Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADO EN INGENIERÍA INFORMÁTOCA DE GESTIÓN Y SISTEMAS DE INFORMACIÓN

TRABAJO FIN DE GRADO

SISTEMA EXTERNO DE PLANIFICACIÓN Y CONTROL DE VUELO AUTÓNOMO PARA UAV

MEMORIA DEL PROYECTO

Alumno: Martín, Celis, Nicolás

Director: Bilbao, Landatxe, Javier

Curso: 2020-2021

Fecha: 23 de Julio de 2021

Resumen

Este proyecto consiste en la creación de un sistema de programación de vuelos para un dron no tripulado. El sistema tiene que permitir que los vuelos se puedan guardar para su uso posterior.

Las instrucciones del vuelo son lo mas simples posibles y permiten al usuario tener control total del dron. Además, tiene la capacidad de evitar colisionar con objetos en su trayectoria por medio algún tipo de sensor.

Es un sistema capaz de conectarse a múltiples drones independientemente de las características de los mismos. Por esto, el sistema es configurable y permite ajustar valores como el número de canales o la potencia máxima de los mismos. También permite definir los valores por defecto de los canales.

Se ha buscado el reto de diseñar un sistema más barato y más adaptable que el que ofrecen los dispositivos disponibles actualmente en el mercado para el autopilotaje.

Laburpena

Proiektu hau drone ez-tripulatu batentzako hegaldiak antolatzeko sistema sortzean datza. Sistema honek hegaldiak geroko erabilerarako gordetzeko aukera eman behar du.

Hegaldiaren jarraibideak ahalik eta sinpleenak dira eta erabiltzaileak dronearen kontrol osoa izatea ahalbideratzen du. Gainera, bere ibilbidean objektuekin talka egitea saihesteko gai da sentsore sistema baten bidez.

Drone ugarirekin konektatzeko gai den sistema bat da, dronen ezaugarriak edozein izanik ere. Horregatik, sistema konfiguragarria da eta kanalen kopurua edo gehieneko potentzia bezalako balioak doitzeko aukera ematen du. Horrez gain, kanalen balio lehenetsiak definitzea ahalbideratzen du.

Gaur egun merkatuan autopilotajerako eskuragarri dauden gailuek eskaintzen dutena baino sistema merkeagoa eta moldagarriagoa diseinatzea bilatu egin da.

Summary

This project consists of the creation of a flight programing system for a non-manned drone. This system must be able of saving flights for later use.

The instructions are as simple as possible, while permitting a complete control of the drone to the user. Moreover, it is able of preventing collisions with objects on its trajectory by having some type of sensor.

The system can be connected to multiple drones with different characteristics. Therefore, the system is configurable and permits adjusting configuration values, such as channels or maximum power of each channel. It also permits adjusting the default values of each channel.

The challenge has been making a system cheaper and more adaptable than the ones a user can buy nowadays.

Palabras calve

PWM, dron, Autopilotaje, SPI, Arduino, Raspberryu Controlador de vuelo.

Índice de cuadros

Índice de figuras

Índice

1. Glosario

Antes de comenzar con la explicación del proyecto, se definen algunos de los términos utilizados a lo largo de mismo:

- **Pulse Width Modulation (PWM):** es un método de modulación de señales por el cual se puede trasmitir información de un valor analógico. En el caso de los transmisores de radio para los drones este valor estará entre de 0 a 1024. Funciona por medio de generar un pulso de corriente de duración ente uno y dos milisegundos a intervalos regulares. La duración del pulso indica el valor de salida.

- **Pulse Position Modulation (PWM):** se trata de otro método de modulación de señal. En este caso la duración del pulso es la siempre la misma, pero la separación entre pulsos varía. Es esta separación indica el valor de salida. A diferencia de PWM, que necesita un cable para cada canal, este método permite enviar múltiples canales por el mismo cable.

- **Electronic Speed Controler (ESC):** es una pieza de electrónica a la que se referencia como ESC. Esta pieza se encarga de alimentar los motores de acuerdo a la potencia deseada y regular la velocidad.

- **Serial Peripheral Interface (SPI):** se trata de un método de transferencia de información. Este método requiere de un dispositivo que envié información y de tantos dispositivos que la reciban como se desee. Al ser una comunicación entre dos dispositivos, uno de los cables es el chip select, que activa el dispositivo elegido. La transferencia de información de se hace por un cable distinto que la recepción de la misma, evitando así errores.

- **Universal Asynchronous Receiver-Transmitter (UART):** es otro método de transferencia de información, en este caso sólo cuenta con un cable de transmisión y uno de recepción. La comunicación por este método sólo puede llevarse a cabo ser entre dos dispositivos.

- **I2C:** se trata de un protocolo de comunicación serial. El puerto incluye dos cables de comunicación: SDA y SCL. En ocasiones produce errores ya que la transmisión y recepción de información se realizan por el mismo cable (SDA). SCL sólo es el reloj para sincronizar ambos dispositivos.

2. Introducción

En la última década la venta de drones se ha disparado tanto en el sector profesional como en el de ocio. Este incremento ha traído consigo una reducción de los precios bastante considerable. Pero aun hoy en día no es tan económico conseguir un dron programable.

Antes de continuar con la introducción, respondamos a la pregunta ¿qué es un dron? Un dron es cualquier vehículo capaz de desplazarse por el aire sin ningún tripulante.

Definido qué es un dron, podemos apreciar la importancia que tiene hoy en día, desde su uso privado como entretenimiento o profesional (por ejemplo para realizar fotografías y vídeos desde el cielo) hasta su uso militar (frecuentemente para conseguir imágenes en lugares de difícil acceso). Fotografías y vídeos desde el aire, hasta acceder a lugares de difícil acceso. Esbozando la enorme variación de usos, podemos entender también que el espectro de precios es muy amplio.
You: Los drones programables ya existen, obviamente son más caros que los drones que se manejan exclusivamente con mando a distancia. Los drones que se pilotan exclusivamente con mando a distancia contienen un controlador de vuelo no programable.

También existe la posibilidad de sustituir un controlador no programable por un controlador de vuelo programable, como se explica más adelante con más detalle. La posibilidad actual que ofrece el mercado es la de incorporar un controlador de vuelo programable que, por su precio y su peso, no constituye una opción viable para muchos drones. Convertir un dron no programable en un dron que pueda programarse y ejecutar vuelos autopilotados supone actualmente, por costo económico, dificultades técnicas y de adaptabilidad, un reto informático y una oportunidad para solucionar un problema real. No es raro que actuales

usuarios de drones quieran mejorar su dron para que se pueda manejar de manera autónoma o programar un vuelo con el. Es aquí donde entra en escena el proyecto.
La idea principal sobre la que se va a trabajar es diseñar un sistema totalmente independiente del dron, lo más ligero posible, que el usuario podrá conectar sustituyendo su emisora de radio para programar vuelos. Con esto se da la gran ventaja de poder mejorar los drones incorporando elementos muy ligeros, como se verá más adelante, sin necesidad de cambiar piezas.

2.1. Descripción y situación del proyecto

Hemos de remarcar que este es un proyecto en el que se ha comenzado desde cero, con la intención de implementar todas las funcionalidades básicas y el máximo de las funciones de confort o extras.

