, e integrar la información así obtenida del modo más conveniente. Muchas preguntas surgen en este nivel de complejidad: ¿qué es el entorno?, ¿qué geometrías cabe esperar

encontrar en él?, ¿existen objetos móviles?, ¿con qué precisión es necesario representarlo?, ¿qué nivel de conocimiento permitirá obtener el mapa generado?

La otra vertiente de la complejidad del problema tiene que ver con el aspecto computacional de las soluciones planteadas al problema del SLAM, y está indisolublemente ligada a la anterior. El modo en que el robot perciba su entorno, la cantidad de información disponible, así como las técnicas empleadas en su procesamiento, interpretación y combinación, determinarán los recursos computacionales necesarios para la construcción del mapa. Estos recursos no son ilimitados; menos aún si el objetivo es ceñirse a los disponibles a bordo de la máquina. Aquí los interrogantes tienen que ver con la idoneidad de los algoritmos utilizados y la posibilidad de obtener soluciones cuya implementación sea posible en tiempo real.

Así pues, en la base de cualquier solución al problema del SLAM se encuentra siempre con la necesidad de trabajar con cantidades progresivamente crecientes de información, contaminada en mayor o menor medida por ruido, y manipulada mediante modelos que, la mayoría de las veces, no son sino meras aproximaciones a la realidad. No es de extrañar, por lo tanto, que las soluciones más exitosas hasta el momento hayan estado basadas en la utilización de técnicas probabilísticas.

Para realizar el SLAM se pueden usar muchos tipos diferentes de sensores, las bondades y defectos de cada sensor llevan a la creación de nuevos algoritmos. El requisito indispensable para que un sensor se pueda usar para el SLAM es que sea estadísticamente independiente para hacer frente a las medidas parciales y al ruido en las medidas. Los diferentes tipos de sensores pueden ir desde los escáneres láser a los sensores táctiles dando lugar a una gran variedad de soluciones prácticas con sus algoritmos relacionados.

3.2- Sistemas de tiempo real

Por lo general las decisiones de los sistemas autónomos están ligadas a un tiempo concreto, si hay una pared y no giran a tiempo chocarán con ella, por ejemplo. Los sistemas de tiempo real (STR) son sistemas deterministas en los que, ante determinadas entradas, tienen salidas conocidas y además también responden en un tiempo conocido. Estos sistemas son utilizados en situaciones donde el tiempo es crítico.

Los STR se pueden dividir en dos grupos:

- Tiempo real estricto (hard real time): Cuando es absolutamente necesario que la respuesta se produzca dentro del límite especificado. Por ejemplo: un ABS, el sistema en función de la velocidad, del ángulo de giro y del deseo del conductor de frenar, tomará la decisión correcta y todo ello en un tiempo determinado.
- Tiempo real no estricto (soft real time): Cuando se permite la pérdida ocasional de especificaciones temporales, aunque debe cumplirse normalmente. Ej.: sistema de adquisición de datos
- Tiempo real firme (firm real time): Cuando se permite la pérdida ocasional de especificaciones temporales, pero dicha perdida no es útil ya que la respuesta retrasada es descartada. Ej.: sistema multimedia.

3.3.- Sensores

Las tipologías de sensores se pueden dividir absolutos y de generación de datos. Los absolutos identifican unívocamente objetos en el entorno cuya posición puede ser estimada por los sensores. La generación de datos no puede, de antemano, asumir la identificación de los obstáculos, solo emitir datos sobre ellos que posteriormente pueden dar a conocer la posición.

Un ejemplo de sensores absolutos son el punto de acceso Wi-Fi (midiendo la intensidad de los diferentes puntos de acceso), sensores táctiles o las balizas de radio. Para algunas aplicaciones al aire libre los GPS (Global Positioning System) han dejado atrás a la mayoría de los sensores debido a su precisión.

Los sensores de adquisición de datos en su mayoría son ópticos. Los sensores ópticos pueden ser unidimensionales, 2D mediante barrido o 3D con LiDARs de alta definición, cámaras en 2D y 3D, aunque también existen los acústicos.

3.3.1- Sensores de distancia

ToF (time of fligth), esta técnica es utilizada por muchos sensores de distancia actuales, consiste básicamente en medir el tiempo que tarda un objeto, una partícula o una onda en atravesar un medio. Es muy utilizado con ondas electromagnéticas, con luz o con sonido, pues tienen velocidades conocidas de desplazamiento en la atmósfera. Una de las tipologías más usadas es que tanto el receptor como el emisor estén cercanos, emitir la señal y medir el tiempo que tarda en volver después de reflejarse en un obstáculo. Para hallar la distancia **d**

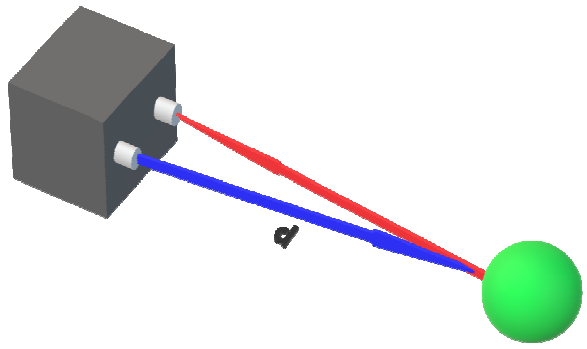


Figura 1

$$d = \frac{v \cdot t}{2}$$

Donde **t** es el tiempo entre la emisión y la recepción y **v** es la velocidad del objeto, partícula u onda. La condición de fiabilidad del sensor es que la velocidad no varíe para el medio de aplicación del sensor, o al menos que sea conocida.

LiDAR (**Li**ght **D**etection **a**nd **R**anging) es una tecnología que emplea la medición punto a punto para crear una nube de puntos que representa el entorno. Los sensores que dan pie a esta tecnología miden la distancia calculando el retraso de tiempo entre la emisión de un pulso laser del infrarrojo cercano, normalmente entre las longitudes de onda de 1040nm y 1065nm, y su recepción después de reflejarse en el objetivo (ToF). El retraso se transforma en distancia usando la conocida velocidad de la luz.

Figura 2

Aunque son los sensores más utilizados tienen diversos inconvenientes a tener en cuenta, sobre todo, en la detección de obstáculos que pueden causar falsos negativos.

Para que la señal emitida vuelva con una magnitud suficiente para que pueda ser procesada la superficie del obstáculo debe tener una reflexión difusa, es

decir, que la energía reflejada se dispersa uniformemente. El papel, mármol, madera son ejemplos de materiales con superficies difusas. Si la reflexión es especular, es decir, la energía reflejada se dispersa de forma concentrada, lo más probable es que el sensor no reciba de vuelta la señal. El metal pulido y los espejos son ejemplos de materiales con superficies especulares.

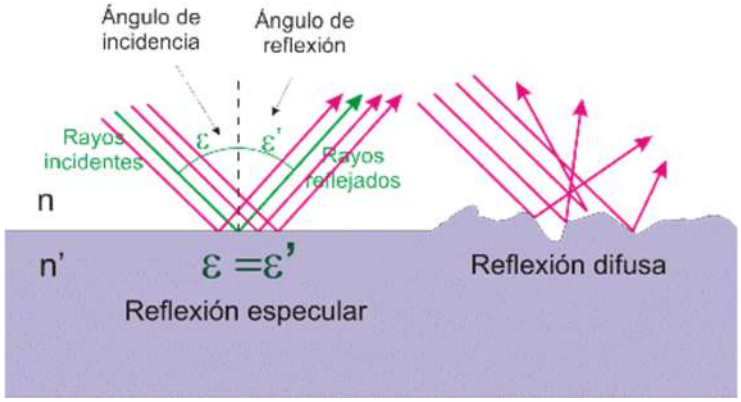


Figura 3

Otro fenómeno a tener en cuenta es la absorción, una superficie puede absorber la energía y reflejar una parte pequeña que sea insuficiente para que el receptor lea la señal. Un ejemplo claro en el espectro visual son las superficies negras o blancas.

Aunque la emisión se hace en un espectro cercano al visual, puede que una superficie que en nuestro espectro tenga unas propiedades en el espectro infrarrojo tenga otras, convirtiendo superficies aparentemente opacas en transparentes.

El ángulo de incidencia del haz sobre la superficie del obstáculo junto con las características ópticas de la superficie también interfiere en la medida de la señal.

La atenuación es otra de las causas que pueden perjudicar la medida, pues es el fenómeno con el que una señal pierde potencia al transitar por cualquier medio.

Esto afecta a la distancia máxima a la que es capaz de detectar un obstáculo el sensor.

El tamaño del obstáculo también afecta a la medida de la señal. Un obstáculo demasiado pequeño puede producir que el haz reflejado no tenga la potencia suficiente para volver al receptor. Esto es debido a que solo se ha reflejado parte del haz incidente, ya que éste es mucho más grande que el obstáculo a causa de la difusión, que es función de la distancia.

Los efectos de la atenuación y el tamaño de los obstáculos juntos son la causa de la pérdida de definición con la distancia.

Estos problemas pueden provocar deficiencias en la compresión de entornos que para nuestros ojos serian sencillos.

Los sensores de Ultrasonidos o sensores ultrasónicos son detectores de proximidad que trabajan libres de roces mecánicos y que detectan objetos a distancias que van desde pocos centímetros hasta varios metros. El sensor emite un sonido y mide el tiempo que la señal tarda en regresar (ToF). Éstos reflejan en un objeto, el sensor recibe el eco producido y lo convierte en señales eléctricas, las cuales son elaboradas en el aparato de valoración. Éstos sensores trabajan solamente donde tenemos presencia de aire (no pueden trabajar en el vacío, necesitan medio de propagación), y pueden detectar objetos con diferentes formas, diferentes colores, superficies y de diferentes materiales. Los materiales pueden ser sólidos, líquidos o polvorientos, sin embargo, han de ser deflectores de sonido.

Figura 4

Estos sensores también adolecen de gran parte de los problemas de los sensores ópticos. La reflexión, la absorción, la difusión y la atenuación afectan a las medidas. Aunque la tipología de las superficies que lo producen difiere de los sensores ópticos, por ejemplo, un trozo de lana será visible para el sensor óptico, pero no por el sensor ultrasónico debido a la absorción del sonido por parte de la lana.

Otra tipología distinta de sensor a distancia son las cámaras. Con un sistema de cámaras se pueden adquirir imágenes y mediante el procesamiento de la imagen generar una reproducción virtual del mismo. Los sistemas con cámaras pueden tener solo una cámara, pero entonces deben ir acompañados de algún sistema de posicionamiento adicional. Existen diversas técnicas innovadoras para generar los mapas del entorno a través de las imágenes. Se basan en superponer los datos de imágenes tomadas desde diferentes puntos y comparar los mismos elementos en las diferentes imágenes y su distancia hasta el observador y, una vez identificado el obstáculo, generar el mapa del entorno.

Estos sistemas también pueden ir acompañados de "Machine Learning" para aportar información adicional sobre los elementos identificados.

3.3.2.- Sensores de posición

El GPS es el sensor de posición por excelencia, es un sistema que permite determinar en toda la Tierra la posición de cualquier objeto con una precisión de unos pocos metros. El GPS funciona mediante una red de, como mínimo,

24 satélites en órbita sobre el planeta Tierra, a aproximadamente 17.700 km de altura, con órbitas distribuidas para que en todo momento haya al menos 4 satélites visibles en cualquier punto de la tierra. Cuando se desea determinar la posición tridimensional, el receptor que se utiliza para ello localiza automáticamente, como mínimo, cuatro satélites de la red, de los que recibe unas señales indicando la identificación y hora del reloj de cada uno de ellos, además de información sobre la constelación. Con base en estas señales, el aparato sincroniza su propio reloj con el tiempo del sistema GPS y calcula el tiempo que tardan en llegar las señales al equipo, y de tal modo mide la distancia al satélite.

Figura 5

El principal inconveniente de este sistema es la dependencia de la cobertura de los satélites, en interiores no funciona correctamente.

3.4.- Sistemas de control

Para poder llevar a cabo el SLAM es necesaria una unidad que reciba las señales de los sensores, procese la información y tome decisiones sobre sus funciones. Para ello existen una gran variedad de sistemas, desde potentes ordenadores a microcontroladores de 8 bits.

3.4.1- Microcontrolador

Un microcontrolador (abreviado μC o MCU) es un circuito integrado programable, capaz de ejecutar las órdenes grabadas en su memoria. Está

compuesto de varios bloques funcionales que cumplen una tarea específica. Un microcontrolador incluye en su interior las tres principales unidades funcionales de una computadora: unidad central de procesamiento, memoria y periféricos de entrada/salida.

Existen muchos tipos de microcontroladores, la gran mayoría se diseñan para usos específicos, aunque hoy en día es fácil encontrar microcontroladores para uso general como AVR Atmel-X (Arduino) o el PIC (Microchip).

Figura 6

El uso de μC en procesos de desarrollo está orientado a la gestión del hardware.

3.4.2- Ordenadores de placa reducida

Existen alternativas de capacidades superiores a los microcontroladores, pero menores que un ordenador convencional. Los ordenadores de placa reducida como la Raspberry Pi o la Beagle Bone tienen todos los componentes de un ordenador convencional, pero con capacidad limitada.

Una de las principales diferencias con los μC, además de sus capacidades de cómputo, es que tienen la capacidad de incorporar un sistema operativo, que permite una capa de abstracción sobre el hardware facilitando el desarrollo de aplicaciones más complejas.

Figura 7

El uso de ordenadores de placa reducida en procesos de desarrollo está orientado a la gestión del software con demandas bajas e interacción con el hardware.

3.5.- Ejemplos de sistemas autónomos

Actualmente podemos encontrar diferentes ejemplos comerciales de sistemas autónomos que se desplazan.

Recientemente se está extendiendo, sobre todo en nuestro país, la comercialización de robots aspiradores domésticos. Estos robots utilizan diversas tecnologías y combinaciones de ellas para interactuar con entornos complicados, por la cantidad de obstáculos, como puede ser una vivienda.

Entre ellas son dignas de mención cinco tecnologías:

- LiDAR: Estos robots llevan una torreta donde un LiDAR realiza barridos para generar una nube de puntos en un plano y generando un mapa y situándose en él por las distancias a los puntos conocidas.
- Cámaras: Estos robots utilizan la adquisición de imágenes junto con un sistema de mapas de ocupación de celdillas para asignar imágenes a cada una de las celdillas para ser capaces de posicionarse en el entorno. Los modelos más avanzados empiezan a añadir cierta capacidad de reconocimiento de imágenes para identificar obstáculos comunes.

- Infrarrojos: Estos robots utilizan los infrarrojos junto con un sistema de mapas de ocupación de celdillas para detectar dónde están los obstáculos y los límites del entorno y marcar qué celdillas están libres y cuáles no. A partir de ahí, de forma aproximada y conociendo de antemano la celdilla de partida son capaces de localizarse en el mapa creado.

- Sensor de contacto: Es un sistema muy parecido al anterior, pero utilizan un sistema más "primitivo" que consiste en un pulsador que se activa al chocar con un obstáculo. Aunque hay muy pocos robots en el mercado que se basen únicamente esta tecnología para realizar el mapa del entorno y localizarse en él. Está tecnología suele incorporarse a todas las demás para evitar daños en el robot y solventar los errores que se puedan cometer en el posicionamiento.

Figura 8

Otro ejemplo de sistema autónomo son los coches autónomos. Estos sistemas se hallan ante unos entornos mucho más complicados que los anteriores y con un nivel de incertidumbre y de peligrosidad elevados. Las tecnologías que suelen

acompañar a estos sistemas para resolver el problema del posicionamiento y percepción del entorno son:

- GPS: Un sistema GPS permite situarse en cualquier parte del planeta con cobertura de satélites. Junto con un sistema de mapas preestablecido es capaz de saber en qué parte del mapa está y cómo es su entorno cercano.
- Sistemas de cámaras: Estos sistemas utilizan IAs (inteligencias artificiales) basadas en "Machine Learning" para entender el entorno y situarse en él.
- LiDAR: También utilizan sensores LiDAR para poder estimar con exactitud las distancias con los obstáculos del entorno.
- Sensores térmicos: Utilizados para la detección de peatones o seres vivos en el entorno.

Figura 9

Para finalizar, un ejemplo de un sistema autónomo en el mundo de la aviación son los pilotos automáticos que acompañan a todos los aviones. Aunque desconocido por muchos usuarios muchas veces los pilotos automáticos realizan el vuelo completo, incluyendo el aterrizaje, sin la intervención del piloto. Estos los pilotos automáticos usan sistemas propios de su entorno:

- Equipo telemétrico: (DME, del inglés: Distance Measuring Equipment) es un sistema electrónico que permite establecer la distancia entre éste y una estación emisora con una posición conocida. Este sistema de radiofrecuencia se utiliza para ayudar al piloto o al piloto automático a situar el avión respecto a la pista.
- Barómetro: Es un sensor de presión que permite conocer la altura a la que se encuentra el avión.
- Giroscopio: Es un sensor que permite conocer la velocidad angular del aparato respecto de sus ejes.
- Brújula: Es un sensor que permite conocer la orientación del avión respecto al polo magnético del planeta.
- GPS: Permite conocer la posición del avión en el planeta.
- Radar: Es un sensor ToF que permite reconocer el entorno mediante la emisión y recepción de ondas electromagnéticas.

Figura 10

3.6.- Uso de drones

Un vehículo aéreo no tripulado (VANT), UAS (Unmanned Aerial System), comúnmente conocido como dron, es un vehículo sin tripulación, reutilizable, capaz de mantener de manera autónoma un nivel de vuelo controlado y sostenido, y propulsado por un motor de explosión, eléctrico o de reacción.

El diseño de los drones tiene una amplia variedad de formas, tamaños, configuraciones y características. Existen dos variantes a nivel de control: los controlados desde una ubicación remota (RPAS), y aquellos de vuelo autónomo en los que se preprograman planes de vuelo y vuelan usando automatización dinámica.

3.6.1.- Ala fija

Los aviones, ala deltas y planeadores son los más conocidos de este grupo. Estas naves se caracterizan por depender de su velocidad y la forma de sus alas para desplazar volúmenes de aire que permitan su vuelo. Estas naves, permiten vuelos más largos a velocidades constantes gracias a su eficiencia aerodinámica. Sus altas velocidades hacen de esta clase de naves una herramienta muy útil para el reconocimiento de grandes extensiones de terreno, como pueden ser estudio del avance de incendios y realización de mapas cartográficos.

Figura 11

3.6.2.- Ala rotatoria

Los helicópteros, multicópteros y autogiros, entre otros, forman parte de este grupo. Llevan a cabo su vuelo gracias a la sustentación de las hélices de los rotores. Estas aeronaves poseen una autonomía muy limitada; sin embargo, poseen bondades únicas: despegue vertical, vuelo estacionario, gran

maniobrabilidad y versatilidad en sus aplicaciones. Es una nave ideal para la toma de imágenes nítidas, escaneado preciso, búsqueda detallada o vuelo en entornos difíciles.

Figura 12

3.6.3.- Multirrotores

Son una variedad de naves de ala rotatoria, pero tienen suficientes diferencias de las anteriores para definir un tipo. El elegir más o menos motores, influirá enormemente en las características del multicóptero, cambiando la estructura, el tipo de vuelo, el coste, el control, el peso y el consumo. El disponer de múltiples motores también facilita una mayor tolerancia a fallos del sistema.

4.-Requisitos y consideraciones

Una vez conocidos los elementos actuales de la tecnología podemos abordar las siguientes consideraciones para el proyecto.

La elección del sensor resultara la parte fundamental del proyecto y por ello le dedicaremos más recursos, para poder diseñar un sistema robusto y fiable para la obtención de datos.

Para la elección de los sensores nos basaremos en las siguientes premisas:

- Debemos escoger sensores ligeros, con bajo consumo y de fácil acceso.
- Usaremos sensores que detecten la distancia de punto a punto. Son más fáciles de conseguir y más sencillos de implementar.
- Usaremos dos tipos de sensores ópticos y un sensor acústico.
    - Usaremos un LiDAR
    - Usaremos un sensor infrarrojo
    - Usaremos un sensor de ultrasonidos
- No usaremos sensores de video, puesto que deben ir acompañados con técnicas avanzadas de reconocimiento de imágenes.

Una vez claro qué sensor nos permitirá un mejor rendimiento, montaremos un dron con capacidad para utilizarlo. Para la elección del hardware del dron:

- Como vamos a tener que montar un dron con autopiloto, utilizaremos Ardupilot, que es un hardware especializado de código abierto.
- Realizaremos los ensayos en espacios cerrados, por lo que nuestra dependencia de la señal GPS no debe ser obligatoria.
- Usaremos una Raspberry Pi por su capacidad de cómputo, pero instalaremos una versión del sistema operativo en tiempo real. En el

mercado podemos encontrar un "Shield" de Raspberry Pi con una batería de sensores diseñados para drones, llamado navio2, y lo acompaña con una versión de Raspbian (sistema operativo oficial de Raspberry Pi) modificada para ser un sistema operativo en tiempo real, el Emlid Raspbian.

Para la programación del firmware de los componentes en Raspberry Pi requeriremos de la librería pigpio.py, que nos permite gestionar las GPIOs por hardware en vez de por software, que es el estándar, lo que nos permite emplear interrupciones en los pines de entrada/salida y de esta manera, por ejemplo, tener un control más preciso de las señales PWM o monitorizar sin bloquear el programa y tener que recurrir a la multitarea.

Para ayudarnos en la selección del sensor y para la posterior interpretación de los datos, diseñaremos una interfaz con las siguientes premisas:

- Entre las librerías graficas libres para Python podemos encontrar vPython, que nos permitirá, de forma fácil, representar en 3D los datos obtenidos.
- Para poder usar la nube de puntos fuera del dron dotaremos de una conexión Wi-Fi con protocolo UDP para la comunicación. Para realizar la conexión Wi-Fi usaremos el hardware embebido que hay en la Raspberry Pi 3 B+

Una vez claras las partes solo resta la integración final, para la que consideraremos:

Que nuestra propuesta de diseño tiene que permitir una posterior interpretación de los datos para que en un futuro sea sencillo resolver el SLAM.

5.- Materiales y métodos

Conociendo los requisitos a seguir para realizar el proyecto, nos respaldaremos en los siguientes materiales, programas y métodos.