Con respecto a la situación de mercado, hay dos opciones principales a la hora de comprar un dron, montado o por piezas. En el caso de los drones que ya vienen montados, resultan de interés los programables, cuyo precio mínimo ronda lo 700 euros. En el caso de drones que se adquieren por piezas el precio a tener en cuenta para este proyecto es el del controlador de vuelo, que cuesta entorno a 180 euros si es programable frente a los 40 o 50 si no es programable.

Conociendo los datos económicos se puede apreciar que el margen es amplio y que lograr diseñar un sistema por debajo de los 100 euros proporcionaría un producto competitivo. Por lo que se ha establecido ese límite económico para el desarrollo del proyecto.

2.2. Motivos para la elección del proyecto

Los motivos que han impulsado la elección de la presente propuesta son los siguientes:

- **Utilidad del proyecto**:

  Como usuario de un dron e interesado por los sistemas aeronáuticos, me he planteado las dificultades que supone convertir un dron en programable con los programadores de vuelo que ofrece el mercado. Adquirir un dron programable es caro para su uso en ocio y también lo es para un uso profesional en que puedan requerirse diversos aparatos. Diseñar un sistema más ligero, adaptable y barato que pueda incorporarse a distintos drones, supone un reto con fines prácticos y que puede ser interesante para distinto perfil de usuarios.

- **Posibilidad de aprendizaje**:

  Durante esta carrera, no hemos tenido ocasión de manejar y experimentar con dispositivos electrónicos, y el poco contacto que hemos tenido me ha interesado mucho. Por este motivo considero que diseñar este sistema me ofrece una gran oportunidad para aprender más sobre electrónica y señales.

3. Planteamiento

En este apartado se explica el desarrollo tanto de la idea como de su puesta en marcha. Detallando qué herramientas se han utilizado, los objetivos, el diseño y la elección del hardware entre las opciones disponibles.

3.1. Objetivos del proyecto

Como ya hemos mencionado previamente, el proyecto consiste en diseñar un sistema que permita al usuario programar un vuelo en su dron. Para alcanzar esta meta se requiere lograr los siguientes objetivos:

- Programar un sistema capaz de manejar un dron guiándose por las instrucciones registradas por el usuario.
- Conseguir un sistema versátil y válido para múltiples drones, de distintas especificaciones.
- Construir un código capaz de hacer tanto tareas sencillas como complejas.
- Conseguir un código robusto capaz de afrontar imprevistos a la hora del vuelo.

Además de los objetivos esenciales del proyecto, existen también objetivos secundarios. Estos objetivos no son estrictamente necesarios pero sería ideal alcanzarlos:

- Lograr un sistema que permita que se pueda retomar el control del vuelo en cualquier momento.
- Lograr un sistema que permita elegir un modo de vuelo en interior, añadiendo nuevos sensores.
- Implementar sistemas de seguridad en caso de fallo.

3.2. Herramientas

Con el propósito de alcanzar la meta se utilizan múltiples herramientas. Estas herramientas se pueden dividir en tres secciones:

- **Gestión y orden**: estas son las herramientas que permitirán mantener los distintos componentes del proyecto ordenados.
  - **Github**: Con el fin de mantener el código ordenado y accesible desde cualquier parte.
  - **Office 365**: Necesario para mantener los documentos ordenados y estructurados, en especial Word y Excel.
  - **Google drive**: Para guardar los fichero y evitar perdidas.
- **Desarrollo**: serán las herramientas que se emplearán a la hora de implementar el código.
  - **Pycharm**: Compilador y editor de Python.
  - **Visual Studio Code**: Es un compilador muy versátil que se empleará en caso de necesitar algún otro lenguaje durante el proyecto.
- **Documentación**: Con estas herramientas se generarán los documentos que se requieran.
  - **Gantt**: Para generar los diagramas de Gantt.
  - **LATEX**: Más concretamente Overleaf, un compilador de Latex en la nube para generar PDF.

3.3. Riesgos

En el desarrollo del proyecto también se ha llevado a cabo una previsión de los problemas que pueden surgir y un propuesta de cómo evitarlos.

- **Rotura de alguna de las piezas:** con el fin de evitarlo se adquirirán las piezas en pares, así aun que una de las mismas se rompa siempre se dispondrá de una funcional.
- **Voltajes excesivos:** algunas de las piezas con las que se trabaja no soportan los 5V de arduino, es por esto que se adquirirán conversores a 3,3V para evitar incidentes.
- **Código difícil de comprender:** con el fin de evitar no olvidarse de como funciona el código, se añadirán comentarios y se le darán a las variables nombres significativos.
- **Pérdida de archivos o información:** para evitar esto todo el código se guardará en Github y los documentos en Google drive.
- **Mala planificación y retrasos:** es importante llevar el proyecto a buen ritmo para que el resultado final sea satisfactorio. Por esto se invertirán unos días para planear plazos y horas a dedicar en cada sección, evitando así imprevistos.
- **Datos erróneos o insuficientes:** para evitar que las pruebas no sean suficientes o los datos no sean representativos, se realizarán múltiples y variadas pruebas.

3.4. Estructura de descomposición del trabajo

Tras analizar el proyecto se ha concretado la siguiente estructura de trabajo:

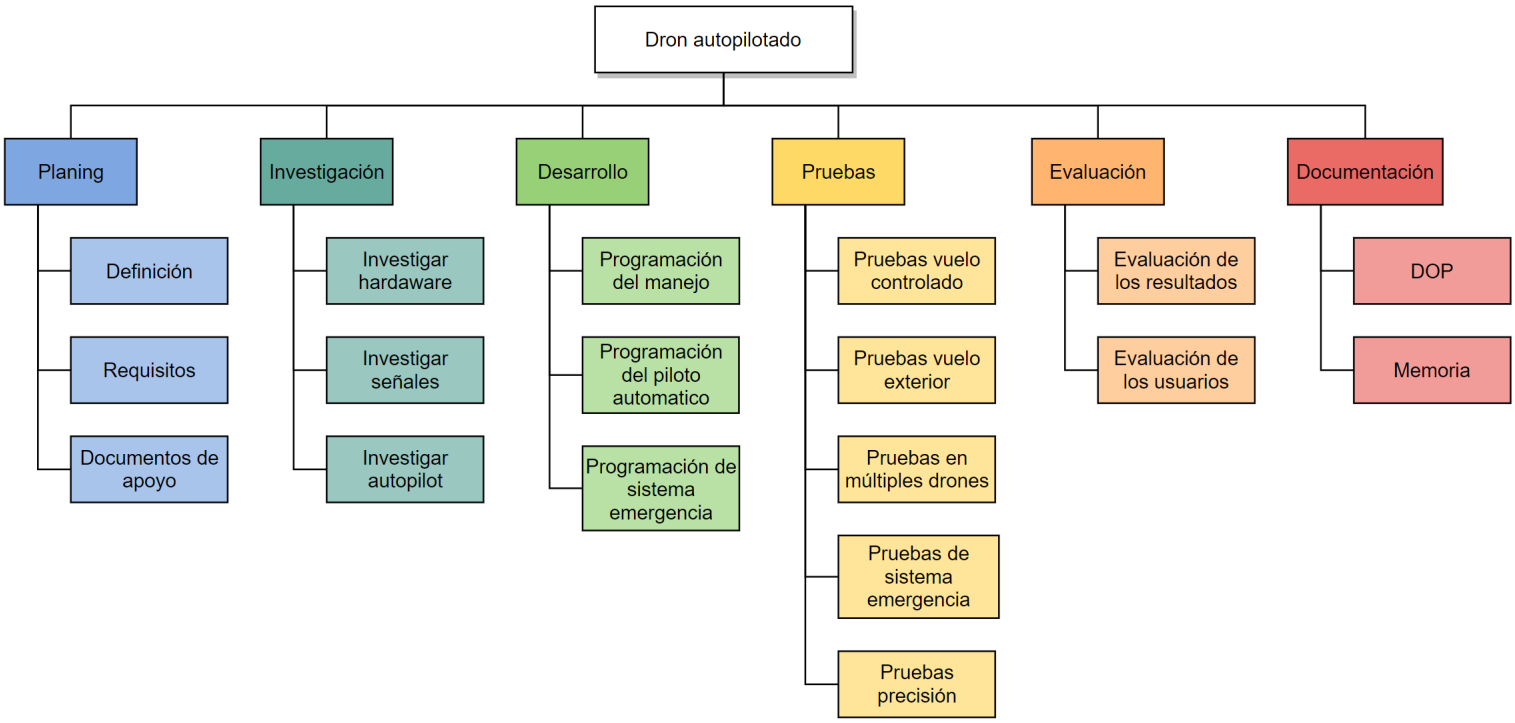

Figura 1: Estructura de descomposición del trabajo

3.5. Descripción de los paquetes de trabajo

Ahora que ya se conocen los paquetes de trabajo, se presenta una descripción más detallada de cada uno de ellos:

1.1 Definición
Descripción: Consiste en definir el proyecto, sus especificaciones, la idea y sus límites **Recursos necesarios:** Office365 (Word y Excel) **Precedencias:** Ninguno
1.2 Requisitos
Descripción: Analizar qué elementos se van a necesitar para el desarrollo del proyecto de manera eficiente **Recursos necesarios:** Office365 (Word y Excel) **Precedencias:** 1.1 Definición

1.3 Documentos de apoyo
Descripción: La creación de todos los documentos que se generan como parte de la realización de este proyecto **Recursos necesarios:** Office365 (Word y Excel) **Precedencias:** 1.1 Definición, 1.2 Requisitos
2.1 Investigar Hardware
Descripción: Como parte fundamental de este proyecto se requiere un sistema capaz de controlar un dron, para ello se procederá a localizar el mejor de los sistemas, analizarlo y evaluar sus especificaciones físicas **Recursos necesarios:** Internet **Precedencias:** 1.1 Definición, 1.2 Requisitos

2.2 Investigar Señales
Descripción: Como parte esencial del manejo del dron, las señales y su trasmisión son cruciales, esta tarea tiene como objetivo la obtención de los conocimientos necesarios para su manejo de forma correcta. **Recursos necesarios:** Internet **Precedencias:** 1.1 Definición, 1.2 Requisitos
2.3 Investigar autopiloto
Descripción: Búsqueda de sistemas similares e ideas de funcionamiento. Estudiar distintas formas de diseñar el programa y determinar la mejor de ellas con el fin de prevenir cambios mayores en la implantación. **Recursos necesarios:** Internet **Precedencias:** 1.1 Definición, 1.2 Requisitos

3.1 Programación del manejo
Descripción: Programar el controlador utilizado con el fin de que sea capaz de manejar el dron. Que sea capaz de generar las señales necesarias para hacer que se mueva. **Recursos necesarios:** Visual Studio, Pycharm, GitHub **Precedencias:** 1.1 Definición, 1.2 Requisitos, 2.1Investigar hardware ,2.2 Investigar señales, 2.3 Investigar autopilot
3.2 Programación del piloto automático
Descripción: Programar el controlador utilizado con el fin de que sea capaz de manejar el dron siguiendo unas instrucciones indicadas con anterioridad y guardadas en la memoria. **Recursos necesarios:** Visual Studio, Pycharm, GitHub **Precedencias:** 1.1 Definición, 1.2 Requisitos, 2.1 Investigar hardware ,2.2 Investigar señales, 2.3 Investigar autopilot

3.3 Programación del sistema de emergencia
Descripción: Con el fin de prevenir accidentes, en caso de que se pierda el control del dron, implementar un sistema que lo aterrice de manera automática o lo devuelva a la posición original y pase a control manual. **Recursos necesarios:** Visual Studio, Pycharm, GitHub **Precedencias:** 1.1 Definición, 1.2 Requisitos, 2.1 Investigar hardware ,2.2 Investigar señales, 2.3 Investigar autopilot
4.1 Pruebas de vuelo controlado
Descripción: Las pruebas del vuelo del dron en un entorno cerrado o con poco viento para ver como se comporta en condiciones ideales. **Recursos necesarios:** Visual Studio, Pycharm, GitHub **Precedencias:** 1.1 Definición, 1.2 Requisitos 2.1 Investigar hardware 2.2 Investigar señales, 2.3 Investigar autopilot, 3.1 Programación manejo, 3.2 Programación del piloto automático, 3.3 Programación sistema de emergencia

4.2 Pruebas vuelo exterior
Descripción: Las pruebas del vuelo del dron en un entorno abierto en condiciones menos ideales. **Recursos necesarios:** Visual Studio, Pycharm, GitHub **Precedencias:** 1.1 Definición, 1.2 Requisitos, 2.1Investigar hardware ,2.2 Investigar señales, 2.3 Investigar autopilot, 3.1 Programación manejo, 3.2 Programación del piloto automático, 3.3 Programación sistema de emergencia
4.3 Pruebas múltiples drones
Descripción: Probar si el sistema es suficientemente modular como para instalarlo en otro dron. **Recursos necesarios:** Dron **Precedencias:** 1.1 Definición, 1.2 Requisitos, 2.1 Investigar hardware ,2.2 Investigar señales, 2.3 Investigar autopilot, 3.1 Programación manejo, 3.2 Programación del piloto automático, 3.3 Programación sistema de emergencia

4.4 Pruebas sistema emergencia
Descripción: Probar el sistema de emergencia para verificar que en caso de fallo todo funcione correctamente. **Recursos necesarios:** Dron **Precedencias:** 1.1 Definición, 1.2 Requisitos, 2.1 Investigar hardware ,2.2 Investigar señales, 2.3 Investigar autopilot, 3.1 Programación manejo, 3.2 Programación del piloto automático, 3.3 Programación sistema de emergencia
4.5 Pruebas de precisión
Descripción: Probar la precisión del dron a la hora de ejecutar las instrucciones. **Recursos necesarios:** Dron **Precedencias:** 1.1 Definición, 1.2 Requisitos, 2.1 Investigar hardware, 2.2 Investigar señales, 2.3 Investigar autopilot, 3.1 Programación manejo, 3.2 Programación del piloto automático, 3.3 Programación sistema de emergencia

5.1 Evaluación de los resultados

Descripción:
Diseñar una evaluación de los resultados de las pruebas y de la viabilidad en las distintas aplicaciones de los drones.

Recursos necesarios:
Dron, Office 365

Precedencias:
1.1 Definición, 1.2 Requisitos, 2.1 Investigar hardware, 2.2 Investigar señales, 2.3 Investigar autopilot, 3.1 Programación manejo, 3.2 Programación del piloto automático, 3.3 Programación sistema de emergencia, 4.1 Pruebas vuelo controlado, 4.2 Pruebas vuelo exterior, 4.3 Pruebas en múltiples drones, 4.4 Pruebas de sistema de emergencia, 4.5 Pruebas de precisión.