Raspberry Pi

- Raspberry Pi 3 B+

Sensores

- LiDAR Little v3 de Garmin
- Ultrasonidos HC - SR04
- Infrarrojos SHARP GP2Y0A710

Motores

- Servomotores SG90 (muy utilizados en aeromodelismo)
- Motor paso a paso

Dron

- Chasis
- Motores BE1806-2300KV, cuatro unidades
- ESC DYS XSC20A, cuatro unidades
- Emisora FlySky FS-i6
- Receptora Flysky FS-iA6B
- Autopiloto navio2

Python

- Selección de la librería grafica para la representación visual de los datos obtenidos
- Implementación comunicación i2c
- Implementación comunicación wifi (UDP), tanto cliente como servidor
- Implementación IA básica
- Implementación interfaz grafica
- Drone Kit basado en Python 2

Piezas

- SolidWorks
- Repetier, usando el CuraEngine para realizar el "slicer"
- Impresora 3D

6.- Procedimiento

A continuación se redactan los pasos a seguir para la consecución de los objetivos de este proyecto, cuyo detalle se amplía en capítulos posteriores.

El proyecto se puede dividir en dos partes, en la primera parte nos centraremos en la selección del sensor y en la segunda implementaremos el sensor en un dron.

1. Selección del sensor: Buscaremos el mejor sensor que se adapte a nuestros requisitos. Para eso necesitaremos realizar los siguientes puntos.
    a) Estudiar las fichas técnicas de los sensores más comunes del mercado.
    b) Crear sistemas tanto mecánicos, como electrónicos y de software para ensayar los sensores, que serán:
        i) LiDAR, para su uso será necesario:
            (1) Crear la pieza de soporte para el LiDAR usando SolidWorks y la impresora 3D.
            (2) Programar una librería para interaccionar con el LiDAR Little v3 y la Raspberry Pi usando el i2c para la comunicación.
        ii) Ultrasonidos, para su uso será necesario:
            (1) Crear la pieza de soporte para el Ultrasonidos usando SolidWorks y la impresora 3D.
            (2) Programar una librería para interaccionar con el sensor de ultrasonidos HC - SR04 y la Raspberry Pi usando la interfaz propia del sensor.
        iii) Infrarrojos, para su uso será necesario:

(1) Crear la pieza de soporte para el Infrarrojos usando SolidWorks y la impresora 3D.

(2) Programar una librería para interaccionar con el conversor analógico digital (ADC) y la Raspberry Pi usando i2c. El sensor irá conectado al ADC.

iv) Servomotores, para su uso será necesario:

(1) Crear las piezas de soporte para construir un sistema que permita realizar barridos con los sensores usando SolidWorks y la impresora 3D.

(2) Programar una librería para interaccionar con los servomotores y la Raspberry Pi, usando el protocolo propio de los servomotores basado en PWM (Pulse Width Modulation).

v) Motores paso a paso, para su uso será necesario:

(1) Crear las piezas de soporte para construir un sistema que permita realizar barridos con los sensores usando SolidWorks y la impresora 3D.

(2) Programar una librería para interaccionar con los drivers de los motores paso a paso y la Raspberry Pi usando el protocolo propio.

vi) Para hacer más flexibles los ensayos de sensores y aprovechando que posteriormente habrá que utilizarlo, crearemos la interfaz de comunicación Wi-Fi entre la Raspberry Pi y un PC. Para ello se necesitara:

(1) Configurar la Raspberry Pi como punto de acceso, para que cualquier PC se pueda conectar

(2) Programar del servidor que irá alojado en la Raspberry Pi

(3) Programar del cliente que irá alojado en el PC

(4) Programar la interfaz gráfica para representar gráficamente las nubes de puntos por parte del cliente alojado en un PC.

c) Una vez realizadas todas las implementaciones, se abordara el ensayo de los sensores para comprobar sus resultados.

d) Con los ensayos, se compararan los resultados y escogeremos qué sensor montar en el dron.

2. Dimensionar el dron: Una vez hayamos escogido la mejor opción para el sensor, diseñaremos un dron para que pueda albergarlo.

a) Se dimensionarán y elegirán los componentes.

i) Crearemos las piezas para ensamblar el sensor en el dron.

ii) Con todos los pesos seleccionaremos los motores.

iii) Con los motores seleccionaremos los controladores de velocidad (ESC).

iv) Con los ESC seleccionaremos la batería.

b) Se crearan y montaran los componentes.

i) Diseñaremos una placa para la distribución de la alimentación en el dron.

ii) Fabricaremos la placa prototipo para la distribución de la alimentación.

iii) Conectaremos todo el sistema de impulsión del dron y la alimentación de las partes.

iv) Conectaremos la estación de tierra y el dron a través de Wi-Fi.

Una vez mencionados todos los pasos a realizar para cumplir los objetivos vamos a proceder a la realización de los pasos mencionados.

7.- Selección del sensor

Una vez determinado el procedimiento a seguir para la realización del proyecto, pasamos a la parte más importante, la selección del sensor.

7.1.- Estudio de especificaciones

Debido a que los sensores solo miden la distancia a un punto, para poder generar la nube de puntos necesitamos girar el sensor para realizar un barrido.

Una de las de las características críticas a tener en cuenta es cuánto tiempo necesita el sensor para realizar una medida, este tiempo determinará el tiempo o la resolución del barrido.

- Láser: 3ms tiempo de ciclo.
- Ultrasonidos: 30ms tiempo de ciclo.
- Infrarrojos: 25ms tiempo de ciclo.

Donde cada ciclo es el tiempo entre dos medidas consecutivas.

Otra consideración para tener en cuenta es el consumo activo de los sensores. Puesto que el consumo en un dron es importante tendremos en cuenta que:

- Láser: 135mA
- Ultrasonidos: 15mA
- Infrarrojos: 50mA

También hay que tener en cuenta el peso del sensor y el peso de la instrumentación si la necesita:

- Láser: 22g

- Ultrasonidos: 9g
- Infrarrojos: 11g

Una de las características más importantes es el rango de distancia:

- Láser: 0,02m a 40m
- Ultrasonidos: 0,02m a 2m
- Infrarrojos: 0,1m a 1m

Estos sensores solo leen la distancia en un único punto, para poder utilizarlos para adquirir datos sobre el entorno necesitaremos desplazar el punto de lectura mediante algún tipo de motor.

Vamos a ver diferentes tipos de motor para considerar su inclusión en el sistema de posicionamiento.

Hay que analizar las diferentes aproximaciones para realizar barridos que aprovechen sus ventajas e inconvenientes. Podemos embarcar el sensor encima de una estructura giratoria motorizada (llamada gimbal) utilizando:

- Servomotores:

Es la solución más sencilla de implementar, pero presenta diversos problemas. En primer lugar, en el momento de tomar la medida después de habernos desplazado no podemos estar seguros de si hemos llegado o no a la posición de destino, es decir, el tiempo desde que enviamos la posición hasta que el servomotor llega a esa posición depende del controlador del servomotor, y es variable, con lo que tendremos que dar un margen de tiempo entre enviar la posición y medir. Esto, entre otras cosas, lo convierte en el sistema más lento.

En segundo lugar, con el paso del tiempo, los engranajes se desgastan y el sistema se vuelve inestable para mantener una posición, lo que añadirá ruido a las medidas.

Otro inconveniente es que la mayoría de los servomotores tienen solo un semicírculo de zona útil, es decir no pueden dar vueltas completas, con lo que tendríamos solo radianes de nube de puntos.

- Motores paso a paso:

Más rápido que un servomotor, pero tiene el inconveniente de que no podemos saber si nos hemos saltado algún paso, deformando la nube de puntos.

Debido a los imanes de cada paso para un motor equivalente a los demás, ésta es la solución más pesada, lo cual peyora su elección para instalarla en un dron donde el peso de los componentes es importante.

- Motor y encoder:

Esta solución es la que permite realizar unos barridos más rápidos y permite saber en todo momento el ángulo de la medida.

Con las soluciones anteriores la posición era estimada en función del paso o de la posición enviada. En este caso podemos girar el motor a una velocidad constante y realizar medidas constantemente.

Tiene el inconveniente de que necesita un control muy preciso de la posición, que permita parar cada 360º para volver sobre sí mismo, para no cizallar los cables del sensor montado, o implementar un sistema inalámbrico para que el sensor se alimente y envíe los datos, por ejemplo, por inducción se puede alimentar y con un fotodiodo y un led crear un protocolo de comunicación para enviar los datos.

Este sistema no es compatible con el sensor de ultrasonidos porque el tiempo que tarda el sonido en recorrer la distancia entre el sensor y el objetivo no es despreciable frente a la velocidad angular. En función de la velocidad angular, el sensor estaría en una posición distinta que impediría al receptor leer la señal de vuelta.

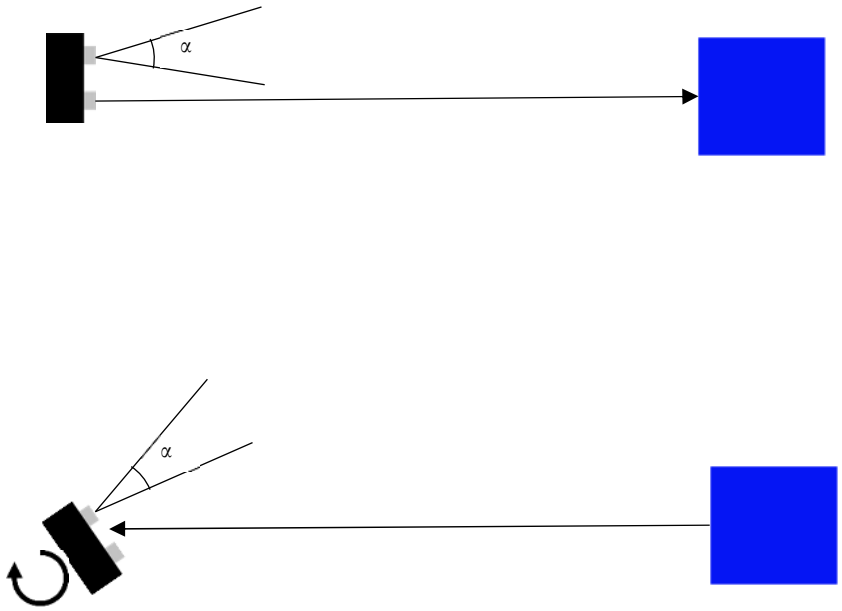


Figura 13

Ejemplo:

<u>Ultrasonido.</u>

A una velocidad angular de ω = 3 rad/s (aproximadamente media vuelta por segundo), sabiendo que la velocidad del sonido es v = 343.3m/s, frente a un obstáculo a d = 3m y con un ángulo máximo de recepción de α = 0,012 rad (este ángulo puede ser aproximado por la mitad de la divergencia).

Para que haya una lectura correcta de la señal enviada, el tiempo que tarda en ir y volver la señal t_s debe ser menor que el tiempo en girar el ángulo máximo de recepción t_g.

$$t_s = \frac{2 \cdot d}{v} = 0{,}0175s$$

$$t_g = \frac{\omega}{v} = 0{,}00412s$$

$$t_s \nless t_g$$

La velocidad máxima a partir de la que esto no ocurriría seria:

$$\omega < \frac{\alpha \cdot v}{2d} = 0{,}687rad/s$$

Aproximadamente una vuelta cada 10 segundos.

Láser.

A una velocidad angular de ω = 3 rad/s (aproximadamente media vuelta por segundo), sabiendo que la velocidad del sonido es v = $3 \cdot 10^8$m/s, frente a un obstáculo a d = 3m y con un ángulo máximo de recepción de α = 0,05 rad.

$$t_s = \frac{2 \cdot d}{v} = 2 \cdot 10^{-8}s$$

$$t_g = \frac{\omega}{v} = 0{,}00412s$$

$$t_s < t_g$$

Aplicar un motor con un encoder al sensor de ultrasonidos no aporta ninguna ventaja.

Descartaremos el uso de un motor sin escobillas con encoder magnético para la lectura de la posición debido a que la complejidad de la implementación se escapa del objetivo de este proyecto.

7.2.- Montaje

Para el montaje de los ensayos implementaremos cada uno de los sensores y de los motores.

7.2.1.-Interfaz

La adquisición de datos será llevada a cabo por una Raspberry Pi, que actuara de punto de acceso Wi-Fi, permitiendo que cualquier ordenador se conecte a ella. A su vez crearemos un servidor mediante protocolo UDP para atender a cualquier cliente que se conecte y suministrarle los datos de los sensores.

Para mejorar la adquisición de datos usaremos una librería de interfaz gráfica de Python llamada vpython que nos permitirá visualizar los puntos en un entorno 3D. Esta librería tiene la virtud de funcionar en cualquier plataforma ya que usa el navegador predeterminado para visualizar una página offline, es decir creando un servidor local donde alojarla, y ejecutando una script de JavaScript en el servidor. Esta librería estará implementada en el cliente que ejecutará desde el PC. Sera el cliente quien le mande que tipo de sensor y motor que tiene conectada la Raspberry Pi en cada momento mediante la elección del programa.

Esta interfaz contendrá tres botones para realizar las tomas de datos:

- Connect: Al pulsar este botón se realiza la conexión UDP/IP a la IP estática del servidor. Si la conexión se realiza con éxito el texto del botón cambiara a “connected”.

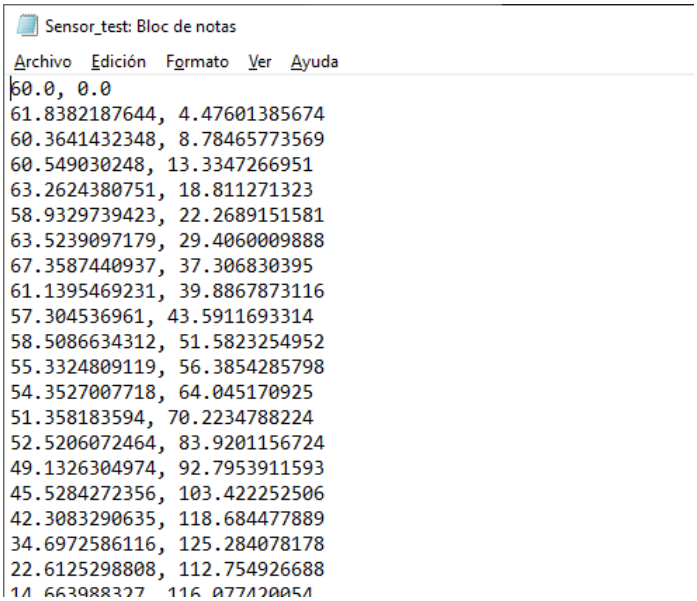
- Scan: Al pulsar este botón se realizara la obtención de puntos para el programa seleccionado.
- Close: Al pulsar este botón se cierra la conexión.

Para cambiar el programa enviado al servidor hay que cambiar el valor de la variable "program" en la script del cliente.

La parte fundamental de la interfaz será una ventana donde se mostraran los puntos obtenidos en tiempo real. Esta ventana mostrará los puntos en un espacio 3D, donde se podrá manipular la vista para una mejor percepción.

Figura 14

Además, el cliente, se encargara de convertir las lecturas de los sensores y los motores en coordenadas cartesianas para representarlas e imprimirlas en un archivo de texto llamado "Sensor_test.txt" para permitir posteriores tratamientos.