5.2 Evaluación de los usuarios

Descripción:
Crear un documento con las opiniones de los usuarios y estudiar la viabilidad en el mercado.

Recursos necesarios:
Dron, Office 365

Precedencias:
1.1 Definición, 1.2 Requisitos, 2.1 Investigar hardware ,2.2 Investigar señales, 2.3 Investigar autopilot, 3.1 Programación manejo, 3.2 Programación del piloto automático, 3.3 Programación sistema de emergencia, 4.1 Pruebas vuelo controlado, 4.2 Pruebas vuelo exterior, 4.3 Pruebas en múltiples drones, 4.4 Pruebas de sistema de emergencia, 4.5 Pruebas de precisión.

6.1 DOP

Descripción:
Creación del documento de objetivos del proyecto.

Recursos necesarios:
Office365

Precedencias:
1.1 Definición, 1.2 Requisitos

6.2 Memoria
Descripción: Creación de la memoria del proyecto. **Recursos necesarios:** Office365, Latex **Precedencias:** Ninguna

Cuadro 1: Tareas descritas detalladamente

3.6. Planificación temporal

Esta es una pequeña tabla con los tiempos estimados para cada parte del proyecto.

Tarea	Esfuerzo en horas	Duración en días
Fase 1: Planificación y diseño	59 horas	180 días
Definición	2 horas	1 día
Requisitos	2 horas	1 día
Documentos de apoyo	20 horas	150 días
DOP	5 horas	3 días
Memoria	30 horas	150 días
Fase 2: Investigación	50 horas	20 días
Investigar Hardware	10 horas	3 días
Investigar Señales	10 horas	3 días
Investigar autopilot	30 horas	20 días
Fase 3: Desarrollo	325 horas	110 días
Programación del manejo	20 horas	15 días
Programación del piloto automático	300 horas	90 días
Programación del sistema de emergencia	5 horas	5 días
Fase 4: Pruebas	30 horas	20 días
Pruebas de vuelo controlado	5 horas	2 días
Pruebas de vuelo exterior	10 horas	10 días
Pruebas en múltiples drones	7 horas	20 días
Pruebas de sistema de emergencia	5 hora	5 días
Pruebas de precisión	3 horas	6 días
Fase 5: Evaluación	15 horas	20 días
Evaluación de los resultados	7.5 horas	10 días
Evaluación de los usuarios	7.5 horas	10 días
Total proyecto	479 horas	180 días

Cuadro 2: Distribución de horas

3.6.1. Diagrama de Gantt

Según las estimaciones anteriores, se prevé finalizar el proyecto la primera quincena de Julio. El siguiente cronograma representa el orden en el que se realizarán las tareas definidas previamente para alcanzar la fecha estimada.

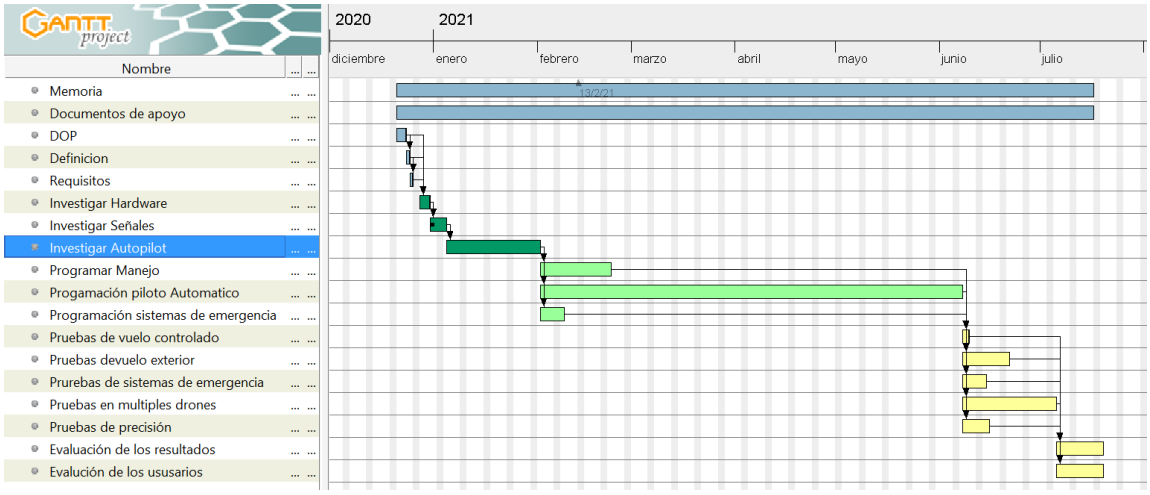

Figura 2: Diagrama de Gantt

4. Presupuesto

Como ya se ha mencionado previamente,la meta es lograr un sistema que pueda implementarse con un coste de adquisición inferior a los 100 euros para que pueda ser comercialmente viable. Además, a este precio hay que sumarle el coste adicional de la primera versión con las horas extra de desarrollo y de investigación que conllevan. El presupuesto queda de la siguiente forma:

	Materiales	Personas	Equipos
Coste desarrollo	120€	3832€	43€
Costes variables	55€/u	4€/u	0.005€/u

Los costes materiales son las piezas de equipo que se han tenido que comprar para la realización del proyecto. Y en costes variables aparece el precio de cada unidad a producir. Es decir, durante el desarrollo se han invertido 120€ en hardware que habría que recuperar.

Además del material empleado para llevar a cabo el desarrollo, se han invertido 479 horas de trabajo. Dado que en España un ingeniero informático junior cobra de media 8 euros la hora, el total suma 3832 euros. También hay que añadir a los costes variables la media hora que deben dedicarse a montar el sistema para poder venderlo.

Por último, hay que valorar el coste de los equipos, el desgaste que sufren por su uso los ordenadores y demás herramientas que se emplean. Se ha estimado el precio del ordenador, múltimetro y demás piezas en 1.300 euros, y supuesto que se cambiarán cada 15 años. Esto resulta en una amortización anual de 86€ y un total durante el desarrollo de 43€. También le se ha tenido en cuenta el desgaste de la media hora que lleva montar e iniciar el sistema.

Teniendo en cuenta esto y que el precio de venta seria de entorno a los 100€ se tendrán que vender un total de 92 unidades para recuperar la inversión. El beneficio por cada venta a podría alcanzar los 40€.

5. Antecedentes

En esta sección se presenta el estado actual de los sistemas utilizados a lo largo del proyecto. Además del estado, también se explican algunas de sus variantes y cuáles son las funciones que realizan. En esta sección aparecerán variantes de alto presupuesto en las que está inspirado el desarrollo, pero no se dispone de los recursos necesarios para emularlos.

5.1. Tipos de drones y sus usos

Como ya se ha comentado previamente, se conoce como dron a toda aeronave no tripulada. Hoy en día son herramientas muy útiles con múltiples usos y en términos generales su forma varía en función de los mismos. Los dos más comunes son los siguientes:

Los drones conocidos como cuadricópteros. Se trata de aeronaves con cuatro hélices estructuradas en forma de cruz para maximizar la estabilidad. También existen variantes con más motores para más seguridad y más potencia. Gracias a la estabilidad de estos vehículos, el autopilotaje no es ninguna novedad, ya se disponen de sistemas como ardupilot que permite programar un vuelo. Estos sistemas son muy buenos para la toma de fotografías en zonas con difícil acceso para es el ser humano.