Sensor_test: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
60.0, 0.0
61.8382187644, 4.47601385674
60.3641432348, 8.78465773569
60.549030248, 13.3347266951
63.2624380751, 18.811271323
58.9329739423, 22.2689151581
63.5239097179, 29.4060009888
67.3587440937, 37.306830395
61.1395469231, 39.8867873116
57.304536961, 43.5911693314
58.5086634312, 51.5823254952
55.3324809119, 56.3854285798
54.3527007718, 64.045170925
51.358183594, 70.2234788224
52.5206072464, 83.9201156724
49.1326304974, 92.7953911593
45.5284272356, 103.422252506
42.3083290635, 118.684477889
34.6972586116, 125.284078178
22.6125298808, 112.754926688
14.663988327, 116.07742005
```

Figura 15

El código correspondiente al cliente (Radar_cliente.py) y al servidor (Radar_servidor.py) se pueden encontrar en el Anexo 1, así como la librería de la conexión sobre Wi-Fi (WIFI.py).

7.2.2.- LiDAR

Para implementar el LiDAR seguiremos el siguiente esquema:

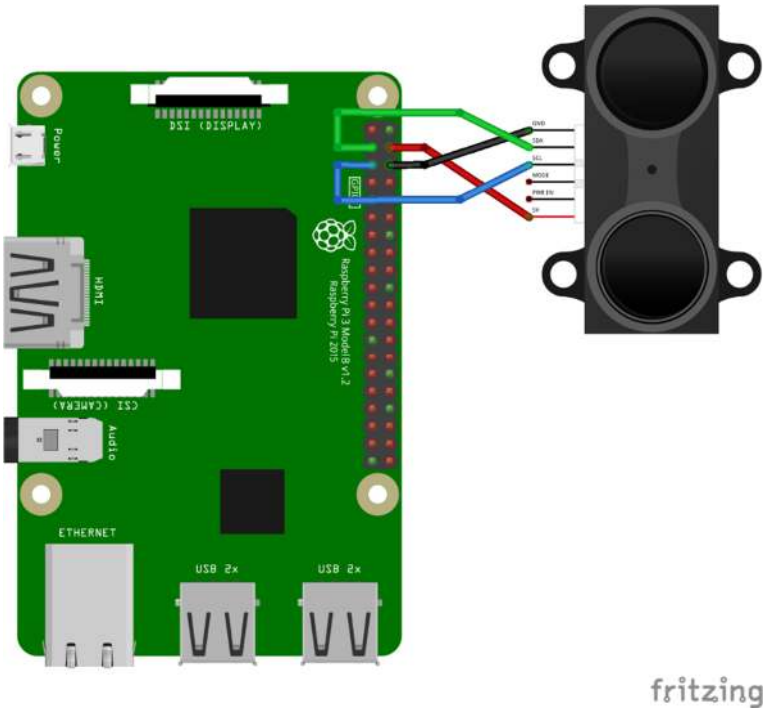


Figura 16

Como se puede observar en el esquema, conectamos directamente la línea de i2c a la Raspberry Pi, aunque el LiDAR 3v Lite se alimente a 5v y los puertos de la Raspberry funcionen a 3,3v no hay peligro, pues la Raspberry Pi será el "Master" y será quien controle el nivel de la línea SDA.

Usaremos la librería de LiDAR (lidar.py) en Python que se encuentra en el anexo 1. En ella se puede ver la lectura de datos a través del protocolo i2c.

7.2.3.- Ultrasonidos

Para implementar el sensor de Ultrasonidos seguiremos el siguiente esquema:

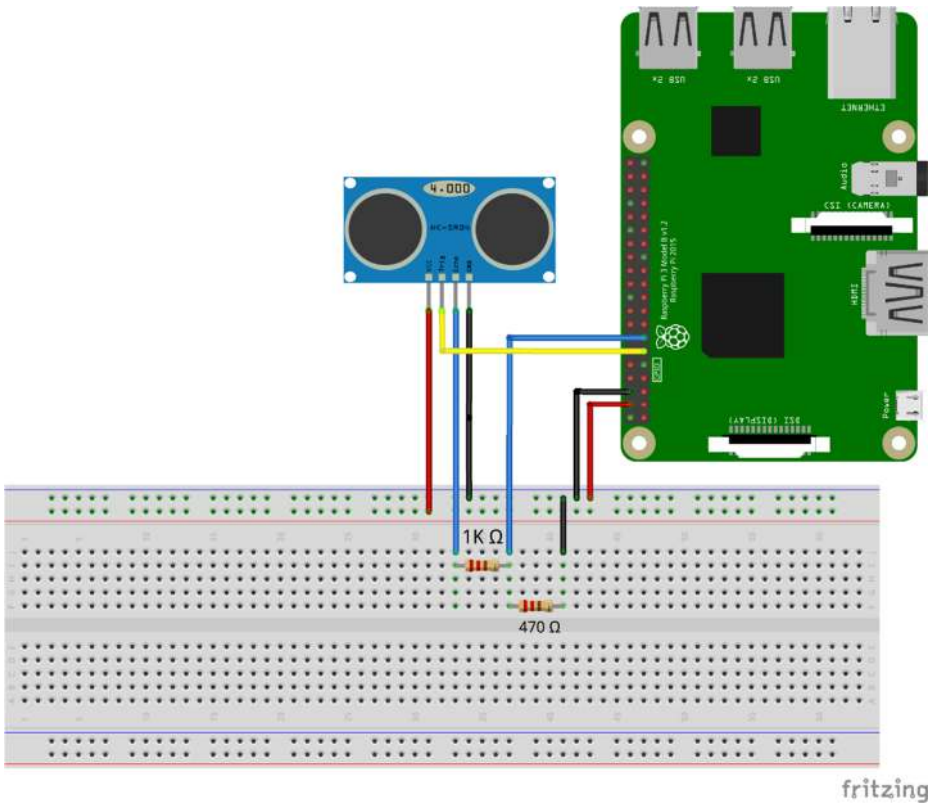


Figura 17

Como se puede observar, es necesario poner un divisor de tensión a la salida "echo" del sensor, pues el sensor trabaja a 5v y los puertos digitales de la Raspberry Pi trabajan a 3,3v y esto podría dañarla. La otra señal "trigger" es manejada por la Raspberry Pi y el sensor la interpreta correctamente.

Para el divisor de tensión hemos escogido las resistencias estándar de 1kΩ y 470Ω.

Usaremos la librería de Ultrasonidos (Ultrasonic.py) en Python que se encuentra en el anexo 1. En ella se puede ver cómo se lee el tiempo entre el disparo y la recepción del eco, todo ello usando las bondades de la librería de

pigpio que nos permite manipular las entradas/salidas digitales de la Raspberry Pi por hardware.

7.2.4.- Infrarrojo

Para implementar el sensor de Infrarrojos seguiremos el siguiente esquema:

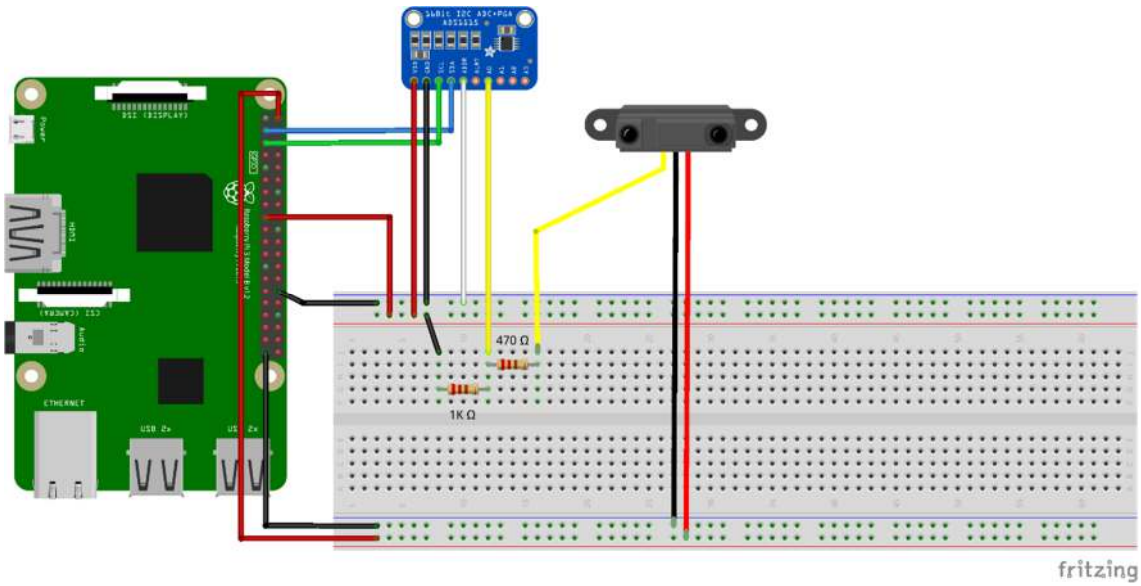


Figura 18

El sensor infrarrojos trabaja a 5v, aunque en las especificaciones marca que puede trabajar a 3,3v sus prestaciones disminuyen mucho. Como el conversor analógico digital necesario para poder leer la señal de 0v a 5v del sensor trabaja a 3,3v necesitaremos poner un divisor de tensiones. Usaremos una resistencia de 1kΩ y otra de 470 Ω. Conectaremos la línea del i2c a la Raspberry Pi. Para seleccionar la dirección del dispositivo, el ASD1115 permite escoger entre cuatro direcciones, en función de cómo se conecte el pin ADDR, en nuestro caso lo hemos conectado a GND, lo que lleva a tener la dirección 0x48.

El conversor es de 16 bits, es decir, el fondo de escala llega hasta 0xFFFF que se corresponde con 3,3v. La conversión es simple, solo hay que dividir el valor obtenido por 0xFFFF para hallar la tensión.

De la hoja de especificaciones técnicas del sensor GP2Y0A21YK podemos observar que el nivel de tensión para distancias mayores a los 20cm se puede aproximar por una función exponencial. De este modo:

$$distancia = a \cdot voltaje^{b}$$

Para hallar las constantes a y b que parametrizan la curva, basta con tomar dos puntos de ella. Para obtener una mejor aproximación usaremos dos puntos empíricos para realizar los cálculos.

70 cm corresponden a un valor de tensión de 0,14v

30 cm corresponden a un valor de tensión de 0,27v

Tras resolver el sistema de dos ecuaciones y dos incógnitas obtenemos:

$$distancia = 5.54 \cdot voltage^{-1.29}$$

Usaremos la librería del conversor analógico - digital (ADC_16bits.py) y la librería de Sensor infrarrojos (IR_Sensor.py) en Python que se encuentra en el anexo 1. En la primera podemos observar la lectura del conversor sobre i2c y la conversión de bytes a voltaje. En la segunda podemos ver la conversión de voltaje a distancia.

7.2.5.- Servomotor

Para implementar el servomotor seguiremos el siguiente esquema:

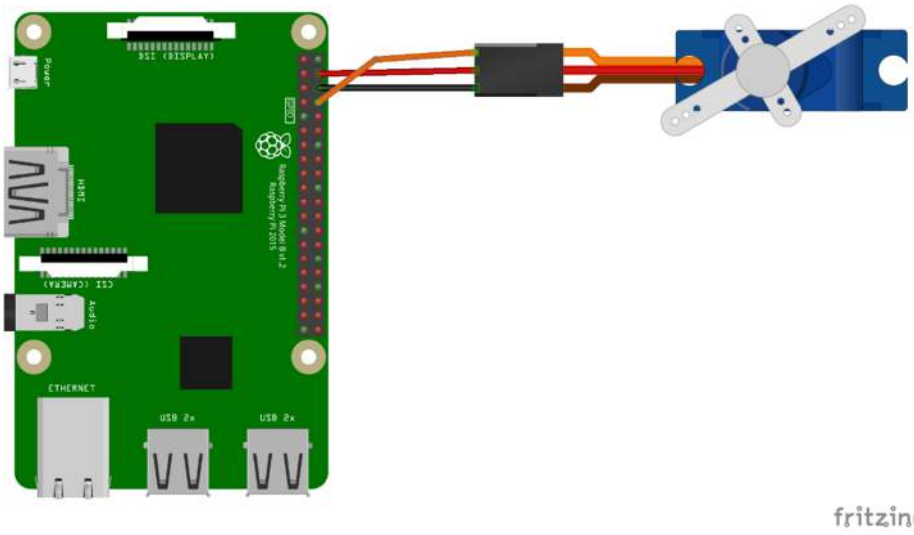


Figura 19

Debido a que el consumo del servomotor SG90 no supera el amperaje máximo de salida de la Raspberry Pi, se puede conectar directamente. En caso de conectar más servomotores deberíamos utilizar una alimentación externa.

Utilizaremos la librería de Servo Motor (Servo.py) que se puede encontrar en el anexo 1. En ella se puede observar cómo se genera una señal PWM por hardware utilizando la librería pigpio para controlar la posición del mismo. Debido a que no tenemos retroalimentación de la posición del servomotor esperaremos un tiempo antes de enviarlo a una nueva posición.

7.2.6.- Motor paso a paso

Para implementar el motor paso a paso seguiremos el siguiente esquema:

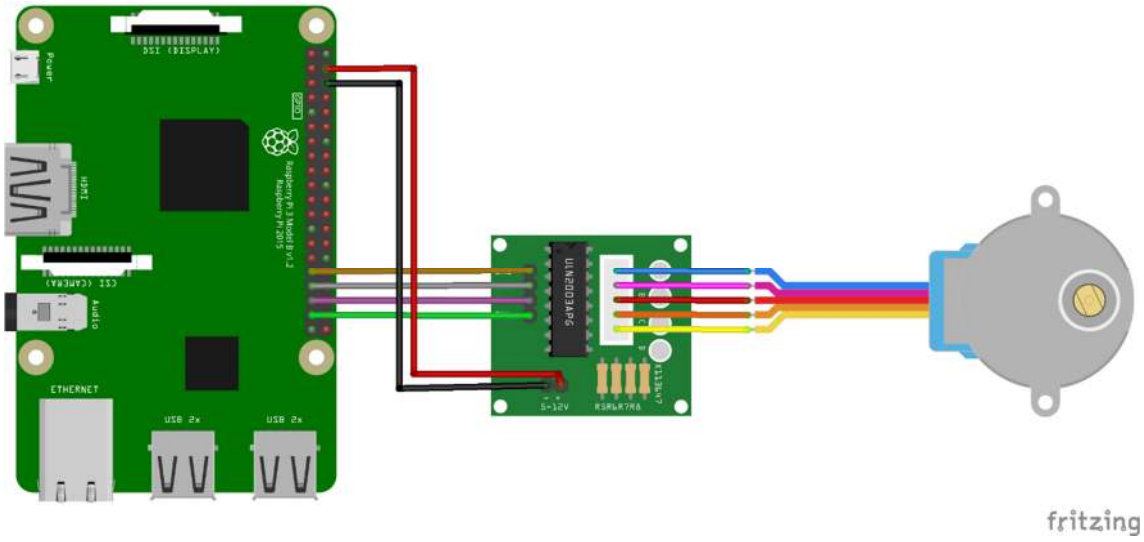


Figura 20

Al igual que el servomotor, el motor paso a paso 28BYJ-48 puede conectarse directamente a la Raspberry Pi. También conectaremos cada uno de los pasos a un pin para ir enviando cada combinación. Para alimentar los polos del motor usaremos un ULN2003 que nos permite secuenciar la alimentación de los polos en función de cada uno de los pasos que activemos.

Utilizaremos la librería Motor paso a paso (Step.py) que se puede encontrar en el anexo 1. En ella se puede observar cómo se sigue la secuencia de pasos. El tiempo entre pasos para esta configuración, para no saltarse ninguno, es de 0,8ms.

7.3.- Ensayos

Para realizar los ensayos hemos alimentado la Raspberry Pi usando una batería externa para poder mover todo el circuito a la zona de ensayos. La zona de ensayos consiste en un rincón cerrado con espacios abiertos en dos de los lados.

Es importante que la zona de ensayos sea un lugar conocido, para que todos los sensores actúen bajo las mismas condiciones y luego podamos comparar las nubes de puntos que obtengamos.

El sensor se situará siempre en la misma posición, marcado en rojo en la figura 21. Debido a la disposición inicial del sensor los ejes están situados tal y como se muestra en la misma figura.

Hemos incluido en los ensayos también una garrafa de agua para ver las diferencias de percepción que genera cada sensor al ser un objeto transparente. Los espacios abiertos permitirán ver la distancia máxima de cada sensor.

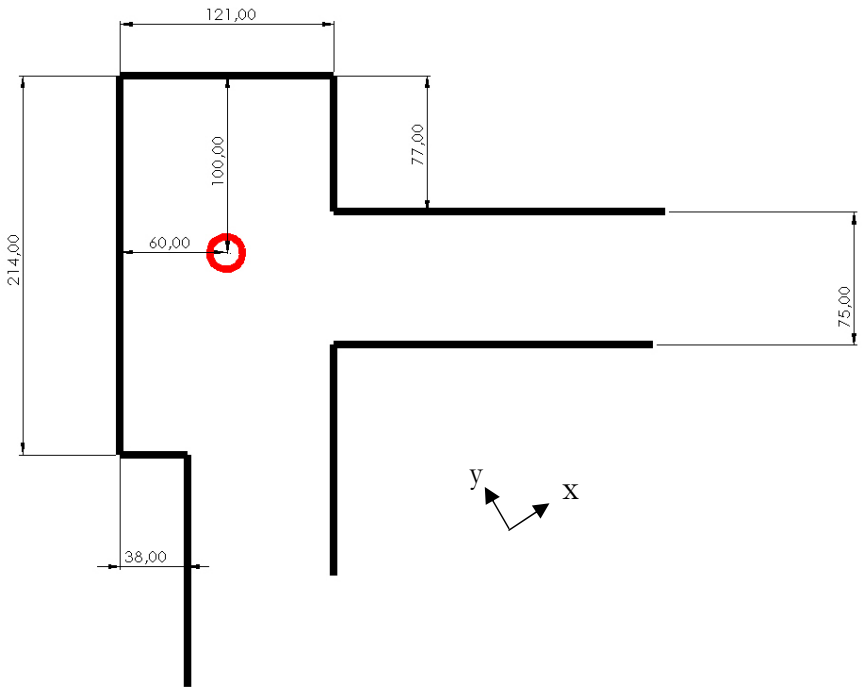


Figura 21

Figura 22

Figura 23

En la figura 22 podemos ver una fotografía del lugar que usaremos para los ensayos y la figura 23 muestra uno de los ensayos realizados.

Ensayaremos cada uno de los sensores con el motor paso a paso que permite una lectura de 360º, lo que permitirá tener más puntos con los que comparar cada ensayo. También ensayaremos el LiDAR 3v lite con el servomotor que solo recorre 180º.

Para obtener la nube de puntos usaremos el programa que se ha creado para obtener los puntos de la Raspberry Pi, conectándonos por Wi-Fi. De este modo los visualizaremos y, además, obtendremos un fichero de texto con las coordenadas de los puntos leídos en cada ensayo.

Los planos de las piezas usadas para ensamblar cada sensor y cada motor se encuentran en el anexo 2.

7.3.1.- LiDAR y motor paso a paso

En este ensayo usaremos el sensor LiDAR v3 lite y el motor paso a paso.

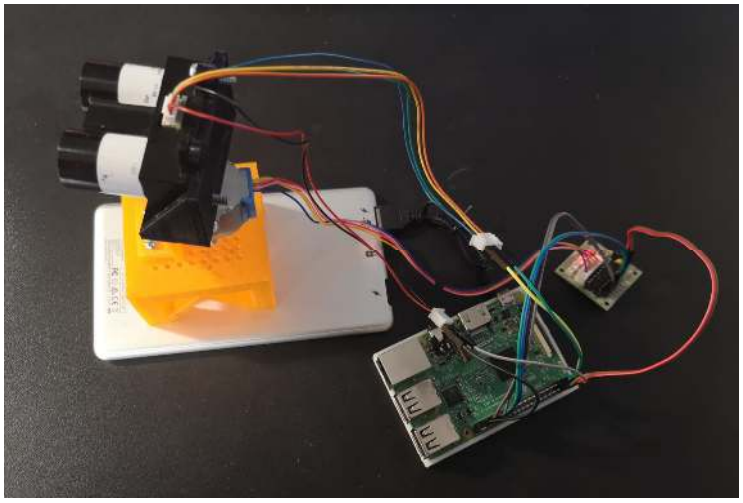


Figura 24

Los resultados han sido:

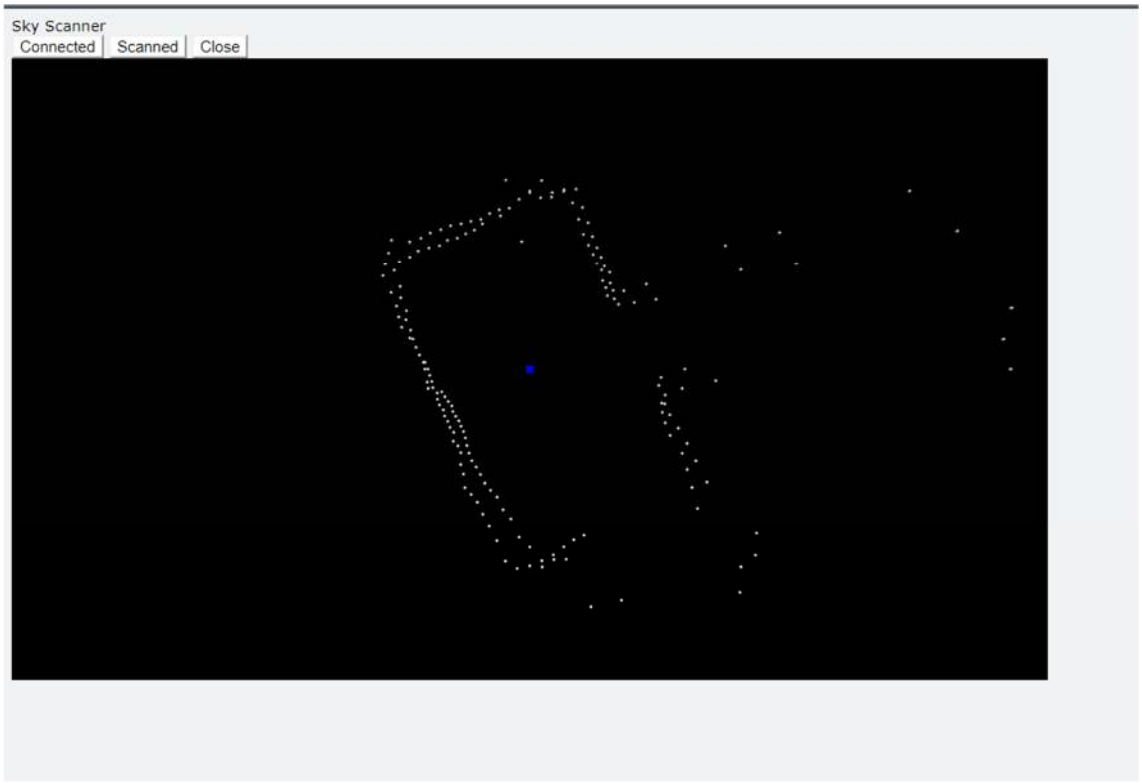


Figura 25

Podemos observar una gran definición del entorno, pero también se observa que la garrafa de agua, al ser transparente para el espectro infrarrojo, ha sido prácticamente ignorada.

7.3.2.- LiDAR y servomotor

En este ensayo usaremos el sensor LiDAR v3 lite y el servomotor.

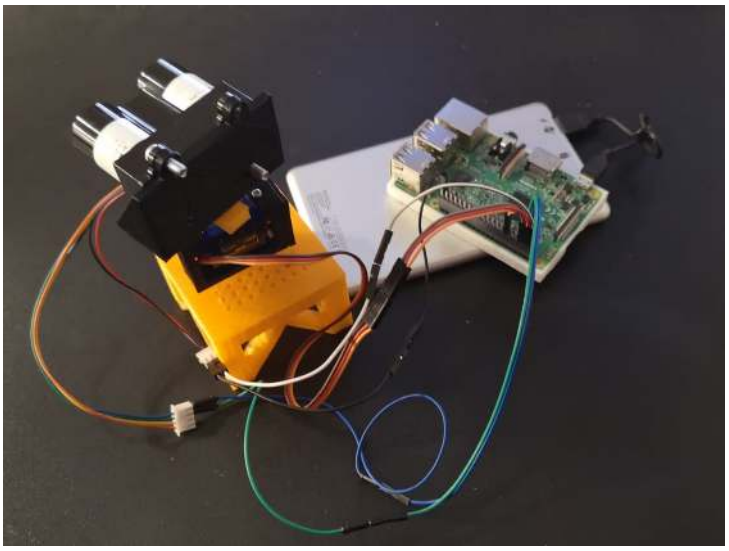


Figura 26

Los resultados han sido:

Figura 27

En este ensayo podemos observar un poco de pérdida de definición respecto al anterior ensayo, causado por las vibraciones del servomotor, además de la evidente lectura en solo 180º.

7.3.3.- Ultrasonidos y motor paso a paso

En este ensayo usaremos el sensor de ultrasonidos HCSR-04 y el motor paso a paso.

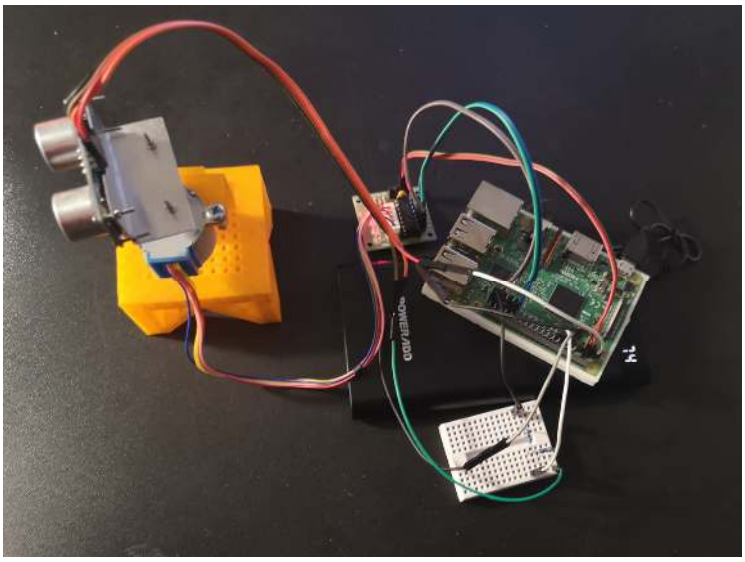


Figura 28

Los resultados han sido:

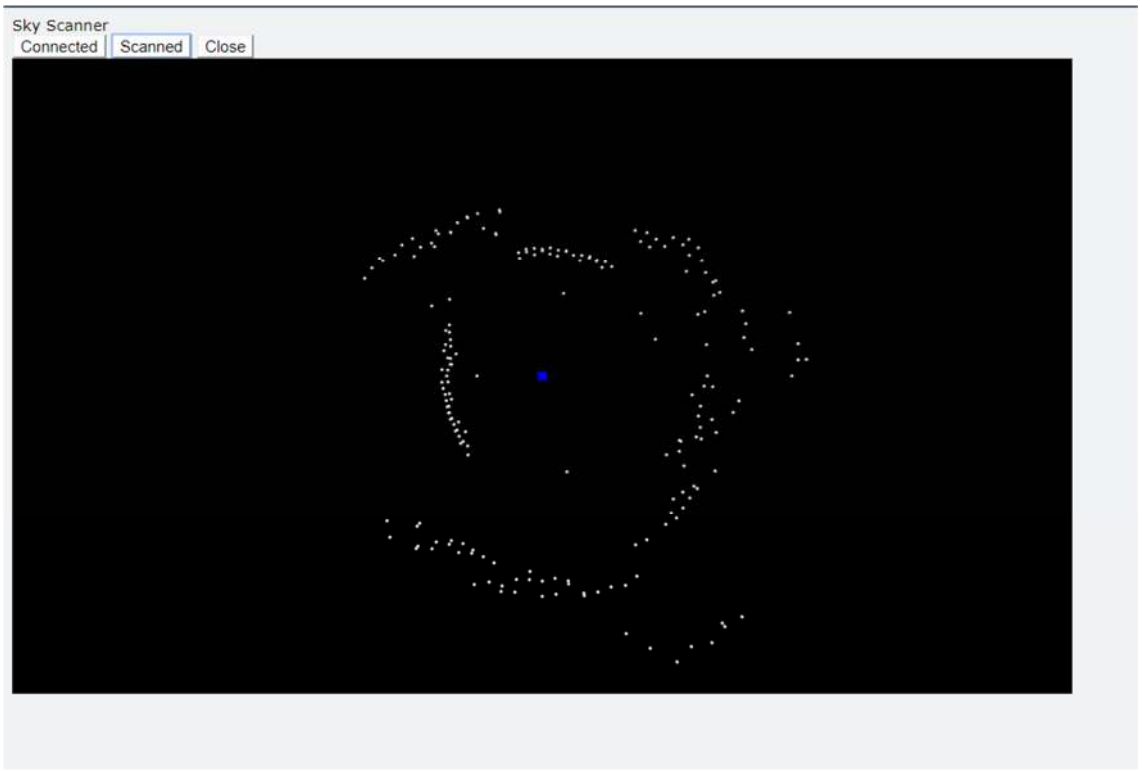


Figura 29

Podemos ver que es más difícil interpretar la nube de puntos. Se observa falta de definición debida a las esquinas, que producen la perdida de la señal debido

al ángulo de reflexión, además de la perdida de la medida en distancias superiores a 2 metros.

7.3.4.- Infrarrojo y motor paso a paso

En este ensayo usaremos el sensor Infrarrojos GP2Y0A21YK y el motor paso a paso.

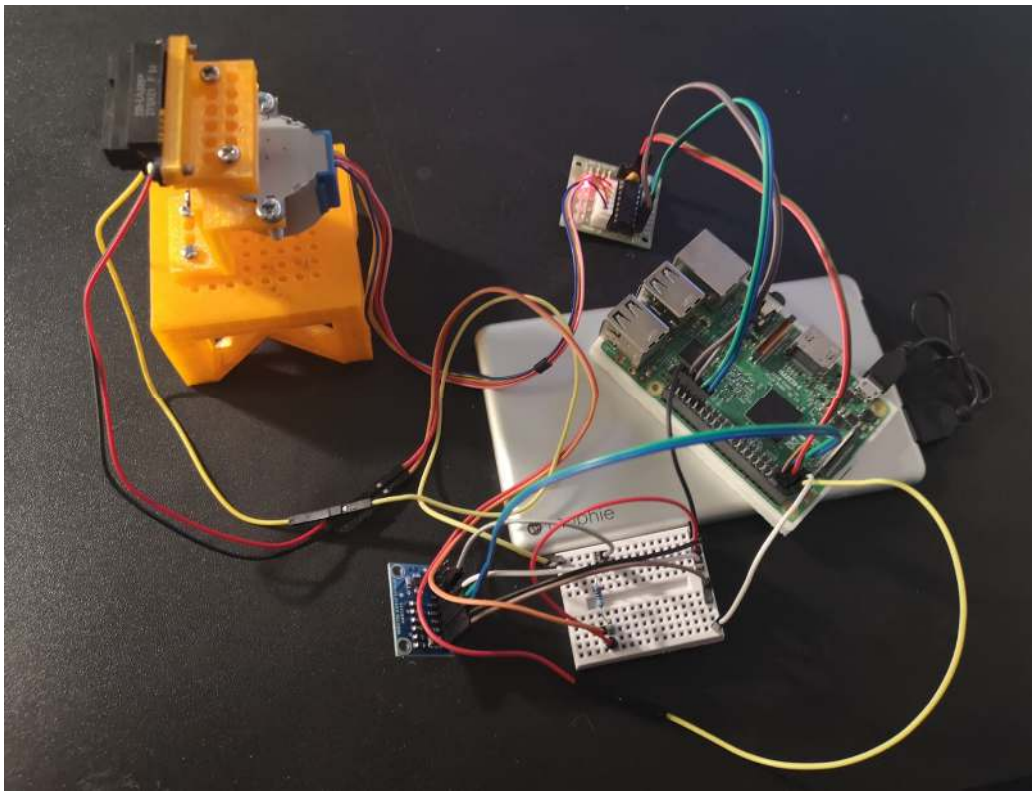


Figura 30

Los resultados han sido:

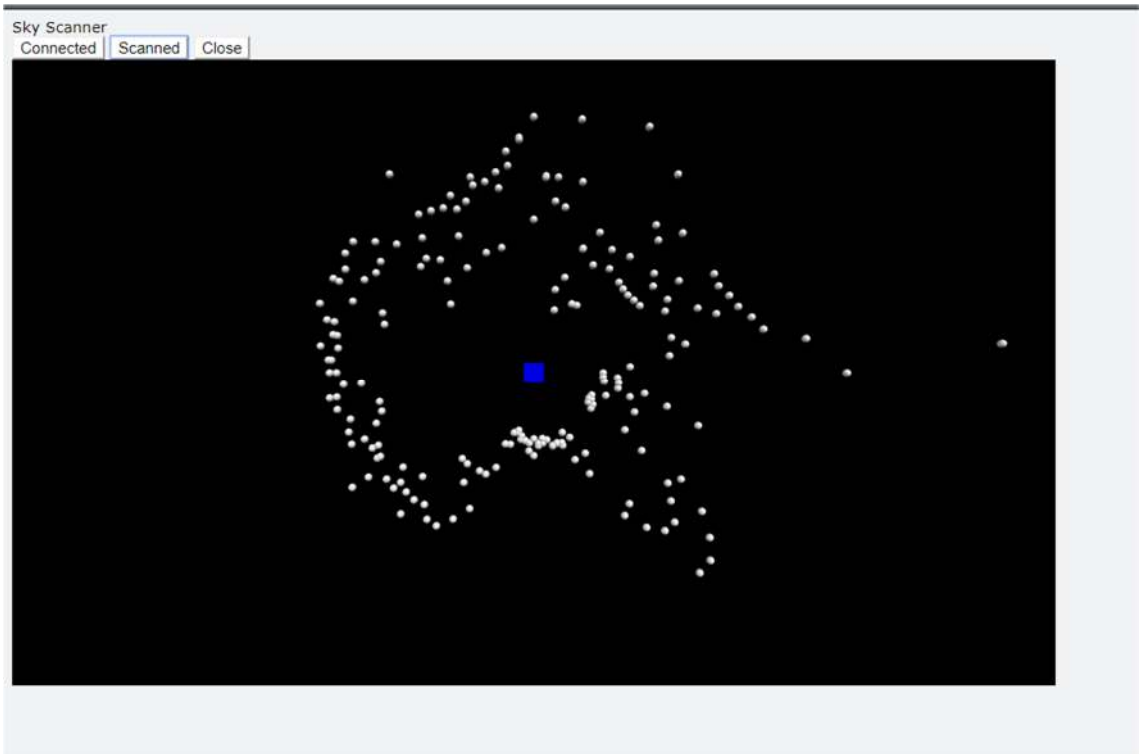


Figura 31

En este caso es difícil interpretar los datos debido a la gran cantidad de ruido que produce el sensor, además del limitado rango de medida de como máximo 1 metro. El comportamiento de este sensor para rangos superiores a su máximo sitúa los puntos cerca del rango mínimo, siendo incapaz de distinguir entre los obstáculos cercanos y cuando esta fuera de alcance, distorsionando las medidas.

7.4.- Resultados

Si superponemos los resultados obtenidos en los ensayos obtenemos la representación de la figura 32.

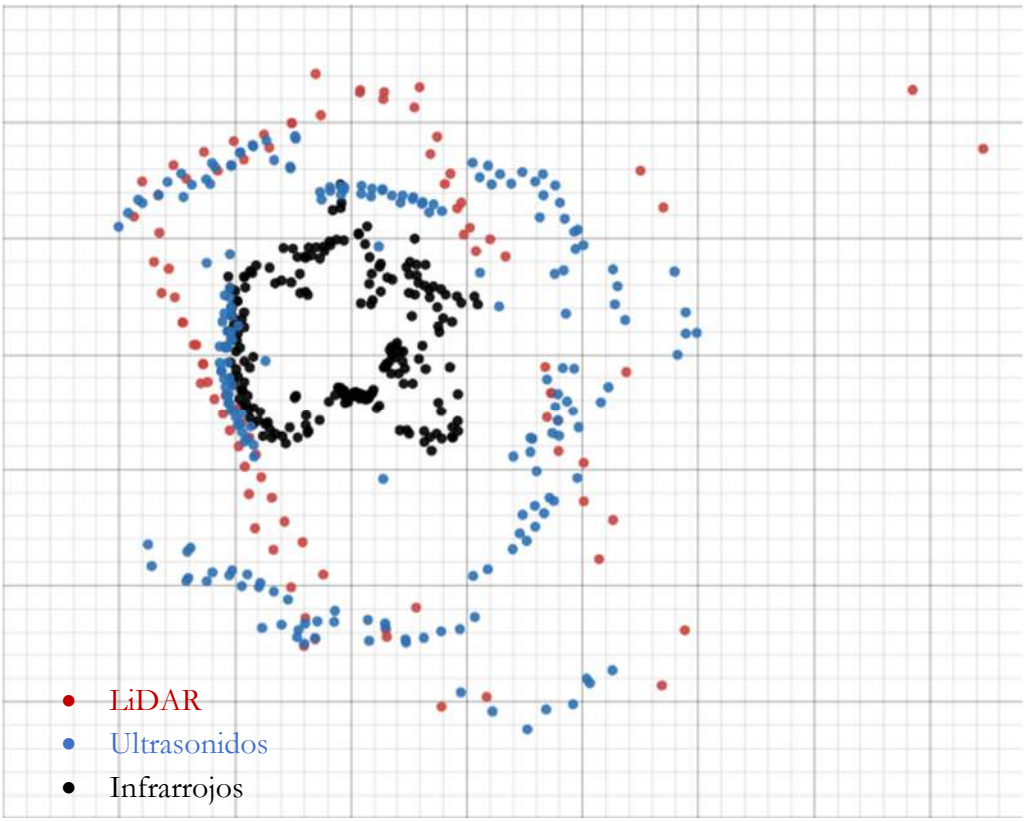


Figura 32

Como se puede observar, el sensor que da mejores resultados es el LiDAR v3 Lite junto con el motor paso a paso. Es el sensor que más se acerca al plano real del entorno representado en la figura 21.

8.- Propuesta de dron

Después de haber estudiado todas las posibilidades, vemos que el mejor sensor que podemos emplear para la adquisición de datos es el sensor láser.

Teniendo este elemento claro, vamos a proponer el diseño de un dron que tenga las capacidades para utilizar el LiDAR para resolver el auto posicionamiento.

8.1.- Sensor

Teniendo en cuenta que el LiDAR solo nos toma medidas de un punto, tenemos diversas configuraciones para la generación de la imagen 3D del entorno.

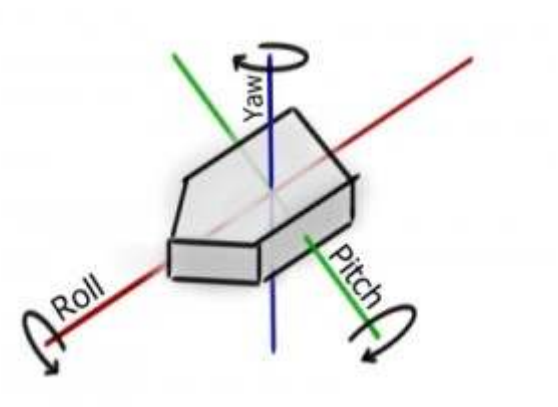


Figura 33

- Montar el sensor en un gimbal de 2 ejes para generar una nube de puntos de la semiesfera inferior del dron.
- Montar el sensor en un gimbal de un solo eje con la misma dirección que el cabeceo (pitch o eje y) del dron. Y utilizar la propia guiñada (yaw o eje z) del dron para obtener la semiesfera inferior del dron.

Otra solución es obviar la semiesfera de puntos inferior que aporta una información relativamente útil para el posicionamiento del dron y utilizar las siguientes dos opciones.

- Montar el sensor en un gimbal de un solo eje con la misma dirección que la guiñada (yaw o eje z) del dron para obtener el plano de puntos en el que se halla el dron.
- Fijar el sensor a la parte delantera del dron y utilizar la propia guiñada (yaw o eje z) de éste para obtener el plano de puntos en el que se halla el dron.

Debemos recordar que los barridos se realizarían con el dron en el aire, con lo que la inclusión de partes móviles puede ocasionar problemas al control por la generación de inercias inesperadas.

Viendo las posibilidades, acoplaremos el sensor LiDAR directamente al dron, sin usar ningún tipo de motor. Las piezas empleadas se encuentran en el anexo 2.

8.2.- Dron

Una vez seleccionado el sensor y su montaje se procederá a diseñar un dron que permita posteriormente alojar el sensor.

8.2.1.- Introducción

Las partes fundamentales de un dron son:

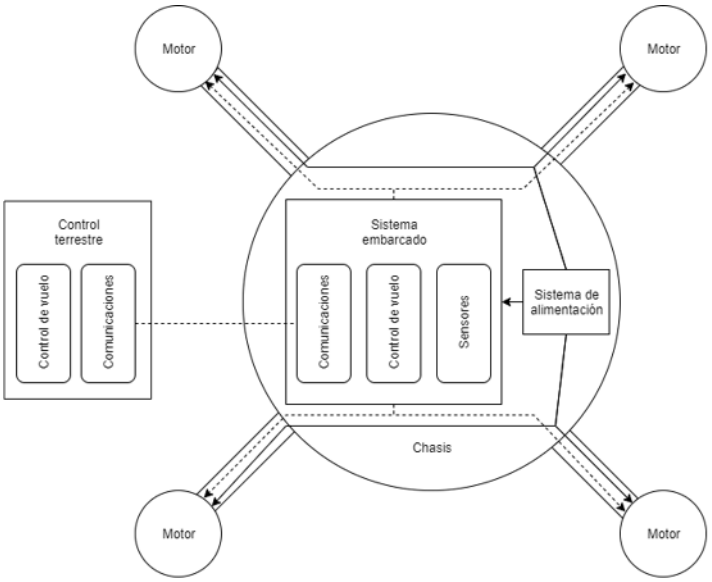


Figura 34

- Chasis: es la parte estructural encargada de acoger el resto de los elementos, mantenerlos protegidos y unidos.
- Motores: ejecutan las órdenes de movimiento mediante el movimiento de hélices, para elevar y trasladar la nave.
- Alimentación: energía para alimentar todos los componentes del dron.
- Sistema embarcado: se encarga de todos los procesos de información, control y comunicación de la nave. Este componente engloba:
    - Sensores: recogen información del entorno. Algunos de estos son los giroscopios, cámaras, acelerómetros, barómetros, GNSS o magnetómetros.
    - Comunicaciones: Se encargan de resolver el intercambio de información y órdenes entre la aeronave y el control terrestre. Engloba las antenas, los receptores y transmisores de radio, Wi-Fi u otras tecnologías de comunicación inalámbrica.
    - Sistema de control de vuelo: El encargado de mantener la altura, la estabilidad, la velocidad y la dirección coordinando los motores. Para llevar a cabo su cometido interpreta los datos adquiridos por

los sensores y las comunicaciones y toma la decisión de cómo actuar sobre los motores y actuadores.

- Control terrestre: equipo encargado de la supervisión de vuelo y monitorización del vuelo en tierra. Se reciben los datos tomados por la aeronave, a la par de ser posible enviar nuevas órdenes o modificar las iniciales. Este equipo puede tomar dos formas no excluyentes:
    - Radio controladora: se trata del mando a distancia que hace uso de ondas de radio para comunicar órdenes directas al UA (velocidad de ascenso/descenso, órdenes de desplazamiento, velocidad, armado/desarmado de motores y otras funciones implementadas en el dron). De esta forma le damos la condición de RPAS al sistema.
    - Estación terrestre: hace la función de una radio controladora a la que podría añadir algún tipo de procesamiento de alto nivel, como por ejemplo los datos recogidos por los sensores. El uso de estaciones terrestres viene ligada a la limitada capacidad de procesamiento que pueden tener los drones por sus limitaciones de carga.