Existen también drones que tienen forma de avión. Son menos comunes y sus sistemas de pilotaje automático son más caros, pero esta forma les da una velocidad máxima y, especialmente, una autonomía muy superior a los cuadricópteros. Debido a sus precio, su uso es más frecuente en sectores como el militar, donde pueden emplearse para sobrevolar zonas e investigar el terreno. El pilotaje automático de este tipo de drones, además de ser más caro, es más complejo debido a que su maniobrabilidad es más reducida.

5.2. Pilotaje automático

Se define como sistema de pilotaje automático aquél que controla un vehículo de forma total o parcialmente autónoma. Estos sistemas comenzaron incorporándose a vehículos de gran tamaño como los aviones, y poco a poco se han ido incorporando en productos accesibles al consumidor medio, como es el caso de robot aspirador -conocido por una de sus marcas, Roomba-, los cortacesped automáticos,...

El pilotaje parcialmente autónomo implica que el usuario del aparato tiene que darle algún tipo de instrucciones, ya sean coordenadas, un área a recorrer o alguna otra forma de medir las distancias o los puntos que deberá atravesar. Una vez se indica el recorrido, el aparato se mueve de forma autónoma.

En el caso del pilotaje totalmente automático el usuario no tendrá que hacer nada, el aparato recorrerá el terreno de forma autónoma cuando lo considere oportuno o se le indique. Un ejemplo de pilotaje autónomo es el robot aspirador. Este robot recorre la casa generando un mapa y al finalizar regresa al punto de salida.

5.3. Estado actual de las tecnologías

El pilotaje automático ha evolucionado a pasos agigantado en la ultima década.

La empresa estadounidense Tesla es un claro ejemplo de los avances tecnológicos que proporcionan estos sistemas, sin embargo, es raro encontrar aplicaciones económicas. Es por esto mismo que la mayoría de información existente queda descartada a la hora de desarrollar el controlador programable que se ha diseñado para este proyecto.

Los sistemas actuales funcionan por reconocimiento de imágenes, además de sensores, mientras que el sistema desarrollado en este proyecto solo utilizará sensores. Esto hace que la potencia de computo necesaria sea menor, además de abaratar los costes y aligerar el hardware a cambio de seguridad y potencia.

6. Captura de Requisitos

El objetivo de este proyecto es diseñar un sistema que permita programar vuelos. Idealmente, deberá diseñarse de un modo que permita al usuario incorporarlo de manera sencilla, es decir, que para incorporar esta nueva función el usuario no requiera conocimientos informáticos distintos a la programación. Las instrucciones del vuelo se tienen que dar de forma sencilla, han de ser completamente personalizables y deben permitir que los distintos canales del dron se ajusten de forma independiente.

No se requerirá crear una aplicación para programar los vuelos, pero ha de haber un sistema que verifique que los ficheros donde están las instrucciones tienen el formato correcto para garantizar el buen funcionamiento del sistema.

6.1. Casos de Uso

Como se explica en el *desarrollo*, se pondrán dos sistemas (raspberry y arduino) funcionando coordinadamente, por lo que se pueden identificar dos actores distinto. El primero de ellos es el usuario del dron, que es quien ejecuta las instrucciones por medio de la consola de comandos. El segundo actor es el propio sistema de piloto automático (autopilot), que podrá enviar los datos leídos del fichero a la raspberry al arduino.

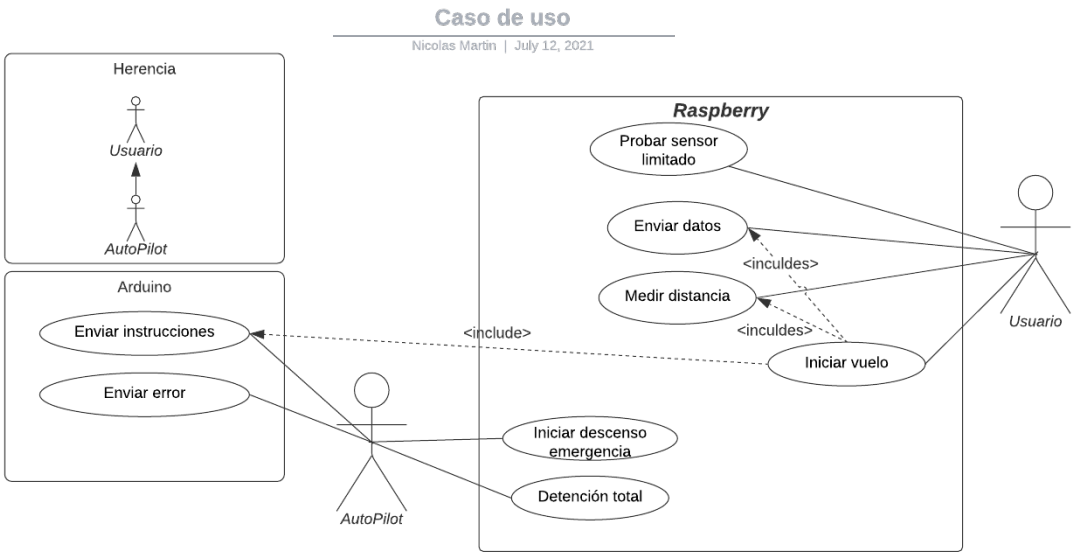

Figura 3: Casos de uso

Se diseñará un sistema muy sencillo que requiera muy poca interacción del usuario, siendo la mayoría de sus acciones probar si el sistema funciona antes de cargar el vuelo. El usuario también puede crear un comportamiento nuevo en un vuelo, pero al ser algo externo que implica añadir código, no se ha incluido en los casos de uso.

El autopilot además de interactuar con el arduino, podrá también realizar todas las acciones que realiza el usuario.

6.2. Casos de Uso Extendidos

Ya que el sistema del piloto automático tendrá poca interacción con el usuario de forma directa -debido a que en la mayoría de ocasiones las interacciones se realizarán de forma remota desde otro dispositivo- se optará por un manejo a través de la consola de comandos.

La mayoría de las acciones que se pueden tomar por parte del usuario tienen dos formas de ejecutarse: directamente en un comando o por medio de iniciar el menú. La primera opción está principalmente diseñada para la realización de las pruebas, pero se mantendrá como funcionalidad.

6.2.1. Medir distancia.

- **Nombre**: Medir distancia.
- **Descripción**: El usuario inicia este modo y selecciona cuántas iteraciones desea hacer, después el sensor se pone en marcha y se muestra en pantalla la distancia que se está midiendo.
- **Actores**: Usuario y Autopilot
- **Precondiciones**: El número de iteraciones es un valor natural mayor o igual que cero.
- **Postcondiciones**: Se muestran los datos por pantalla.
- **Flujo de eventos**:
  - **1A**: El usuario inicia la aplicación en modo menú y este modo abre un menú que le permite elegir la acción a realizar.
  - **2A**: Selecciona el modo "medir distancia".
  - **3A**: Indica el número de iteraciones.
  - **4A**: Los datos se muestran por pantalla.