El funcionamiento de un UAS puede ser muy variable en consecuencia de sus elementos, pero apoyándonos en nuestro esquema y en lo descrito sobre sus partes, podemos hablar de un funcionamiento genérico común en la gran mayoría. La aeronave, pese a no ser pilotada por un ser humano dentro de la propia nave, sí que debe ser controlada de alguna forma: bien mediante radiocontrol o mediante un control de vuelo. Se puede implementar el uso alternado entre el modo RPAS o el automatizado. Este modo automático es supervisado por la Estación de Control Terrestre o por el autopiloto. Desde la estación terrestre, podemos configurar la nave, enviarle órdenes o recibir información desde ella (posición GNSS, altura, velocidad, etc.). También sirve como punto de referencia de la nave, en caso de necesidad de regreso urgente

o de haber finalizado la misión de vuelo. La nave se guiará a través de datos tomados mediante sus sensores y antenas y actuará en consecuencia de su estado de vuelo y su programación de control de vuelo, o bien en consecuencia de las órdenes enviadas desde el control por radio. Independientemente de cuál de los dos caminos tome, toda orden o ejecución de movimiento pasará por el autopiloto. Esta herramienta nos facilita enormemente el manejo de la aeronave, ya que traduce nuestros movimientos o cambios dinámicos en el entorno a la aceleración o deceleración de los distintos motores de la nave y la lectura de datos tomados, de este modo dicho movimiento se realiza correctamente. En resumen, nos permite centrarnos en el movimiento total que queremos realizar, y no en la necesidad de controlar los motores uno a uno.

8.2.2- Dimensionamiento

Una vez conocidas las partes, lo principal para abordar el diseño, es estimar el peso del dron para dimensionar los componentes, luego iteraremos hasta encontrar qué componentes se adaptan a nuestra solución.

Ítem	Masa
Batería	120g
Raspberry, navio2 y electrónica adicional	200g
Motores y ESC	25*4g
Chasis	50g
Gimbal y sensores	125g
Hélices	30g
Cableado y tornillería	50g
Total	475g

La masa que deberá alzar el dron será dividida por cada uno de los cuatro motores. El motor escogido es DYS BE1806-2300KV, el cual, según indica el fabricante, es capaz de levantar una masa, con una batería de 3 celdas de litio (11.3v) y una hélice de 6" de diámetro como las seleccionadas, de entre 390g y 531g en función de las condiciones de uso y de la corriente suministrada.

Esta masa es para que el dron se mantenga en una posición estática, una buena estimación es que el dron sea capaz de alzar el doble.

Entonces si el dron debe alzar el doble:

$$m_{requerida} = 475 \cdot 2 = 950g$$

Y cada uno de los cuatro motores es capaz de alzar 390g

$$m_{capaz} = 4 \cdot 390 = 1560g$$

Vemos que se cumple:

$$m_{capaz} > m_{requerida}$$

Sabiendo que estos motores son válidos para la batería de 3 celdas, nos queda seleccionar los reguladores de tensión (ESC) para que alimenten a los motores bajo las instrucciones del autopiloto.

Para estos motores necesitamos un ESC que cumpla con la potencia y amperaje máximos de los motores. Tiene que poder suministrar 84,4W con una corriente de línea de 7,6A, por eso hemos seleccionado el DYS XSC20A que es capaz de suministrar hasta 100W de forma continuada.

Habiendo comprobado que todas las piezas cumplen con los requisitos, podemos seguir con la conexión. El esquema de conexión del dron es el siguiente:

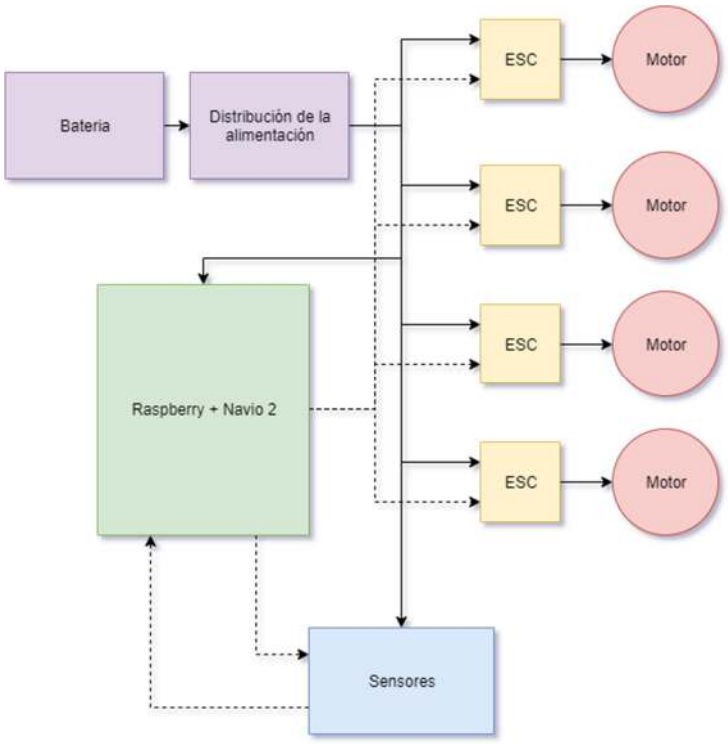


Figura 35

8.2.3.- Alimentación

Todo el sistema tiene que ir alimentado desde la batería, se diseñara un sistema de distribución de energía.

Para permitir que se puedan intercambiar las distintas partes del dron, vamos a diseñar un sencillo circuito para alimentar todos los componentes:

- El circuito debe tener una salida de 12v (directamente desde la batería) para cada uno de los circuitos reguladores de velocidad de los motores. Cada regulador de velocidad tiene un consumo máximo de 20A. Añadiremos una salida adicional para posibles usos futuros. En total habrán 5 salidas.

- El circuito debe tener salidas de 5v para toda la electrónica adicional. Pondremos 14 salidas de 5v, estas salidas tendrán una potencia total de 25W, es decir, suministraremos 5A a repartir entre ellas.
- Añadiremos un LED (Light Emitting Diode) para asegurarnos que hay alimentación.
- Protegeremos la placa contra conexión inversa.

Para que sea más cómodo conectar los elementos, usaremos conectores de PCB para las salidas de 12v y para las salidas de 5v pondremos pines en la placa para que se puedan conectar fácilmente conectores Dupont.

Para la etapa de 5v – 5A necesitaremos un DC-DC Buck. Usaremos un LM1084-5A que cumple con nuestros requisitos y además es de alta eficiencia, con lo que la energía disipada en la conversión es menor aumentando la duración de la batería.

Para proteger la placa de una conexión inversa indebida, pondremos un diodo a la entrada. Para eso usaremos un MBR60100CT que es un puente de diodos de 30A cada uno, pero los conectaremos en paralelo para asegurarnos una entrada de hasta 60A.

El valor de R3 es:

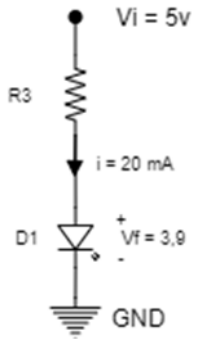


Figura 36

$$R3 = \frac{v_i - v_f}{i} = \frac{5 \cdot 3{,}9}{0{,}02} = 70\ \Omega \approx 68\Omega$$

El valor de R_1 y R_2 es el recomendado por el fabricante en la hoja de especificaciones. La relación que deben cumplir es:

$$\frac{R_2}{R_1} \approx 3$$

Debido a las existencias que teníamos en el laboratorio hemos usado:

$$\frac{R_2}{R_1} = \frac{1000}{330} = 3{,}03 \approx 3$$

El valor de los condensadores C_1 y C_2 es de 10µF, recomendado también por el fabricante. El condensador C_2 es de tantalio.

El esquema del circuito será:

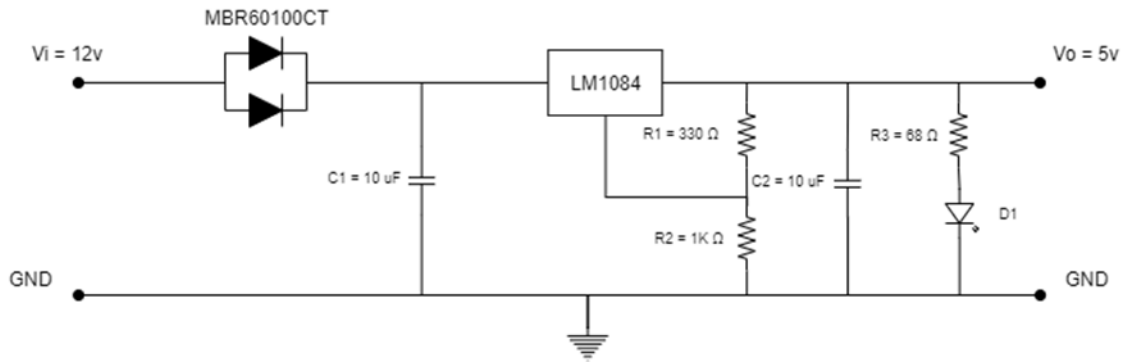


Figura 37

Como solo vamos a utilizar un circuito, usaremos una PCB de prototipado preagujereada donde ensamblar los componentes. Utilizaremos, también, conectores para facilitar la conexión y evitar tener que soldar.

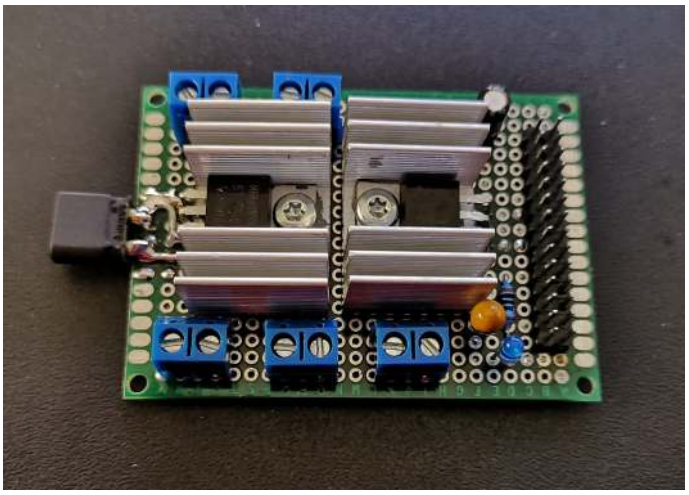


Figura 38

8.3.- Posicionamiento

En la propuesta del dron lo único pendiente por definir es la relación que tendrán el autopiloto y sensor.

El sensor se puede conectar directamente al puerto i2c del navio2 y usando las librerías que se han creado para el LiDAR obtener los puntos, tratarlos y operar usando la librería de DronKit, que es una librería de código abierto que permite enviar ordenes al autopiloto.

Aunque esta relación pueda parecer suficiente, es mejor no consumir recursos del autopiloto. La mejor alternativa para no sobrecargar los recursos del autopiloto es enviar los datos usando el programa creado para el envío de puntos a través de Wi-Fi a una estación de tierra y esta, a su vez, tratarlas y enviar las ordenes al autopiloto a través de MAVLink, que es una capa de abstracción sobre la comunicación especializada en drones. La siguiente figura muestra un esquema del sistema:

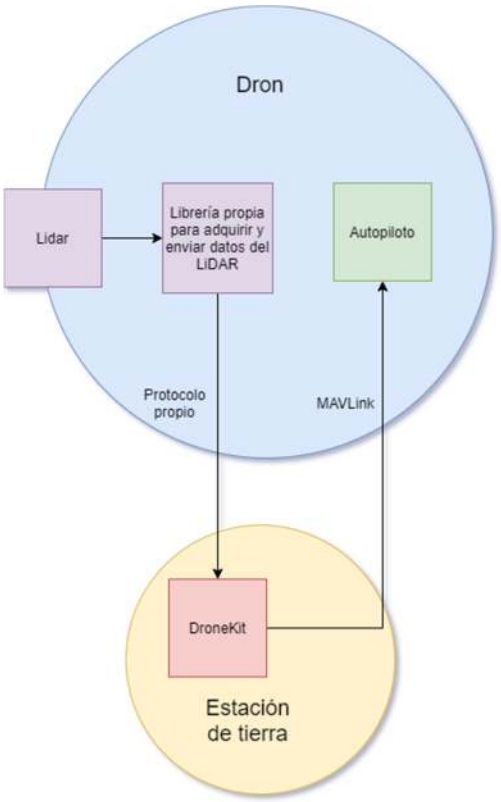


Figura 39

De este modo, los cálculos necesarios para el SLAM y el posicionamiento no comparten recursos de computación con el autopiloto. Así no se corre el riesgo de afectar el rendimiento del autopiloto.

Otra alternativa, aunque más costosa, es añadir una Raspberry Pi adicional que se empleara como compañero de la que soporta el auto piloto. Esta Raspberry Pi conectaría directamente con el LiDAR y procesaría las nubes de puntos. La conexión entre las dos Raspberry Pi se realizará a través de la UART y usando el protocolo MAVLink. Esta Raspberry Pi empleará Emlid Raspbian como sistema operativo, para beneficiarse de ser un sistema de tiempo real, e incorporará el DronKit.

Con cualquiera de ambos métodos podemos, en un futuro, emplear librerías como OpenCV para la interpretación de los datos y la resolución del SLAM.

El sensor LiDAR se conectara a esta Raspberry Pi adicional a través del bus i2c, igual que en los ensayos. Para alimentar al sensor es posible utilizar los pines de 5v de la fuente diseñada.

En la figura 40 podemos ver el ensamblado final del dron.

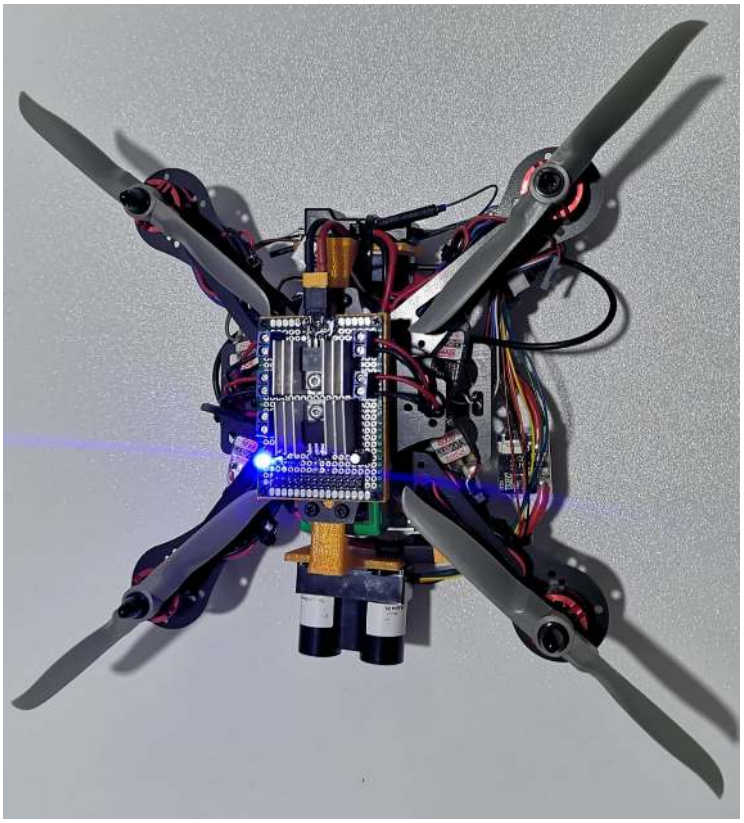


Figura 40

9.- Presupuesto

Después de realizar el proyecto, podemos abordar el desglose del coste

Item	coste unitario		cantidad		coste total	
Ingeniero	20	€/h	300	h	6000	€
Lidar	150	€	1	u.	150	€
Raspberry	35	€	2	u.	70	€
Motores	12	€	4	u.	48	€
Bateria	20	€	1	u.	20	€
ESC	6	€	4	u.	24	€
Sensor Ultrasonico	6	€	1	u.	6	€
Sensor Infrarrojos	5	€	1	u.	5	€
ADC	4	€	1	u.	4	€
Emisor/Receptor	60	€	1	u.	60	€
Navio2	200	€	1	u.	200	€
Hilo de impresión	25	€/Kg	0,5	Kg	12,5	€
Electronica adicional	20	€	1	u.	20	€
Helices	2	€	1,5	u.	3	€
Total					6622,5	€

Las horas presupuestadas corresponden a las horas dedicadas al proyecto. Este tipo de trabajo no se diferencia mucho de mi empleo actual, por lo que solo he contemplado mi coste de oportunidad.

10.- Conclusiones

Con este proyecto nos hemos sumergido en el campo de los sistemas autónomos y de los drones, consiguiendo aplicar técnicas básicas para desarrollar un sistema de sensores para un dron. Hemos podido analizar diferentes tipos de sensor para descubrir sus puntos fuertes y débiles.

Hemos comprobado las bondades de los sensores láser frente a otros sensores de distancia.

Mediante el cumplimiento de los objetivos planteados inicialmente, hemos podido desarrollar diferentes partes del proyecto desde diferentes campos de la ingeniería. Entre dichos campos cabe destacar la ingeniería mecánica y de materiales para el diseño y creación de las diferentes piezas mecánicas mediante impresión 3D, el diseño electrónico de circuitos para las fases de alimentación e instrumentación, la arquitectura de sistemas, la programación, tanto de alto nivel con las librerías visuales como de bajo nivel para la implantación de los sensores y motores y la ingeniería aeronáutica para la elección de motores y hélices para el dron.

También han sido muy gratificantes todas las tareas de nivel técnico relacionadas con el ensamblaje, soldaduras y la toma de datos.

Concluir con una reflexión sobre los recursos del proyecto consumidos. Gran parte de ellos se han consumido en el proceso de selección del sensor, la implantación del software y de la mecánica para cada uno de los sensores, que ha sido necesaria para justificar la inclusión de un sensor mucho más caro que otros existentes. También en la confección de una interfaz capaz de recibir datos en cualquier ordenador que se conecte a través de Wi-Fi y visualizarlos en tiempo real, así como su contraparte, que los puede enviar desde la Raspberry

Pi y que permite seleccionar qué tipo de sensor se va a utilizar. Esto implica que no se ha podido llegar más lejos de los objetivos planteados inicialmente y abordar el trabajo de dotar de autonomía al dron.

11.- Trabajos futuros

Enlazando con el trabajo realizado con este proyecto, existen diferentes proyectos adicionales que podrían ser de interés:

- Dotar de autonomía al dron usando los sistemas confeccionados.
- Implementar el sistema en un proceso industrial o aplicar a usos diversos.
- Crear un sistema comercial de bajo coste para los aficionados a los drones para subsanar errores en el pilotaje y evitar dañar el dron.
- Solventar el problema del SLAM para situaciones "indoor".
- Ensayar otras tipologías de montaje del sensor en el dron, así como otros sensores o combinaciones de ellos.
- Dotar de herramientas al dron para realizar aplicaciones, como por ejemplo dotar de una cámara para vigilar interiores o evaluación de riesgos en minas, ruinas o cavernas.
- Implementar el tratamiento de las señales obtenidas por los sensores para mejorar sus prestaciones.
- Implementar alternancia con el modo RPAS para evitar accidentes por fallo humano, dotando al dron de la capacidad de no acercarse a los obstáculos a menos de la distancia deseada. Esta aplicación puede desarrollarse en un producto comercial con el debido enfoque al cliente final.

12.- Bibliografía

DroneKit Documentation. (s.f.). Recuperado 9 febrero, 2020, de http://dronekit.io/

Antuña Herrero, R. (s.f.). *Drone Design*. Recuperado de http://digibuo.uniovi.es/dspace/bitstream/10651/43461/3/TFG_Antu%C3%B1a_Herrero.pdf

ArduPilot Development Site — Dev documentation. (s.f.). Recuperado 9 febrero, 2020, de https://ardupilot.org/dev/index.html

Boucher, R. J. (1994). *The Electric Motor Handbook: The Complete Handbook of High Performance D.C. Motors*. California, Estados Unidos: AstroFlight.

Control Tower - November 1998 - RC Groups. (1998, 1 noviembre). Recuperado 25 enero, 2020, de https://www.rcgroups.com/forums/showthread.php?393251-Control-Tower-November-1998

Karlsson, N., Di Bernardo, E., Ostrowski, J., Goncalves, L., Pirjanian, P., & Munich, M. (2005). The vSLAM Algorithm for Robust Localization and Mapping. *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, . https://doi.org/10.1109/robot.2005.1570091

Lemaire, T., Berger, C., Jung, I., & Lacroix, S. (2007). Vision-Based SLAM: Stereo and Monocular Approaches. *International Journal of Computer Vision*, *74*(3), 343–364. https://doi.org/10.1007/s11263-007-0042-3

Magnabosco, M., & Breckon, T. P. (2013). Cross-spectral visual simultaneous localization and mapping (SLAM) with sensor handover. *Robotics and Autonomous Systems*, *61*(2), 195–208. https://doi.org/10.1016/j.robot.2012.09.023

Mark Bridge, M. B. (2019, 26 octubre). Self-driving cars take to London roads. Recuperado 25 enero, 2020, de https://www.thetimes.co.uk/article/self-driving-cars-take-to-london-roads-ccx8ssksd

MAVLink Developer Guide. (s.f.). Recuperado 9 febrero, 2020, de https://mavlink.io/en/

Protección contra inversiones de polaridad. (s.f.). Recuperado 25 enero, 2020, de http://www.radioelectronica.es/radioaficionados/19-inversion-polaridad

Riisgaard, S., & Rufus Blas, M. (2004). *SLAM for Dummies*. Recuperado de https://dspace.mit.edu/bitstream/handle/1721.1/36832/16-412JSpring2004/NR/rdonlyres/Aeronautics-and-Astronautics/16-412JSpring2004/A3C5517F-C092-4554-AA43-232DC74609B3/0/1Aslam_blas_report.pdf

13.- Anexo 1

Radar_client.py

```
from TFG_Libraries import Wifi
import vpython as vp
from time import sleep


program = 2


def connect(connect_state_button):
    """Funcion respuesta al pulsar boton, crea la conexion UDP con la
raspberry como cliente"""

    global connection
    connection = Wifi.ClientUDP()  # crea el objeto conexion
    connect_state_button.text = 'Connected'  # cambia el tsxto del
boton


def scan(scan_state_button):

    global connection

    if program == 1:
        """Lidar + Step motor"""

        connection.send('lst')
        connection.send('3.6')
        connection.send('200')

    elif program == 2:
        """Lidar + Servo"""

        connection.send('lse')

    elif program == 3:
        """Ultrasonic + Step motor"""

        connection.send('ust')
        connection.send('3.6')
        connection.send('200')

    elif program == 4:
        """Ultrasonic + Servo"""

        connection.send('use')

    elif program == 5:
        """Infrared + Step motor"""

        connection.send('ist')
        connection.send('3.6')
```

```
        connection.send('200')

    elif program == 6:
        """Infrared + Servo"""

        connection.send('ise')

    elif program == 7:
        """Infrared + Brushless"""

        connection.send('ibr')

    elif program == 8:
        """Ultrasonic + Brushless"""

        connection.send('ubr')

    elif program == 9:
        """Lidar + Brushless"""

        connection.send('lbr')

    scan_state_button.text = 'Scanning'  # cambia el tsxto del boton

    box = vp.box(pos=vp.vector(0, 0, 0), size=vp.vector(5, 5, 5))
    box.color = vp.vector(0, 0, 1)

    file = open('Sensor_test.txt', 'w')

    while True:
        point = connection.receive()
        if point == 'End':
            scan_state_button.text = 'Scanned'
            break
        else:
            point = point.split(',')
            x = float(point[0])
            y = float(point[1])
            z = 0

            print(x, y, z)
            vp.sphere(pos=vp.vector(x, y, z), radius=1)
            val = str(x) + ', ' + str(y) + '\n'
            file.write(val)
    file.close()


def close(close_state_button):
    """Funcion respuesta al pulsar boton, cierra la conexion UDP"""

    global connection

    try:
        close_state_button.text = 'Closing'  # cambia el tsxto del
boton
        connection.send('0')  # enviamos el numero de programa de
cierre a la raspberry
    except NameError:
```

```
        pass
    finally:
        print('Bye')
        close_state_button.text = 'Closed'  # cambia el tsxto del
boton
        vp.scene.exit = False  # cierra la ventana


vp.scene.width = 1000
vp.scene.height = 600
vp.scene.range = 1.3
vp.scene.title = "Sky Scanner\n"
vp.scene.autocenter = True
vp.scene.autoscale = True
vp.button(text="Connect", pos=vp.scene.title_anchor, bind=connect)
vp.button(text="Scan", pos=vp.scene.title_anchor, bind=scan)
vp.button(text="Close", pos=vp.scene.title_anchor, bind=close)


try:
    while True:
        sleep(0.000001)
except KeyboardInterrupt:
    print('Exit')
```

Radar-server.py

```
from TFG_Libraries import *
from numpy import cos, sin, pi
from time import sleep
import pigpio


def calculate(distance_cm, angle_deg):
    """Calculate x and y coordinates, and put them inside a string
ready to send it"""
    angle_rad = angle_deg*pi/180
    x = distance_cm*cos(angle_rad)
    y = distance_cm*sin(angle_rad)
    point_str = str(x) + ',' + str(y)
    return point_str


try:
    while True:
        connection = ServerUDP(server_ip='192.168.50.10')  # Crea un
servidor UDP y espera un cliente
        engine = None
        rpi = pigpio.pi()  # Crea el item raspberry para el uso de los
GPIO

        while True:
            program = connection.receive()  # Recibe el programa para
la accion a realizar

            if program == 'lst':
                """Lidar + Step motor"""
```

```
                lidar = LidarV3(rpi)  # Crea la clase lidar para usar
el dispositivo
                engine = StepMotor(rpi)  # Crea la clase engine para
usar el motor paso a paso
                angle = float(connection.receive())  # 3.6  Recibe el
angulo entre punto y punto
                num_points = int(int(connection.receive())/2)  # 200
Recibe el numero de puntos a medir y lo devide en 2
                for i in range(0, num_points):  # Hace un barrido
hacia un lado
                    engine.go2angle(angle)  # Envia el motor al angulo
correspondiente
                    sleep(0.1)
                    distance = lidar.read_distance()  # Lee del sensor
                    point = calculate(distance, angle*i)  # Calcula la
distancia
                    print(point)
                    connection.send(point)  # Devuelve las coordenadas
del punto
                for i in range(num_points, 0, -1):  # Vuelve haciendo
un barrido hacia el otro lado
                    engine.go2angle(-angle)  # Envia el motor al
angulo correspondiente
                    sleep(0.1)
                    distance = lidar.read_distance()
                    point = calculate(distance, angle*i)
                    print(point)
                    connection.send(point)
                connection.send('End')

            if program == 'ist':
                """Lidar + Step motor"""

                ir = IRSensor(rpi)  # Crea la clase lidar para usar el
dispositivo
                engine = StepMotor(rpi)  # Crea la clase engine para
usar el motor paso a paso
                angle = float(connection.receive())  # 3.6  Recibe el
angulo entre punto y punto
                num_points = int(int(connection.receive())/2)  # 200
Recibe el numero de puntos a medir y lo devide en 2
                for i in range(0, num_points):  # Hace un barrido
hacia un lado
                    engine.go2angle(angle)  # Envia el motor al angulo
correspondiente
                    sleep(0.1)
                    distance = ir.read_distance()  # Lee del sensor
                    point = calculate(distance, angle*i)  # Calcula la
distancia
                    print(point)
                    connection.send(point)  # Devuelve las coordenadas
del punto
                for i in range(num_points, 0, -1):  # Vuelve haciendo
un barrido hacia el otro lado
                    engine.go2angle(-angle)  # Envia el motor al
angulo correspondiente
                    sleep(0.1)
```

```
                    distance = ir.read_distance()
                    point = calculate(distance, angle*i)
                    print(point)
                    connection.send(point)
                connection.send('End')

            elif program == 'lse':
                """Lidar + Servo"""

                lidar = LidarV3(rpi)
                engine = ServoMotor(rpi)
                for angle in engine.servo_range_deg(num_increment=50):
                    engine.go2angle(angle)  # Envia el motor al angulo
correspondiente
                    sleep(0.05)
                    distance = lidar.read_distance()
                    point = calculate(distance, angle)
                    print(point)
                    connection.send(point)
                for angle in
engine.servo_range_deg(start=engine.max_angle, end=engine.min_angle,
num_increment=50):
                    engine.go2angle(angle)  # Envia el motor al angulo
correspondiente
                    sleep(0.05)
                    distance = lidar.read_distance()
                    point = calculate(distance, angle)
                    print(point)
                    connection.send(point)
                connection.send('End')

            elif program == 'ust':
                """Ultrasonic + Step motor"""

                usonic = USonic(rpi)
                engine = StepMotor(rpi)
                angle = float(connection.receive())  # 3.6
                num_points = int(int(connection.receive())/2)  # 200
                for i in range(0, num_points):
                    engine.go2angle(angle)  # Envia el motor al angulo
correspondiente
                    sleep(0.1)
                    distance = usonic.read_distance()
                    point = calculate(distance, angle*i)
                    print(point)
                    connection.send(point)
                for i in range(num_points, 0, -1):
                    engine.go2angle(-angle)  # Envia el motor al
angulo correspondiente
                    sleep(0.1)
                    distance = usonic.read_distance()
                    point = calculate(distance, angle*i)
                    print(point)
                    connection.send(point)
                connection.send('End')

            elif program == 'use':
                """Ultrasonic + Servo"""
```

```
                usonic = USonic(rpi)
                engine = ServoMotor(rpi)
                for angle in engine.servo_range_deg():
                    engine.go2angle(angle)
                    sleep(0.05)
                    distance = usonic.read_distance()
                    point = calculate(distance, angle)
                    print(point)
                    connection.send(point)
                for angle in engine.servo_range_deg
(start=engine.max_angle, end=engine.min_angle, num_increment=30):
                    engine.go2angle(angle)
                    sleep(0.05)
                    distance = usonic.read_distance()
                    point = calculate(distance, angle)
                    print(point)
                    connection.send(point)
                connection.send('End')

            elif program == 'lbr':
                """Lidar + brushless"""

                lidar = LidarV3(rpi)  # Crea la clase lidar para usar
el dispositivo
                engine = Brushless(rpi)  # Crea la clase engine para
usar el motor paso a paso
                engine.turn(120)  # Envia el motor al angulo
correspondiente
                for i in range(0, 200):
                    distance = lidar.read_distance()  # Lee del sensor
                    angle = engine.read_angle()
                    point = calculate(distance, angle)  # Calcula la
distancia
                    print(point)
                    connection.send(point)  # Devuelve las coordenadas
del punto
                connection.send('End')

            elif program == 'ubr':
                """Ultrasonic + brushless"""

                usonic = USonic(rpi)
                engine = Brushless(rpi)
                engine.turn(15)  # Envia el motor al angulo
correspondiente
                for i in range(0, 200):
                    distance = usonic.read_distance()  # Lee del
sensor
                    angle = engine.read_angle()
                    point = calculate(distance, angle)  # Calcula la
distancia
                    print(point)
                    connection.send(point)  # Devuelve las coordenadas
del punto
                connection.send('End')

            elif program == 'ibr':
```

```
                """Infrared + brushless"""

                ir = IRSensor(rpi)
                engine = Brushless(rpi)
                engine.turn(15)  # Envia el motor al angulo
correspondiente
                for i in range(0, 200):
                    distance = ir.read_distance()  # Lee del sensor
                    angle = engine.read_angle()
                    point = calculate(distance, angle)  # Calcula la
distancia
                    print(point)
                    connection.send(point)  # Devuelve las coordenadas
del punto
                connection.send('End')

            elif program == 0:
                engine.free_motor()
                connection.close()
                rpi.stop()
                break
            else:
                connection.send('Program not correct')

except KeyboardInterrupt:
    print('End')
```

Wifi.py

```
 import time
import socket


class IncorrectPass(Exception):
    """The password was not correct"""
    pass


class UDPConnection:

    def __init__(self, port=7777, connection_ip=''):

        self.connection_ip = connection_ip
        self.port = port
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    def send(self, data):
        """Send a encoded string across the socket connection"""
        self.socket.sendto(data.encode('ascii'), (self.connection_ip,
self.port))

    def receive(self):
        """Receive a string from the socket connection, returns a
decoded string"""
        data = self.socket.recv(1024).decode('ascii')
        if data == '':
            raise ConnectionAbortedError
```