  - **1B**: El usuario inicia la aplicación en modo distanceLoop, y añade el parámetro -l siendo el número de iteraciones.
  - **2B**: Los datos se muestran por pantalla.

- **Consola:**

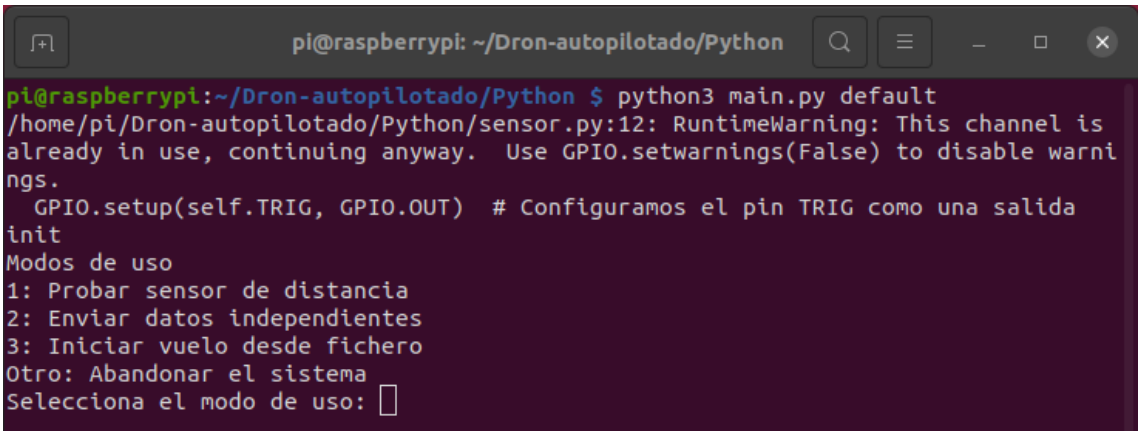

Figura 4: Paso 1A, Menú

Introduciendo el comando "python main.py default" o del ejecutable se abrirá este menú en la consola.

Figura 5: Paso 2A, Elegir el modo

El usuario navega por el menú como se puede observar en la imagen anterior. Introduciendo un 1 se selecciona el modo de medición de distancia.

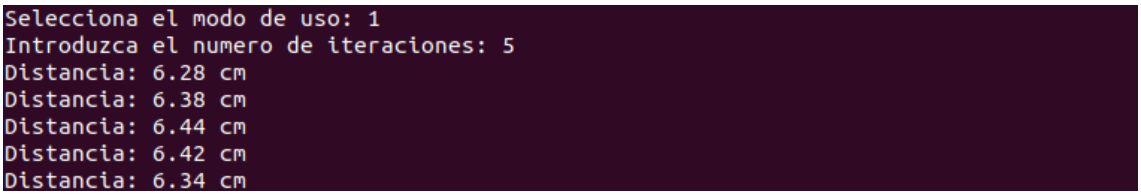

Figura 6: Pasos 3A y 4A, Elegir el número de iteraciones

Tras elegir el modo de prueba del sensor de distancia, se preguntará por el numero de iteraciones. Si se indica cero se mostrará la distancia hasta que se pare la ejecución. En caso de indicarse cualquier otro valor, se mostrará la distancia ese numero de veces.

Todos los pasos descritos previamente también se pueden ejecutar de una sola vez con el comando siguiente, donde x será el número de iteraciones que se quieren hacer.

```
python main.py distanceLoop -l x
```

```
pi@raspberrypi:~/Dron-autopilotado/Python $ python3 main.py distanceLoop -l 5
init
Distancia: 100.22 cm
Distancia: 7.34 cm
Distancia: 7.32 cm
Distancia: 8.31 cm
Distancia: 7.31 cm
```

Figura 7: Pasos 1B, Probar sensor distancia

Como podemos ver, el resultado es exactamente el mismo que si se ejecuta desde el menú, pero mas cómodo para realizar pruebas.

6.2.2. Probar sensor de distancia limitado.

- **Nombre**: Probar sensor de distancia limitado.
- **Descripción**: Se introduce el comando y la distancia de seguridad. El resultado será la distancia y si el dron continua la marcha o se detiene.
- **Actores**: Usuario y Autopilot
- **Precondiciones**: La distancia es un valor número real mayor o igual que cero.

- **Postcondiciones**: Se muestran los datos por pantalla.
- **Flujo de eventos**:
  - **1A**: El usuario inicia la aplicación en modo testControl, y añade el parámetro -sd,la distancia de seguridad mínima antes de detenerse.
  - **2A**: Los datos se muestran por pantalla.
- **Consola:**

```
pi@raspberrypi:~/Dron-autopilotado/Python $ python3 main.py testControl -sd 10
init
Follow 16.74 cm
Follow 16.86 cm
Follow 16.81 cm
Follow 16.75 cm
Follow 16.75 cm
Follow 16.76 cm
Follow 16.76 cm
Follow 16.80 cm
Stop 4.71 cm
Stop 3.82 cm
Stop 3.39 cm
Stop 3.28 cm
Stop 3.84 cm
Stop 3.81 cm
Stop 3.86 cm
Stop 3.49 cm
Follow 17.25 cm
Follow 16.93 cm
```

Figura 8: Paso 1A, Resultado

Introduciendo el comando que se muestra debajo, siendo x la distancia de seguridad mínima. Como se puede ver en el ejemplo, cuando la distancia es inferior a la mínima, se muestra que habría que detener el dron.

```
python main.py testControl -sd x
```

6.2.3. Enviar datos.

- **Nombre**: Enviar datos.
- **Descripción**: Envía datos de forma independiente en lugar de tener que crear un vuelo completo. Es una manera muy útil para comprobar cómo reacciona nuestro dron a los distintos valores de las potencias. Además se puede utilizar para controlar de manera remota el dron siempre que esté en la misma red que el dispositivo desde el que se mandan las instrucciones.
- **Actores**: Usuario y Autopilot
- **Precondiciones**: Una lista de valores válidos, es decir, que estén entre cero y cien o sean alguno de los predeterminados (252,253,254 o 255). La lista debe contener 8 valores o el valor 255 para indicar que ese será el valor final.
- **Postcondiciones**: Se muestran los datos por pantalla. En caso de error en la transferencia también se mostrará en pantalla y finalmente se enviarán las señales al arduino, para su procesado final.
- **Flujo de eventos**:
  - **1A**: El usuario inicia la aplicación en modo menú.
  - **2A**: Selecciona que desea enviar datos.
  - **3A**: Indica qué datos va a enviar, siguiendo las instrucciones que aparecen en pantalla.
  - **4A**: Los datos se muestran por pantalla y se envían al aruduino.

    [Si no hay error]
    - **5AA**: Arduino convierte estos datos en valores de PWM.

- **6AA**: Se generan los pulsos PWM para el controlador de vuelo.

[Si hay error]

- **5AB**: Envía mensaje de error al arduino y reintenta mandar los datos hasta que no haya errores.
- **6AB**: Arduino convierte estos datos en valores de PWM.
- **7AB**: Se generan los pulsos PWM para el controlador de vuelo.

- **1B**: El usuario inicia la aplicación en modo sendData y con el parámetro -d.
- **2B**: Se envían los datos a arudino.