```
        return data

    def close(self):
        """Close connection"""
        self.socket.close()

    @staticmethod
    def _test_(rasporpc):
        message = 'Ping'
        try:
            while True:
                if rasporpc == 'rasp':
                    con = ServerUDP()
                    print('Waiting data...')
                    data = con.receive()
                    print(data, 'to echo')
                    con.send(data)
                    time.sleep(2)
                    print('Closing...')
                    con.close()
                elif rasporpc == 'pc':
                    con = ClientUDP()
                    print('Sending data...', message)
                    con.send(message)
                    print('Receiving echo...')
                    data = con.receive()
                    print(data, 'received')
                    time.sleep(2)
                    print('Closing...')
                    con.close()
                    message = input('Message')
                else:
                    raise ValueError
        except KeyboardInterrupt:
            con.close()
            print('Exit')


class ServerUDP (UDPConnection):

    def __init__(self, port=7777, server_ip='', rasp_pas='skysca'):

        super().__init__(port, server_ip)

        self.socket.bind((self.connection_ip, self.port))  # Bind the
socket to a local address
        self.own_ip, self.port = self.socket.getsockname()
        print('Waiting connection...')
        self.password, client_address = self.socket.recvfrom(1024)
        self.connection_ip, self.port = client_address
        if self.password.decode('ascii') != rasp_pas:
            raise IncorrectPass
        self.socket.sendto('Accepted'.encode('ascii'),
(self.connection_ip, self.port))
        print('Connection--> From:', self.connection_ip, 'To us: ',
self.own_ip)
```

```
class ClientUDP(UDPConnection):

    def __init__(self, port=7777, server_ip='192.168.50.10',
rasp_pas='skysca'):

        super().__init__(port, server_ip)

        print('Conecting to', self.connection_ip)
        self.socket.connect((self.connection_ip, self.port))
        self.socket.sendto(rasp_pas.encode('ascii'),
(self.connection_ip, self.port))
        answer = self.socket.recv(1024).decode('ascii')
        if answer == 'Accepted':
            print(answer)
        else:
            raise ConnectionError(answer, "'Accepted' expected'")
```

Ultrasonic.py

```
import time

import pigpio

class USonic:
    """
    This class encapsulates a type of acoustic ranger.  In particular
    the type of ranger with separate trigger and echo pins.

    A pulse on the trigger initiates the sonar ping and shortly
    afterwards a sonar pulse is transmitted and the echo pin
    goes high.  The echo pins stays high until a sonar echo is
    received (or the response times-out).  The time between
    the high and low edges indicates the sonar round trip time.
    """

    def __init__(self, rpi, trigger=17, echo=27):
        """
        The class is instantiated with the Pi to use and the
        gpios connected to the trigger and echo pins.
        """
        self.rpi = rpi
        self._trig = trigger
        self._echo = echo

        self._ping = False
        self._high = None
        self._time = None

        self._triggered = False

        self._trig_mode = rpi.get_mode(self._trig)
        self._echo_mode = rpi.get_mode(self._echo)
```

```
        rpi.set_mode(self._trig, pigpio.OUTPUT)
        rpi.set_mode(self._echo, pigpio.INPUT)

        self._cb = rpi.callback(self._trig, pigpio.EITHER_EDGE,
self._cbf)
        self._cb = rpi.callback(self._echo, pigpio.EITHER_EDGE,
self._cbf)

        self._inited = True

    def _cbf(self, gpio, level, tick):
        if gpio == self._trig:
            if level == 0:  # trigger sent
                self._triggered = True
                self._high = None
        else:
            if self._triggered:
                if level == 1:
                    self._high = tick
                else:
                    if self._high is not None:
                        self._time = tick - self._high
                        self._high = None
                        self._ping = True

    def read_distance(self):
        """
        Triggers a reading.  The returned reading is the number
        of microseconds for the sonar round-trip.

        round trip cms = round trip time / 1000000.0 * 34030
        """
        if self._inited:
            self._ping = False
            self.rpi.gpio_trigger(self._trig)
            start = time.time()
            while not self._ping:
                if (time.time() - start) > 1.0:
                    return 200
                time.sleep(0.001)
            return self._time * 34030 / (2 * 1000000)
        else:
            return None

    def cancel(self):
        """
        Cancels the ranger and returns the gpios to their
        original mode.
        """
        if self._inited:
            self._inited = False
            self._cb.cancel()
            self.rpi.set_mode(self._trig, self._trig_mode)
            self.rpi.set_mode(self._echo, self._echo_mode)


def _test_():
```

```
    try:
        rpi = pigpio.pi()

        sonar = USonic(rpi, 17, 27)

        end = time.time() + 600.0

        r = 1
        while time.time() < end:
            print("{} {}".format(r, sonar.read_distance()))
            r += 1
            time.sleep(0.03)
    except KeyboardInterrupt:
        sonar.cancel()
        rpi.stop()
        print('Exit')
```

Step.py

```
from time import sleep
import pigpio


class StepMotor:

    def __init__(self, rpi, wires=(26, 19, 13, 6)):
        """ Inicializa el motor con su cableado correspondiente:
         26->in1
         19->in2
         13->in3
         6->in4"""

        self.rpi = rpi

        # set up GPIO output channel
        self.wires = wires
        self.steps = ['0001', '0011', '0010', '0110', '0100', '1100',
'1000', '1001']
        self.current_step = 0

        for output in self.wires:
            self.rpi.set_mode(output, pigpio.OUTPUT)

        # start motor
        self.singular_step(0)

    def free_motor(self):
        """ Para el motor """

        for output in self.wires:
            self.rpi.write(output, False)

    def singular_step(self, step):
        """ Envía al motor el paso requerido, con step = -1 apaga el
```

```
motor"""
        i = 0
        if step > len(self.steps) or step < -1:
            raise ValueError('Step out of range:', step)
        step = int(step)
        if step == -1:
            self.free_motor()
        else:
            for output in self.wires:
                self.rpi.write(output, bool(int(self.steps[step][i])))
                # print(step, '---->', output, '---->',
bool(self.steps[step][i]), '---->', self.steps[step][i])
                i += 1

    def _inc_step_(self):

        if self.current_step >= 7:
            self.current_step = 0
        else:
            self.current_step += 1

        return self.current_step

    def _dec_step_(self):

        if self.current_step <= 0:
            self.current_step = 7
        else:
            self.current_step -= 1

        return self.current_step

    def forward(self, num_steps, init_step=0, vel=0.0008):
        """Hace avanzar el motor hacia delante, vel_max =~
sleep(0.0008)"""

        for num_step in range(init_step, num_steps):
            self.singular_step(self._inc_step_())
            sleep(vel)

    def backward(self, num_steps, init_step=0, vel=0.0008):

        for num_step in range(init_step, num_steps):
            self.singular_step(self._dec_step_())
            sleep(vel)

    def go2angle(self, angle):

        if angle > 0:
            num_steps = int(angle / 0.088)
            self.forward(num_steps)
        elif angle < 0:
            num_steps = int((-angle) / 0.088)
            self.backward(num_steps)

    @staticmethod
    def _test_(rpi):
```

```
        motor1 = StepMotor(rpi)

        try:
            while True:
                print('Forward')
                motor1.go2angle(180)
                print('Backward')
                motor1.go2angle(-180)

        except KeyboardInterrupt:
            motor1.free_motor()
            print('Exit')
```

Servo.py

```
from time import sleep
import pigpio


class ServoMotor:

    def __init__(self, rpi,  servo_type='SG90', pin=18,
multiplexor=False):

        self.pin = pin
        self.servo_type = servo_type
        self.multiplexor = multiplexor
        self.rpi = rpi

        if self.servo_type == 'SG90':
            self.start = 0.6 #0.5  # 1.07  # ms
            self.inc = 0.010
            self.end = 2.4  # 1.64

            self.max_angle = 207  # degrees
            self.min_angle = 0

            self.pwm_period_hz = 50  # Hz

            self.servo_pin = pin

        else:
            raise NotImplementedError

        if not self.multiplexor:
            self.rpi.set_mode(self.pin, pigpio.OUTPUT)

        else:
            raise NotImplementedError

    @staticmethod
    def _ms2us_(miliseconds):
        """Convert milieconds to microseconds"""
        microseconds = miliseconds*1000
        return microseconds
```

```
    def _ms2per_(self, miliseconds):
        """Convert miliseconds of duty cycle from PWM in % of duty
cycle"""

        dutycycle = miliseconds*self.pwm_period_hz/10

        return dutycycle

    def _ms2deg_(self, miliseconds):
        """Convert miliseconds of duty cycle from PWM in degrees"""

        m = (self.max_angle - self.min_angle)/(self.end - self.start)
        r = self.min_angle - m*self.start

        deg = miliseconds*m+r

        return deg

    def _deg2ms_(self, degrees):
        """Convert degrees to miliseconds of duty cycle from PWM"""

        m = (self.max_angle - self.min_angle)/(self.end - self.start)
        r = self.min_angle - m*self.start

        return (degrees-r)/m

    def _move2pos_(self, angle_ms):
        """Send a pwm to move the servo in PWM(us)"""

        if not self.multiplexor:
            pulsewidth = self._ms2us_(angle_ms)
            self.rpi.set_servo_pulsewidth(self.pin, pulsewidth)
        else:
            raise NotImplementedError

    def go2angle(self, angle_deg):
        """The servomotor goes to the given angle in degrees"""

        angle_ms = self._deg2ms_(angle_deg)

        self._move2pos_(angle_ms)

    def free_motor(self):
        """ Stop commanding servomotor"""

        self.rpi.set_servo_pulsewidth(self.pin, 0)

    def servo_range_deg(self, start=None, end=None,
num_increment=None):
        """Returns and array of degrees from start to end degrees with
dimension num_increment"""

        array = []

        if start is None:
            start = self.min_angle
        if end is None:
```

```
            end = self.max_angle

        if start < self.min_angle:
            raise ValueError('Start=', start, ' have to be more than',
self.min_angle)
        if end > self.max_angle:
            raise ValueError('End=', end, ' have to be less than',
self.max_angle)
        if start > self.max_angle:
            raise ValueError('Start=', start, ' have to be less than',
self.max_angle)
        if end < self.min_angle:
            raise ValueError('End=', end, ' have to be more than',
self.min_angle)

        start = self._deg2ms_(start)
        end = self._deg2ms_(end)

        if num_increment is None:
            num_increment = 10
        if num_increment < 0:
            raise ValueError('Num_increment=', num_increment, 'have to
be more than 0')

        if start < end:
            increment = (end - start) / num_increment
            while start < end:
                array.append(self._ms2deg_(start))
                start = start + increment
        elif end < start:
            increment = (start - end) / num_increment
            while start > end:
                array.append(self._ms2deg_(start))
                start = start - increment
        else:
            raise ValueError('Start must not be equal to end')

        return array


def _test_(delay=1):
    rpi = pigpio.pi()
    servo = ServoMotor(rpi)
    try:
        while True:
            for angle in servo.servo_range_deg():
                print(angle)
                servo.go2angle(angle)
                sleep(delay)

    except KeyboardInterrupt:
        servo.free_motor()
        print('Exit')
```

Lidar.py

```
from time import sleep


class LidarV3:
    """These class is used to comunicate with a Lidar device, and read
the distance"""

    def __init__(self, rpi, device_address=0x62, bus=1):
        """Initialize the i2c comunication with the Lidar device"""
        self.rpi = rpi
        self.rpi_i2c_lidar = rpi.i2c_open(bus, device_address)  # open
a communication i2c to Lidar and returns a handle
        sleep(0.022)  # wait to establish the communication

    def _busy_(self):
        """Return False if Lidar is ready for a new command"""

        ready_address = 0x01  # address for busy bit

        ready = self.rpi.i2c_read_byte_data(self.rpi_i2c_lidar,
ready_address)  # read the busy bit

        if ready & (1 << 0) == 1:  # returns True or False depending
of the busy state of the Lidar
            return True
        else:
            return False

    def _write_(self, register, command):
        """Wait until Lidar is not busy and write the byte in the
register"""

        while self._busy_():  # Wait until Lidar stop being busy
            sleep(0.001)

        # Write the byte in the register for the handled Lidar
        self.rpi.i2c_write_byte_data(self.rpi_i2c_lidar, register,
command)

    def _read_(self, register):
        """Wait until Lidar is not busy and return a byte from the
register"""

        while self._busy_():  # Wait until Lidar stop being busy
            sleep(0.001)

        # Read the byte in the register for the handled Lidar
        value = self.rpi.i2c_read_byte_data(self.rpi_i2c_lidar,
register)
        return value

    def read_distance(self):
        """Return measure in cm from lidar"""

        read_address = 0x00
        read_command = 0x04

        low_measure_address = 0x10
```

```
        high_measure_address = 0x0f

        self._write_(read_address, read_command)

        low_measure = self._read_(low_measure_address)
        # print('low', low_measure)
        high_measure = self._read_(high_measure_address) << 8
        # print('high', high_measure)

        return low_measure + high_measure

    def config(self):
        """"""
        pass

    def close(self):
        """Close the communication with Lidar"""

        self.rpi.i2c_close(self.rpi_i2c_lidar)

    @staticmethod
    def _test_(rpi):

        lid = LidarV3(rpi)
        try:
            while True:
                print(lid.read_distance())
        except KeyboardInterrupt:
            print('Exit')
```

ADC_bits.py

```
import time
import pigpio

CONFIG_MSB=0x44#0b10000100
CONFIG_LSB=0x83#0b10000011
ADDRESS=0x48

PTR_conversion=0x00
PTR_configuration=0x01
PTR_lo_tresh=0x02
PTR_hi_tresh=0x03


class ADC:

    def __init__(self, rpi, device_address=0x48, bus=1):
        """Initialize the i2c comunication with the ADC device"""
        self.rpi = rpi
        self.rpi_i2c_ADC = rpi.i2c_open(bus, device_address)  # open a
communication i2c to ADC and returns a handle
        time.sleep(0.022)  # wait to establish the communication
        self._config_([CONFIG_MSB, CONFIG_LSB])
```

```
    def _write_(self, register, command):

        # Write the byte in the register for the handled ADC
        self.rpi.i2c_write_block_data(self.rpi_i2c_ADC, register,
command)

    def _read_(self, register, count):

        self._pointer_(register)
        # Read the byte in the register for the handled ADC
        (leng, value) = self.rpi.i2c_read_device(self.rpi_i2c_ADC,
count)
        if leng < 0 :
            return 0
        return value

    def _pointer_(self, register):

        self.rpi.i2c_write_device(self.rpi_i2c_ADC, [register])

    def _config_(self, config):
        """Configure the ADC"""
        self.rpi.i2c_write_device(self.rpi_i2c_ADC,
[PTR_configuration, CONFIG_MSB, CONFIG_LSB])

    def read(self):

        # Read the byte in the register for the handled ADC
        value = self._read_(PTR_conversion, 2)

        return ((value[0]<<8) + (value[1]))/16383

    def close(self):
        """Close the communication with Lidar"""

        self.rpi.i2c_close(self.rpi_i2c_ADC)

def _test_():
    rpi = pigpio.pi()
    adc = ADC(rpi)
    try:
        while True:
            print(adc.read())
            time.sleep(1)
    except KeyboardInterrupt:
        adc.close()
        print('Exit')
```

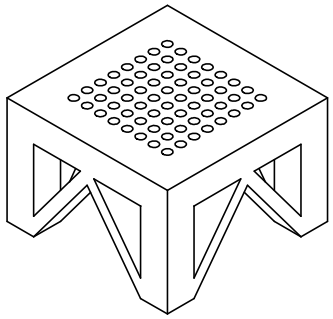
IR_Sensor.py

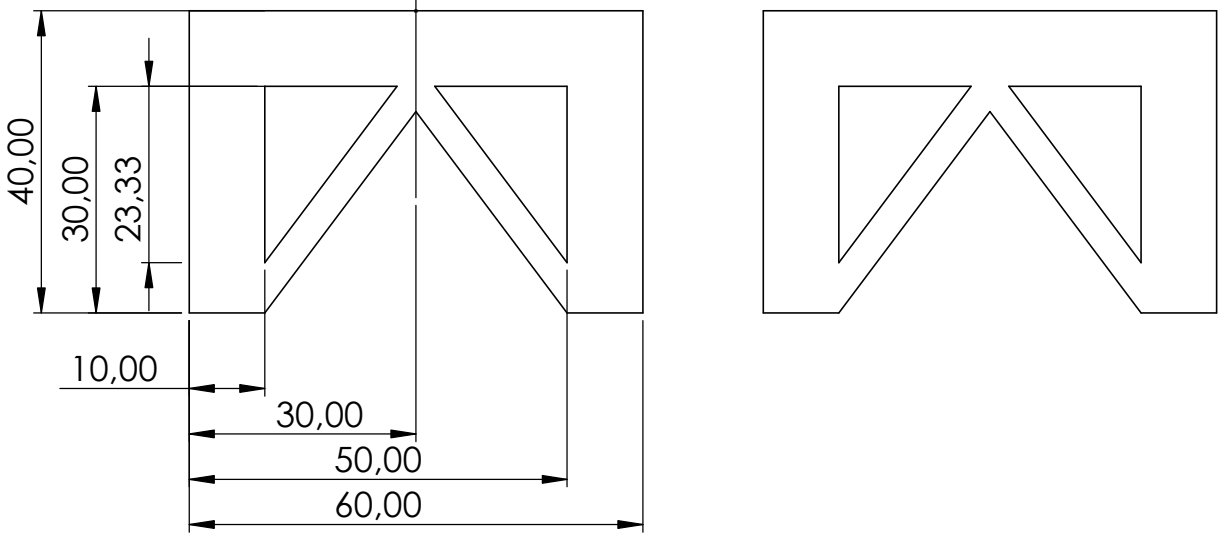
```
import ADC_16bits
import pigpio
import time
```

```
ADC = ADC_16bits.ADC


class IRSensor:

    def __init__(self, rpi):

        self.rpi = rpi
        self.adc = ADC(rpi)

    def read_distance(self):
        """Takes one distance measure"""
        measure = self.adc.read()
        if measure == 0:
            return 0
        return  5.54*pow(measure, -1.29)

    def close(self):
        """Close sensor"""
        self.adc.close()


def _test_():

    rpi = pigpio.pi()
    ir = IRSensor(rpi)
    try:
        while True:
            print(ir.read_distance())
            time.sleep(1)
    except KeyboardInterrupt:
        print('Exit')
```

14.- Anexo 2

En las siguientes paginas se encuentran los planos de las piezas diseñadas con SolidWorks e impresas con impresora 3D. El software de impresión empleado ha sido el Repetier.





TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

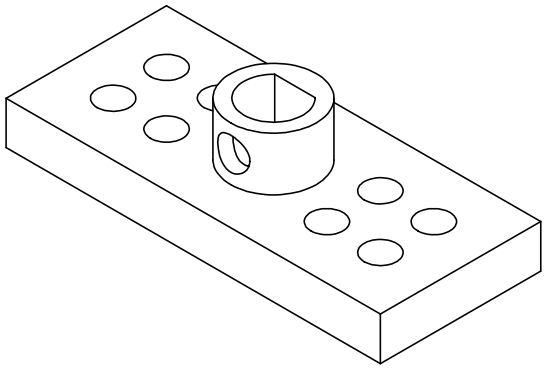
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

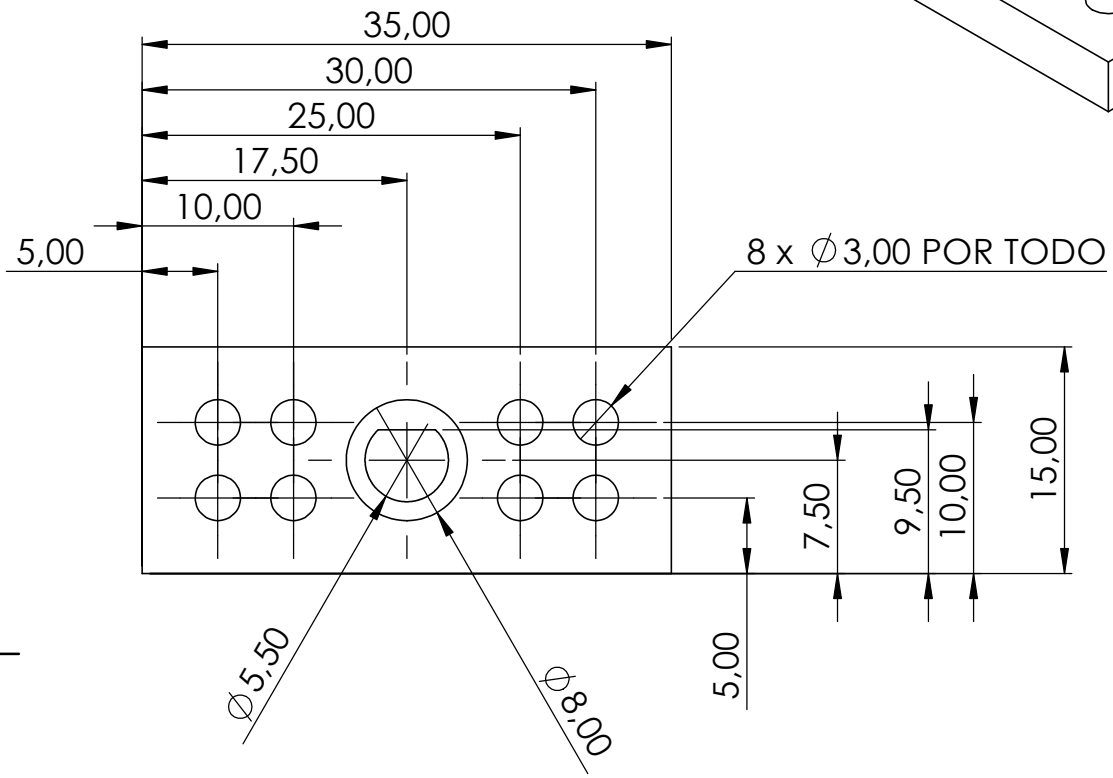
ESCOLA TÈCNICA SUPERIOR ENGINYERIA INDUSTRIAL VALÈNCIA

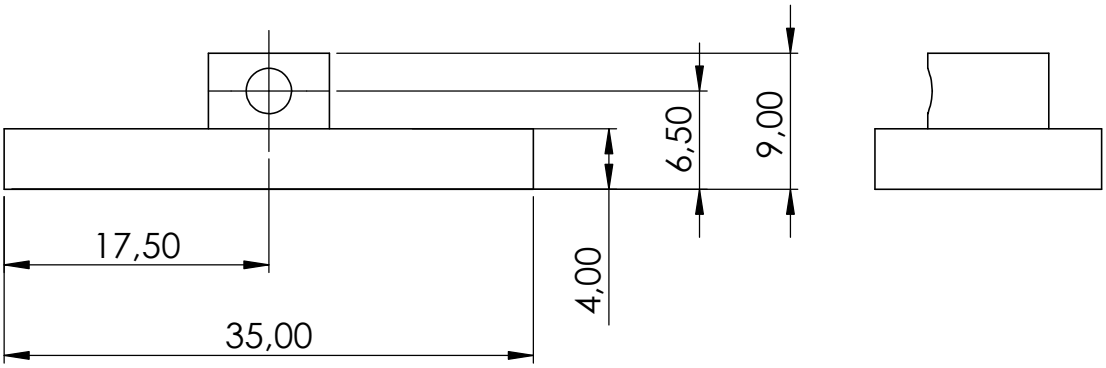
ÁNGEL ESPÍN CARO
Autor

Proyecto:
DISEÑO DE UN SISTEMA DE POSICIONAMIENTO DE DRONES MEDIANTE LIDAR Y RASPBERRY PI

Fecha:	**Escala:**
10/02/2020	1:1
Plano:	**Nº Plano:**
Base torreta	1







TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCOLA TÈCNICA SUPERIOR ENGINYERIA INDUSTRIAL VALÈNCIA

ÁNGEL ESPÍN CARO
Autor

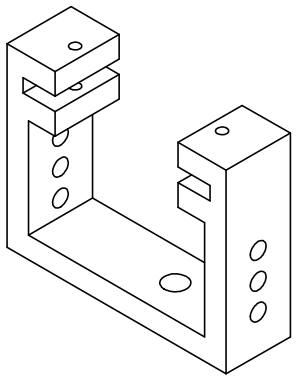
Proyecto:
DISEÑO DE UN SISTEMA DE POSICIONAMIENTO DE DRONES MEDIANTE LIDAR Y RASPBERRY PI

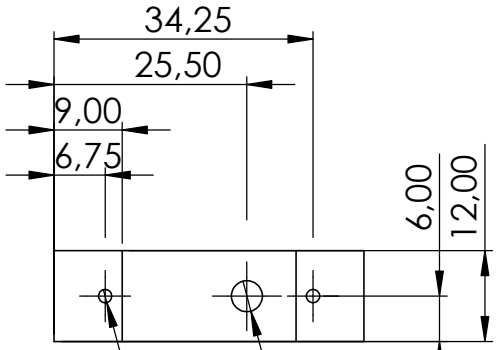
Fecha: 10/02/2020

Escala: 2:1

Plano: Cabeza motor paso a paso

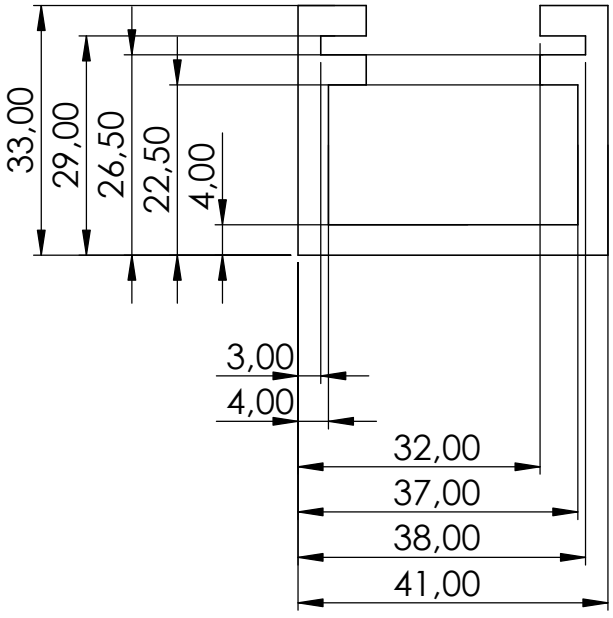
Nº Plano: 2



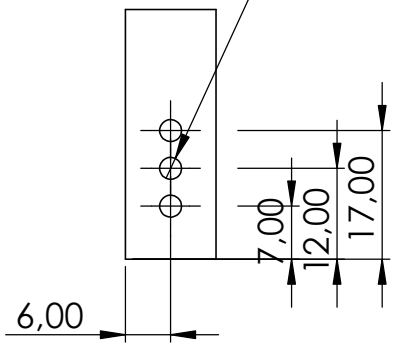


4 x ∅1,75 POR TODO

∅4,10 POR TODO



6 x ∅2,90 POR TODO



TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCOLA TÈCNICA SUPERIOR ENGINYERIA INDUSTRIAL VALÈNCIA

ÁNGEL ESPÍN CARO
Autor

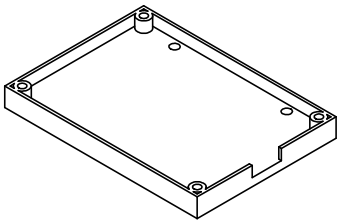
Proyecto:
DISEÑO DE UN SISTEMA DE POSICIONAMIENTO DE DRONES MEDIANTE LIDAR Y RASPBERRY PI

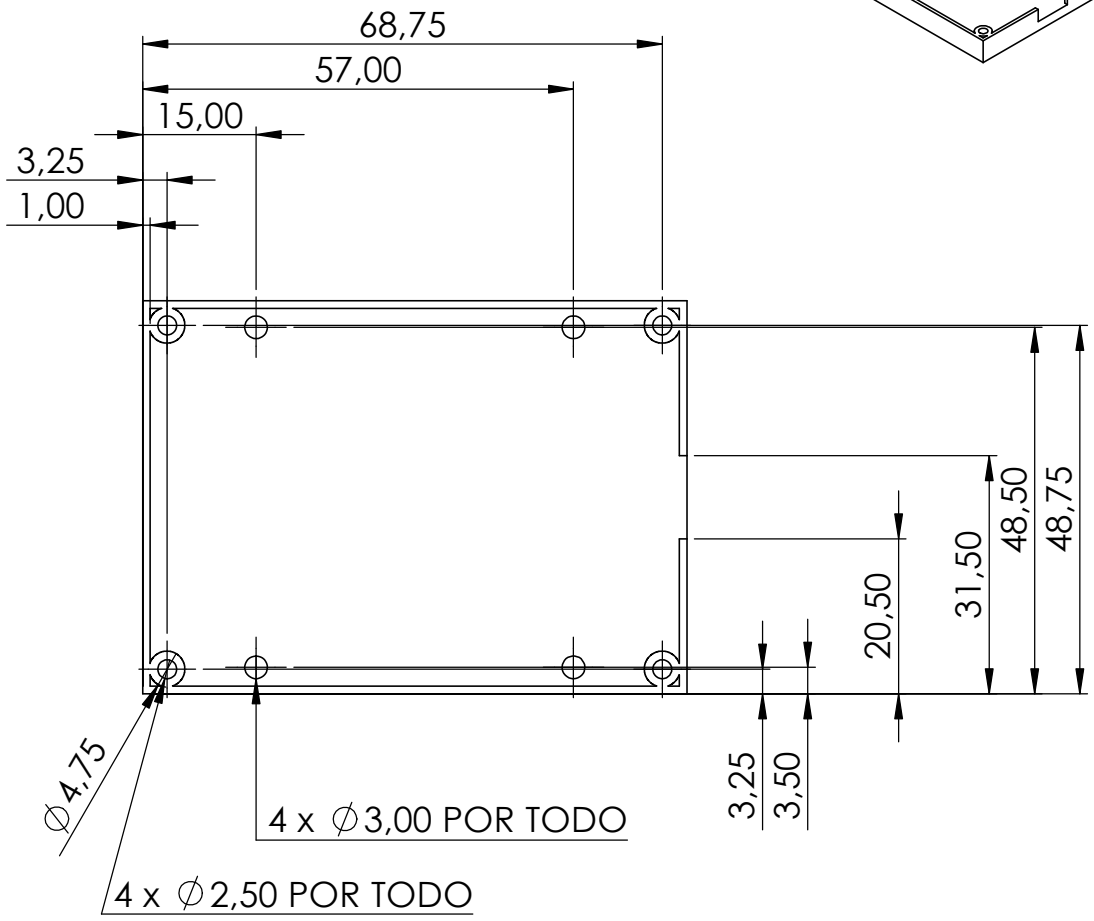
Fecha: 10/02/2020

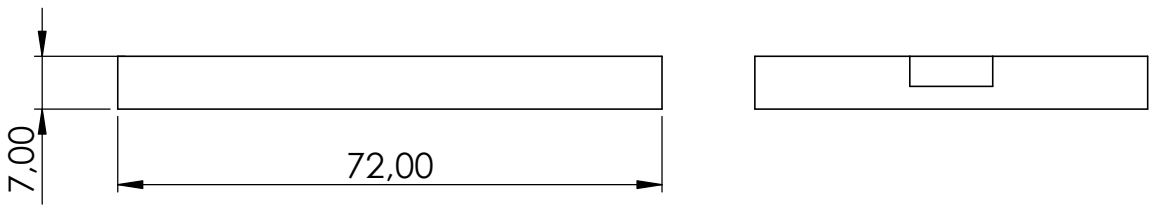
Escala: 1:1

Plano: Correa interior SG90

Nº Plano: 3







TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCOLA TÈCNICA SUPERIOR ENGINYERIA INDUSTRIAL VALÈNCIA

ÁNGEL ESPÍN CARO
Autor

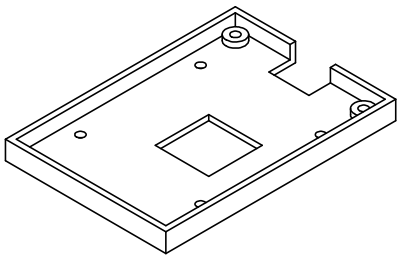
Proyecto:
DISEÑO DE UN SISTEMA DE POSICIONAMIENTO DE DRONES MEDIANTE LIDAR Y RASPBERRY PI

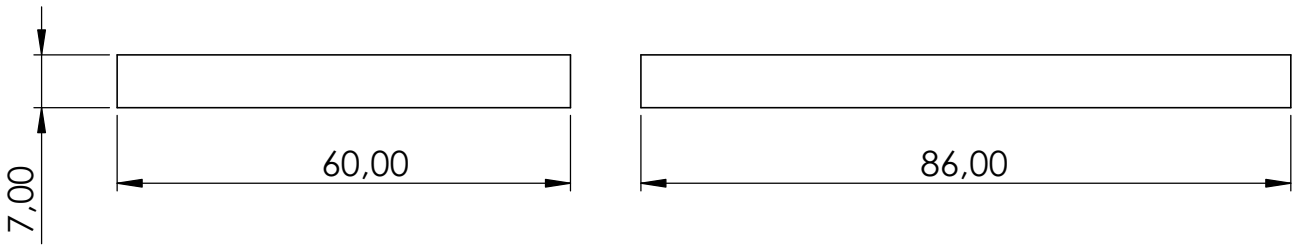
Fecha: 10/02/2020

Escala: 1:2

Plano: Cubierta de PCB de alimentación

Nº Plano: 4





TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

UNIVERSITAT POLITÈCNICA DE VALÈNCIA — ESCOLA TÈCNICA SUPERIOR ENGINYERIA INDUSTRIAL VALÈNCIA

ÁNGEL ESPÍN CARO