  [Si no hay error]

  - **3BA**: Arduino convierte estos datos en valores de PWM.
  - **4BA**: Se generan los pulsos PWM para el controlador de vuelo.

  [Si hay error]

  - **3BB**: Envía mensaje de error al arduino y reintenta mandar los datos hasta que no haya errores.
  - **4BB**: Arduino convierte estos datos en valores de PWM.
  - **5BB**: Se generan los pulsos PWM para el controlador de vuelo.

- **Consola:**

El paso 1A es el mismo que para medir la distancia. Para un ejemplo ver: *imagen del menú*

Una vez seleccionado el modo aparecerán las instrucciones del formato que han de tener los datos a la hora de mandarlos.

```
Selecciona el modo de uso: 2
Los datos seran valores de 0 a 100
 Estos valores indicaran los datos que se mandan
 valores: roll,pitch,yaw,power,aux1,aux2,aux3,aux4
 ejemplo: 100,50,50,100,0,0,100,0
 8 valores o 255 para enviar menos
 ejemplo: 100,255 manda 100 al roll y mantiene los demas valores igual
Itroduzca los datos en el formato especificado:
```

Figura 9: Paso 2A, Introducir los datos a enviar

Una vez se indica que datos se desea mandar, comienza el proceso.

```
Itroduzca los datos en el formato especificado: 50,100,255
Decoded data = [50, 100, 255]
Sending = 50 Last Sended = 10 Reciving = 0
Sending = 100 Last Sended = 50 Reciving = 50
Sending = 255 Last Sended = 100 Reciving = 100
correcto
GPIO.cleanup() y spi.close() ejecutados
```

Figura 10: Paso 4A, Se muestran los datos por pantalla

Se muestran por pantalla los datos y la confirmación de que se ha enviado todo correctamente. Como se puede observar en la siguiente imagen, estos cambios se ven reflejados en las potencias del dron. Los pasos intermedios del arduino no se ven reflejados en la consola.

Todos los pasos anteriores también se pueden ejecutar de un solo vez por medio de la consola de comandos. El comando, en este caso,es el que se muestra a continuación, donde data son los datos que se desean mandar en el mismo formato que se pide en el menú.

```
python main.py sendData -d data
```

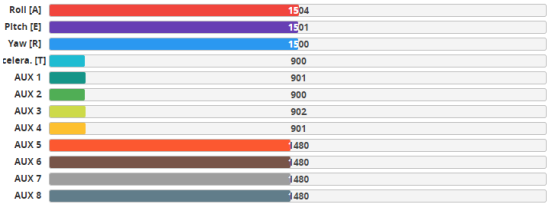

Figura 11: Valores antes del cambio

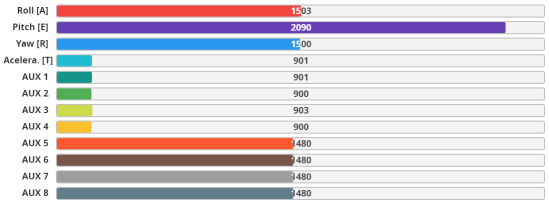

Figura 12: Valores después del cambio

```
pi@raspberrypi:~/Dron-autopilotado/Python $ python3 main.py sendData -d 100,0,0,0,0,
0,0,0
init
Decoded data = [100, 0, 0, 0, 0, 0, 0, 0]
Sending = 100 Last Sended = 10 Reciving = 80
Sending = 0 Last Sended = 100 Reciving = 100
Sending = 0 Last Sended = 0 Reciving = 0
Sending = 0 Last Sended = 0 Reciving = 0
Sending = 0 Last Sended = 0 Reciving = 0
Sending = 0 Last Sended = 0 Reciving = 0
Sending = 0 Last Sended = 0 Reciving = 0
Sending = 0 Last Sended = 0 Reciving = 0
correcto
GPIO.cleanup() y spi.close() ejecutados
```

Figura 13: Paso 3A, Introducir los datos a enviar

Como se puede observar, los cambios también se ven reflejados aquí.

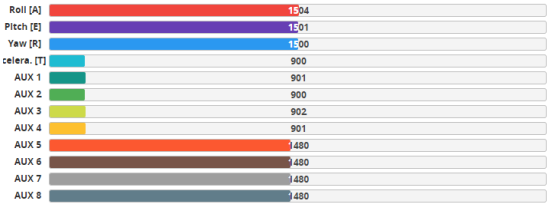

Figura 14: Valores antes del cambio

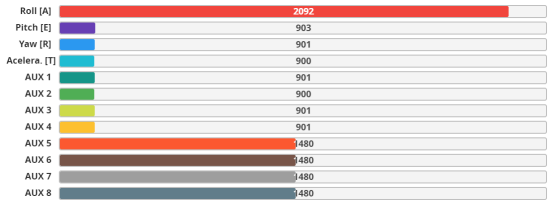

Figura 15: Valores después del cambio

En caso de que haya un error, éste se mostrará en la consola con el siguiente formato:

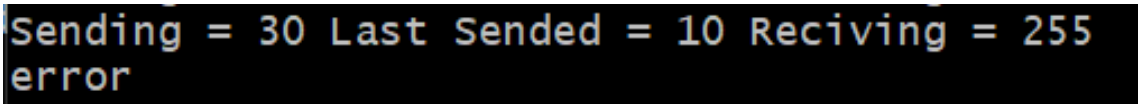

Figura 16: Mensaje de error en el envió de datos

Este mensaje quiere decir que algún elemento de la transmisión no está funcionando correctamente, y que habiénose enviado el valor 10, arduino ha recibido el valor 255.

6.2.4. Iniciar vuelo.

- **Nombre**:Iniciar vuelo.
- **Descripción**: Lee el fichero indicado por el usuario y sigue las instrucciones. Ejecuta los métodos de medir distancia y de enviar datos como parte de su ejecución.
- **Actores**: Usuario y Autopilot
- **Precondiciones**: Un fichero de vuelo que exista y sea correcto.
- **Postcondiciones**: Se ejecutan las insrucciones.
- **Flujo de eventos**:
  - **1A**: El usuario inicia el modo menú.
  - **2A**: Selecciona que quiere hacer el vuelo automático.
  - **3A**: Elige el archivo que contiene las instrucciones del vuelo.
    [si es correcto el fichero]
    - **3AA**: Comienza el vuelo.

    [si es incorrecto]
    - **3AB**: Notifica los errores.

- **1B**: Introduce el comando con la dirección del fichero donde se ha guardado el vuelo que se quiere realizar.

  [si es correcto el fichero]

  - **2BA**: Comienza el vuelo.

  [si es incorrecto]

  - **2BB**: Notifica del error.

- **Consola:**

El paso 1A es el mismo que para medir la distancia. Para un ejemplo ver: *imagen del menú*

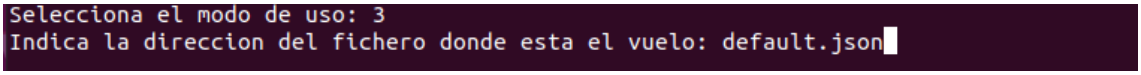

Figura 17: Paso 2A, Selección del fichero

Una vez seleccionamos el fichero comienza el vuelo.