Autor

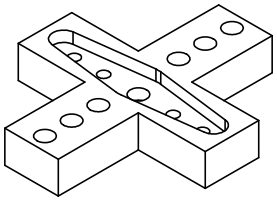
Proyecto:
DISEÑO DE UN SISTEMA DE POSICIONAMIENTO DE DRONES MEDIANTE LIDAR Y RASPBERRY PI

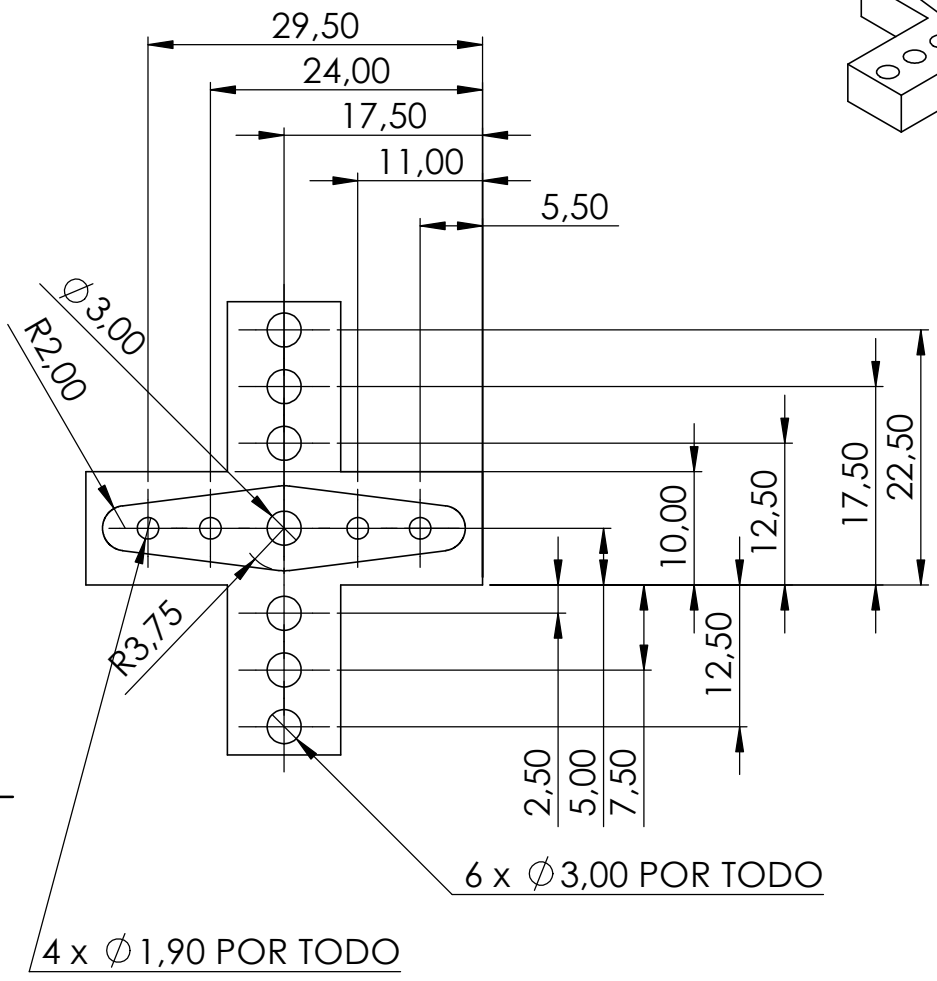
Fecha: 10/02/2020

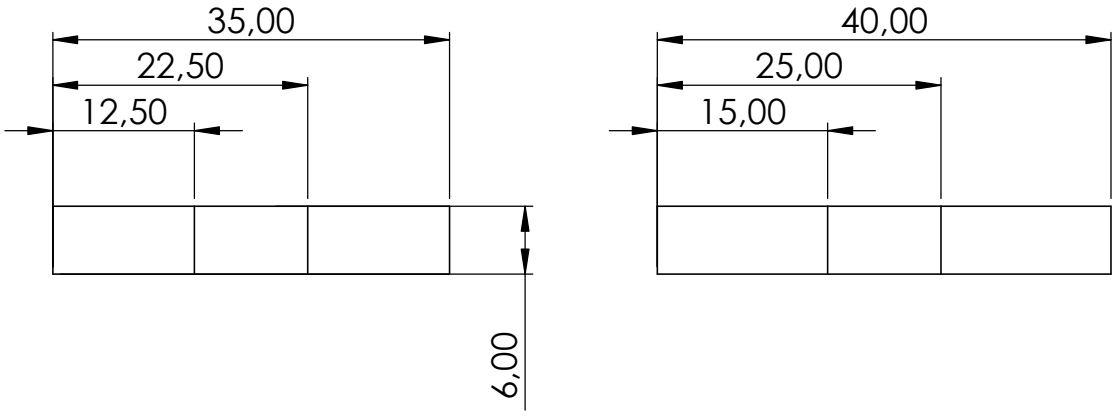
Escala: 1:1

Plano: Cubierta de la PCB del piloto

Nº Plano: 5







TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

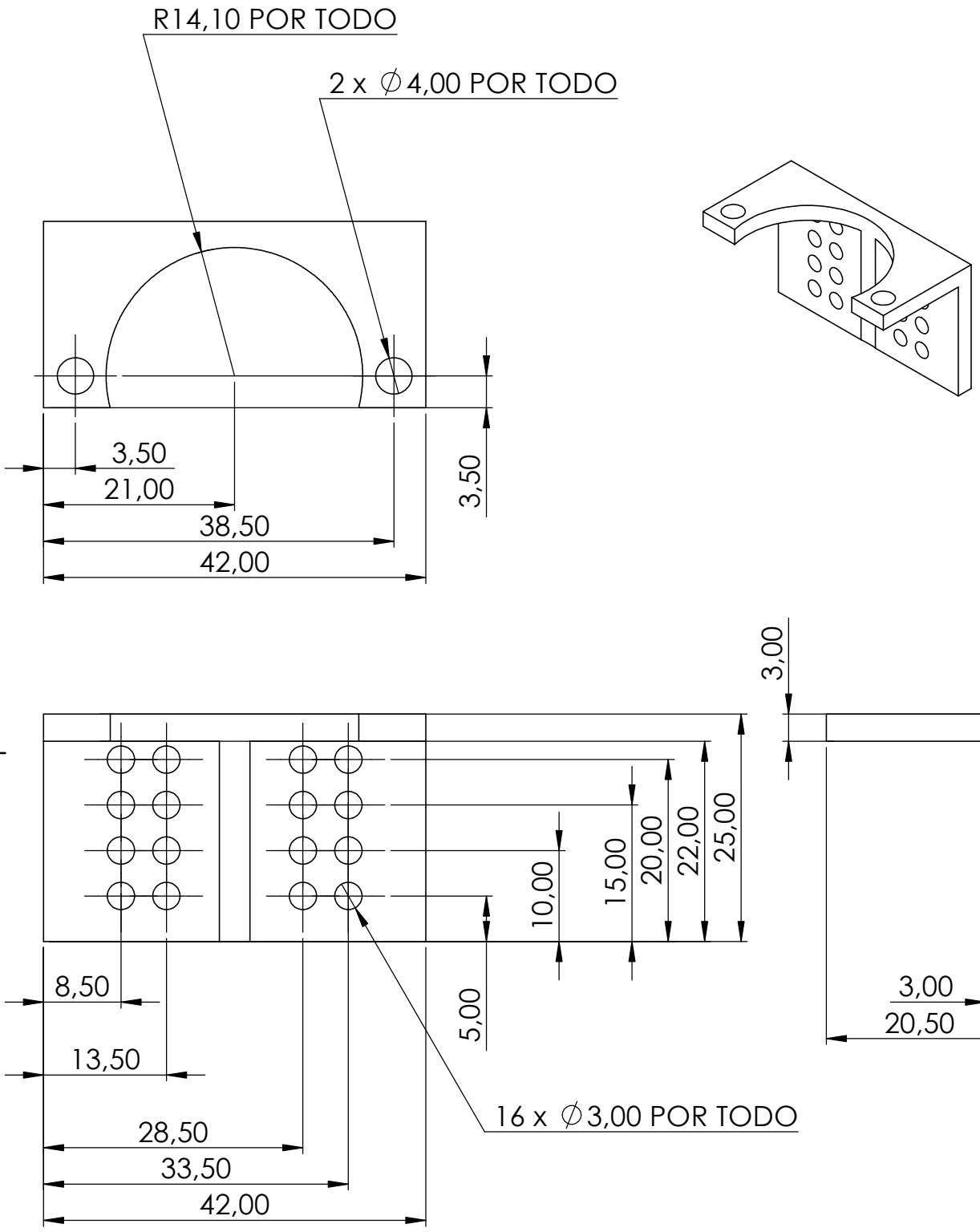
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCOLA TÈCNICA SUPERIOR ENGINYERIA INDUSTRIAL VALÈNCIA

ÁNGEL ESPÍN CARO
Autor

Proyecto:
DISEÑO DE UN SISTEMA DE POSICIONAMIENTO DE DRONES MEDIANTE LIDAR Y RASPBERRY PI

Fecha:	**Escala:**
10/02/2020	3:2
Plano:	**Nº Plano:**
Cabeza SG90	6



TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCOLA TÈCNICA SUPERIOR ENGINYERIA INDUSTRIAL VALÈNCIA

ÁNGEL ESPÍN CARO
Autor

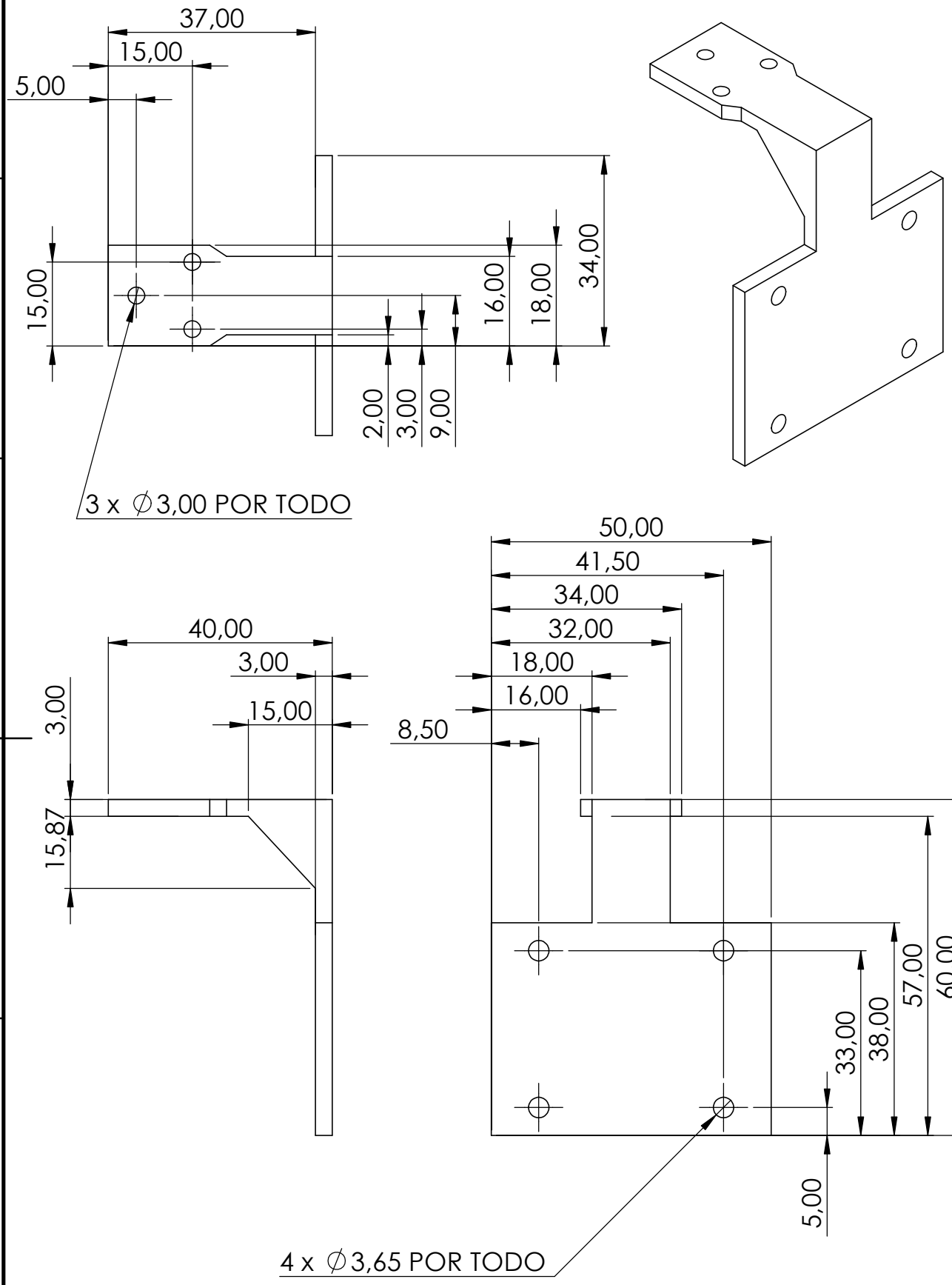
Proyecto:
DISEÑO DE UN SISTEMA DE POSICIONAMIENTO DE DRONES MEDIANTE LIDAR Y RASPBERRY PI

Fecha: 10/02/2020

Escala: 3:2

Plano: Soporte motor paso a paso

Nº Plano: 7



TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCOLA TÈCNICA SUPERIOR ENGINYERIA INDUSTRIAL VALÈNCIA

ÁNGEL ESPÍN CARO
Autor

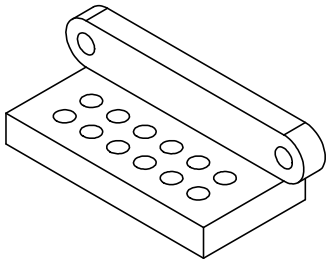
Proyecto:
DISEÑO DE UN SISTEMA DE POSICIONAMIENTO DE DRONES MEDIANTE LIDAR Y RASPBERRY PI

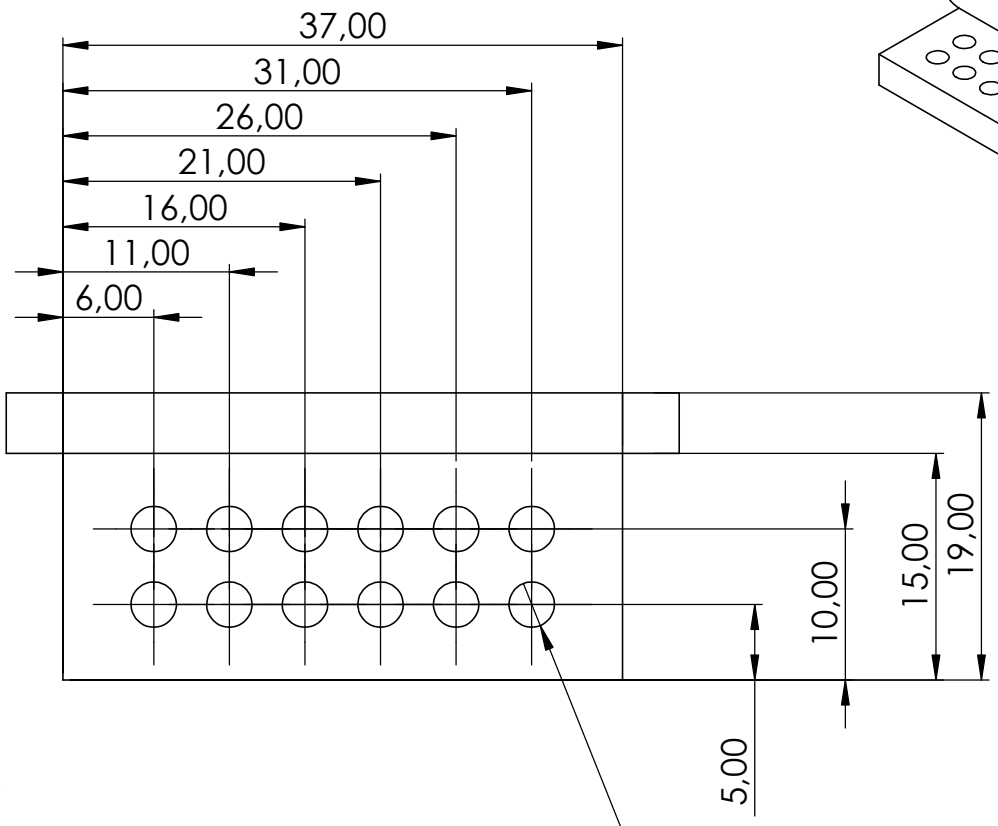
Fecha: 10/02/2020

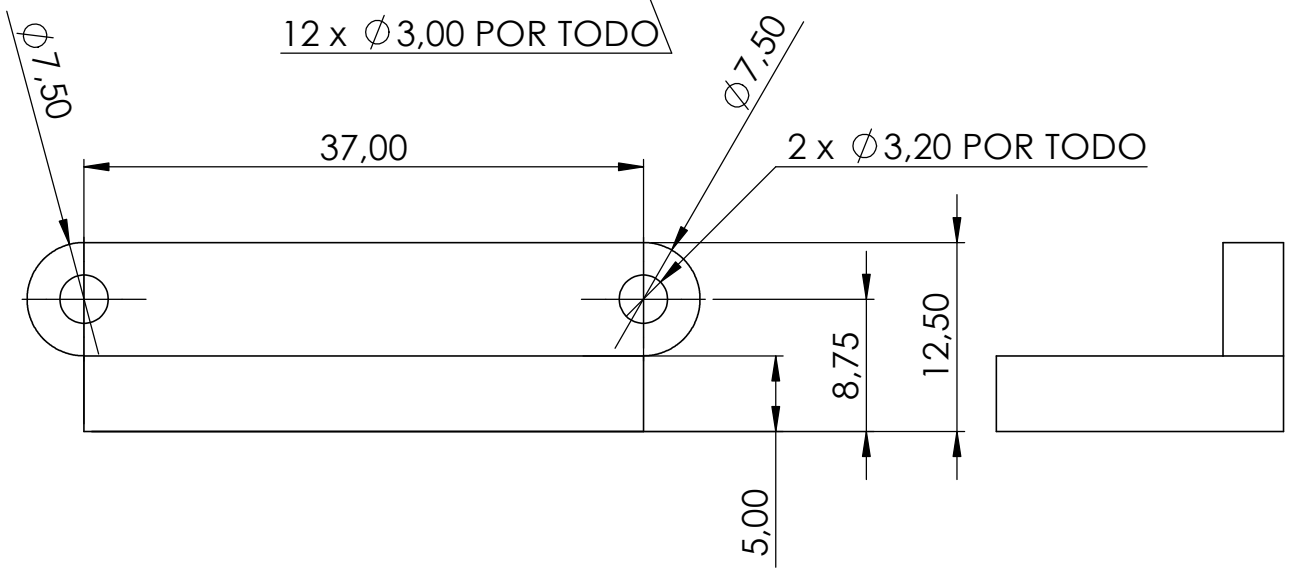
Escala: 1:1

Plano: Soporte acceroios

Nº Plano: 8







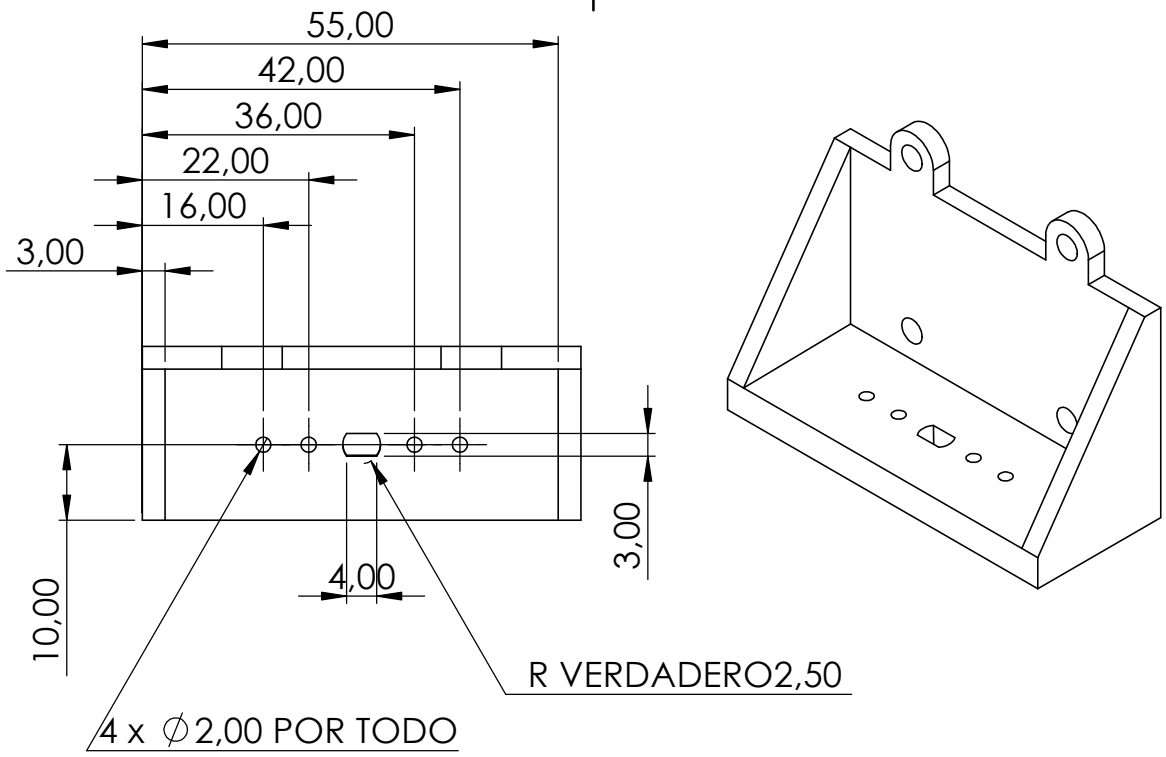
TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

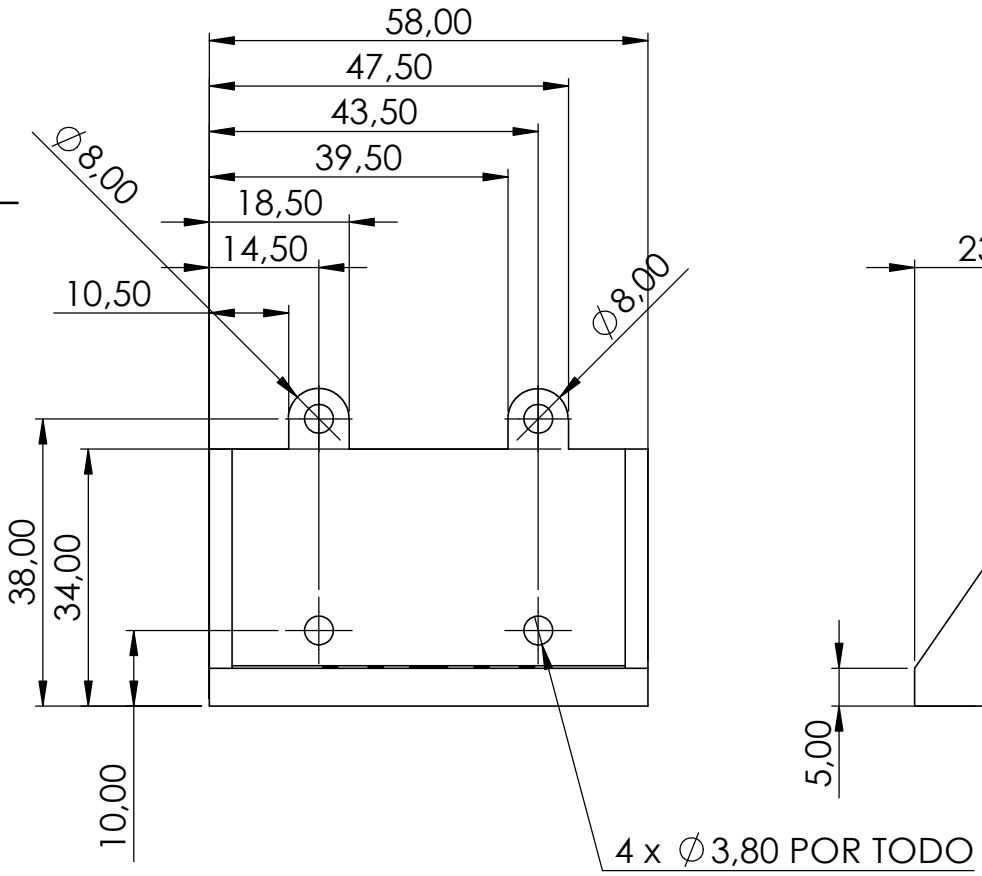
UNIVERSITAT POLITÈCNICA DE VALÈNCIA — ESCOLA TÈCNICA SUPERIOR ENGINYERIA INDUSTRIAL VALÈNCIA

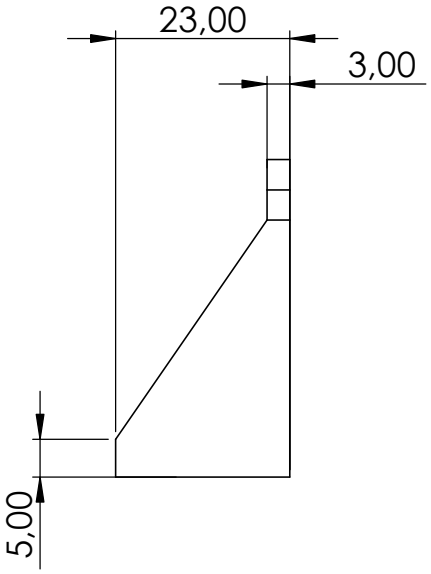
ÁNGEL ESPÍN CARO
Autor

Proyecto:
DISEÑO DE UN SISTEMA DE POSICIONAMIENTO DE DRONES MEDIANTE LIDAR Y RASPBERRY PI

Fecha:	**Escala:**
10/02/2020	2:1
Plano:	**Nº Plano:**
Soporte Infrarojos	9







TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

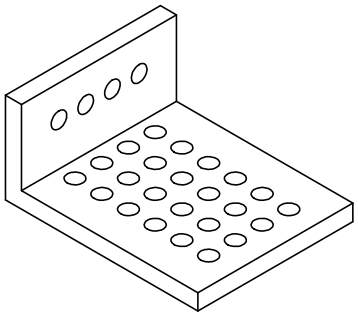
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

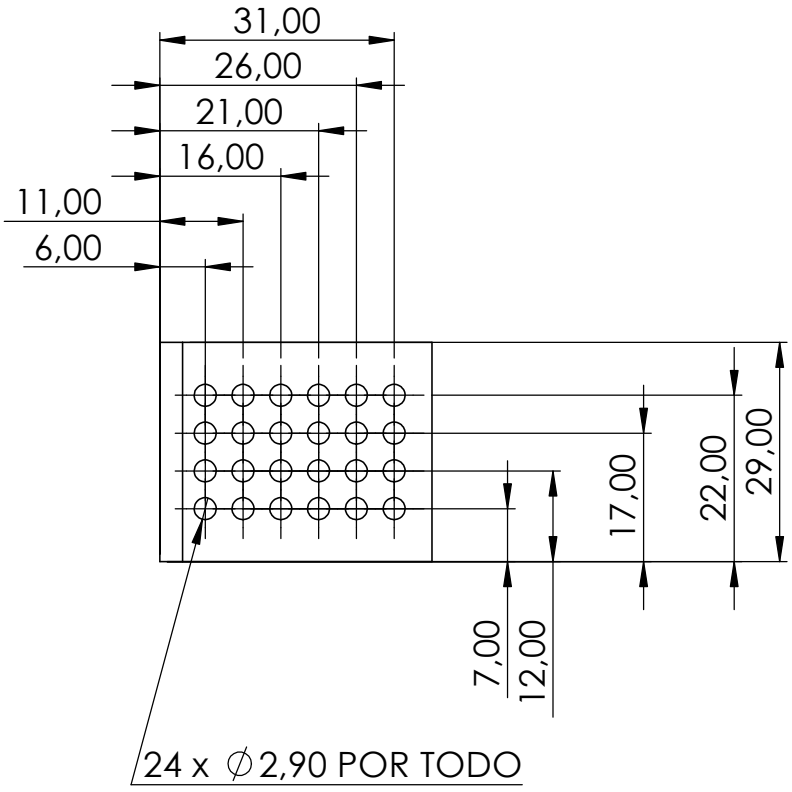
ESCOLA TÈCNICA SUPERIOR ENGINYERIA INDUSTRIAL VALÈNCIA

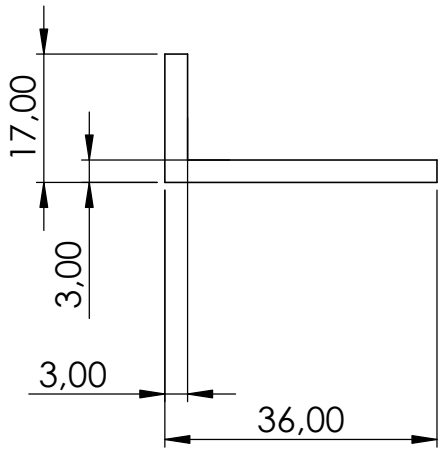
ÁNGEL ESPÍN CARO
Autor

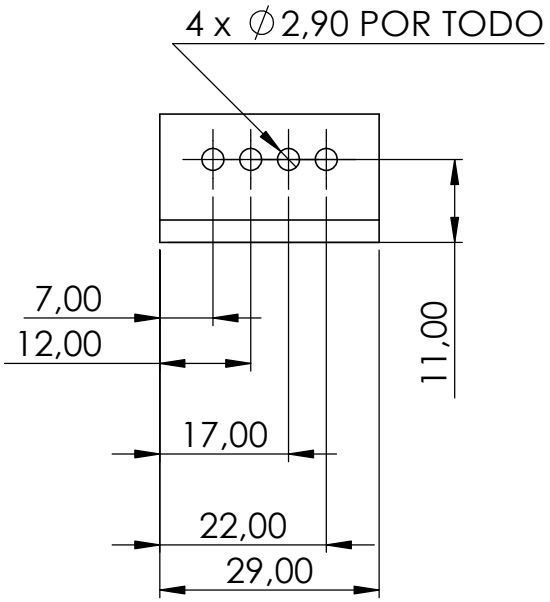
Proyecto:
DISEÑO DE UN SISTEMA DE POSICIONAMIENTO DE DRONES MEDIANTE LIDAR Y RASPBERRY PI

Fecha:	**Escala:**
10/02/2020	1:1
Plano:	**Nº Plano:**
Soporte lidar	10









TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

UNIVERSITAT POLITÈCNICA DE VALÈNCIA — ESCOLA TÈCNICA SUPERIOR ENGINYERIA INDUSTRIAL VALÈNCIA

ÁNGEL ESPÍN CARO
Autor

Proyecto:
DISEÑO DE UN SISTEMA DE POSICIONAMIENTO DE DRONES MEDIANTE LIDAR Y RASPBERRY PI

Fecha:	**Escala:**
10/02/2020	1:1
Plano:	**Nº Plano:**
Amarre	11