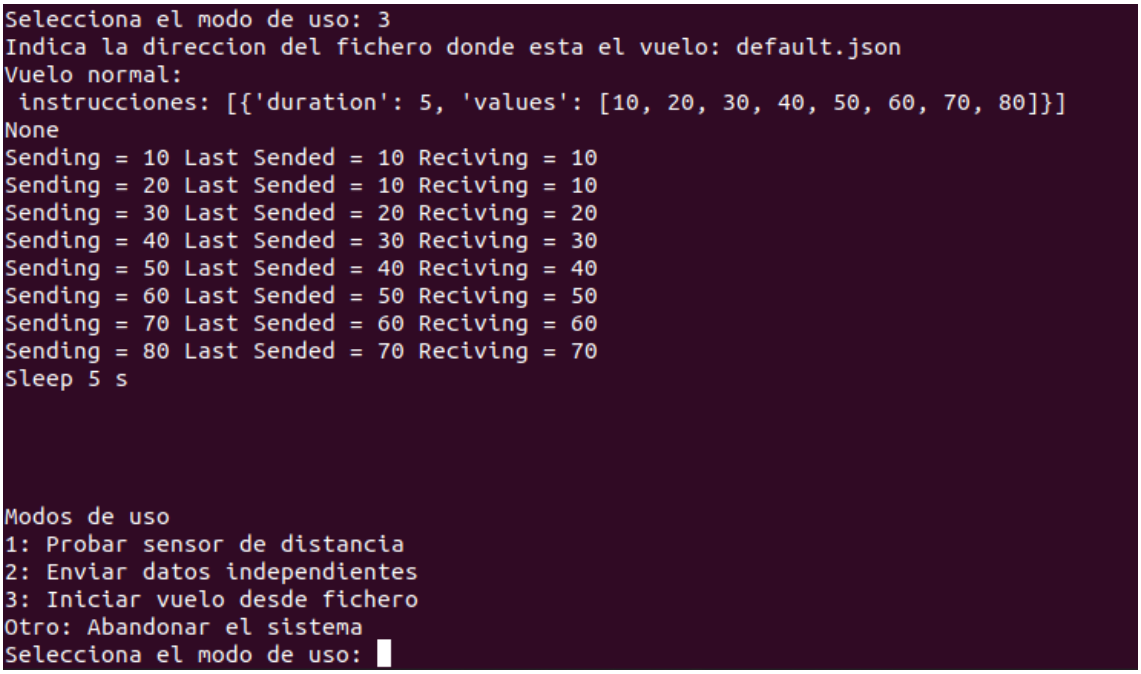

Figura 18: Paso 3AA, Comenzar el vuelo

Si se opta por ejecutar el vuelo por medio de la consola de comandos, el resultado sería el de la siguiente imagen, y el comando sería:

python main.py defaultTest -f file

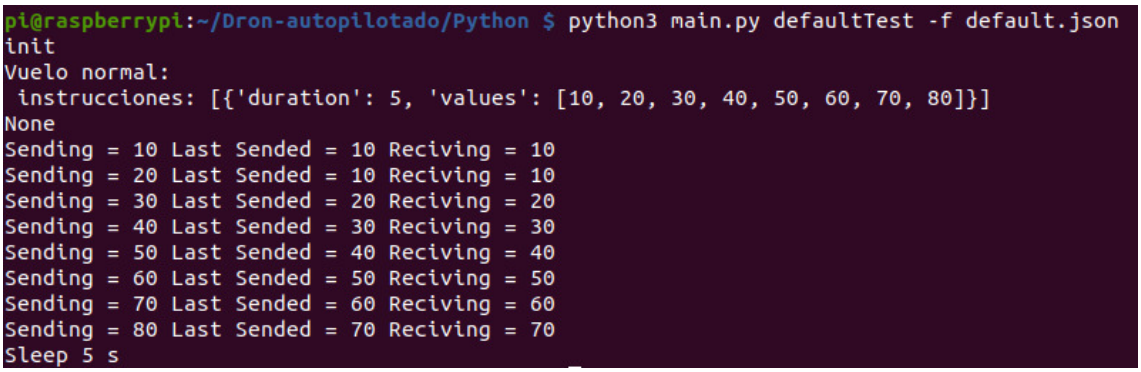

Figura 19: Paso 1B, Comenzar el vuelo

Como podemos ver, el resultado será el mismo que si se ejecuta por medio del menú solo que más cómodo para las pruebas.

```
(venv) C:\Users\Nicolas\Desktop\Guardar\Trabajos\TFG\Dron-autopilotado\Python>python fileProcessor.py VuelosEjemplos\distanceLoop.json
Error, falta el atributo times, ej: "times":5
Error, falta el valor distance en la instruccion: {'values': [0, 100, 0, 60, 0, 0, 0, 0]}
Error, falta el valor duration en la instruccion: {'distance': 15}
```

Figura 20: Mensajes de error en el fichero

En caso de error en el fichero se mostrará el mensaje anterior para ambas situaciones, tanto si se ejecuta por comando como si se hace a través del menú. Este mensaje muestra el error e informa de los elementos necesarios para subsanarlo y que funcione ese tipo de vuelo.

6.3. Diagrama de Clases

En este apartado se explica el diagrama de clases y una breve descripción de cada clase. Hay que destacar que, pese a haber sistemas distintos, el único que cuenta con diagrama de clases es la raspberry pi. El motivo por el que se ha optado no usar orientación a objetos en arduino, pese a que lo permite, es que la falta de memoria en programas un poco complejos suele dar problemas de overflow.

En primer lugar, se analizarán las clases de los vuelos. En éstas es donde se encuentra toda la lógica para gestionar las potencias en función de las instrucciones.

Flight: Esta es la clase padre de todos los vuelos, es el vuelo más simple y una primera versión. Implementa un sistema de vuelo que funciona por tiempos, se le especifican las potencias y la duración, y ejecuta el vuelo tal y como está indicado.

FlightLoop: Hereda todo de la clase Flight, y re-implementa el método fly(). Es un vuelo ligeramente más complejo, que también funciona por tiempo, pero añade la posibilidad de hacer un bucle y repetir las instrucciones tantas veces como el usuario lo desee.

FlightDistance: Hereda todo de la clase Flight, y re-implementa el método fly(). Añade además un método para calcular el tiempo que tarde en recorrer la distancia especifica. Este es el primer tipo de vuelo que se programa, poniendo la distancia que se desea recorrer con cada una de las potencias.

FlightDistanceLoop: En esencia esta clase es lo mismo que FlightLoop, pero en lugar de hacer el vuelo por tiempo en bucle, lo hace por distancia.

FlightSensor: De los vuelos que hay implementados, este es el más complejo. Se programa por distancia al igual que los 2 vuelos anteriores, pero en este caso el dron se detiene en el supuesto de que vaya a colisionar con otro objeto.

Une vez implementados los vuelos, se especifica cómo se crean. Para esto, se necesitan dos cosas: el FilghtFactory y el fichero del vuelo, que se explica en detalle más adelante ***(Enlace a la sección donde se explica).***

FlightFactory: Aquí se lee cual es el tipo de vuelo, desde el fichero donde está programado, y se crea un nuevo vuelo de ese tipo. De esta forma, implementar nuevos vuelos es cómodo y sencillo.

Luego están las clases que se encargan de los datos del sistema: SPITransmiter, DistanceSensor y FileProcesor.

SPITransmiter: Esta clase se ocupa de transmitir la información de manera correcta entre la raspberry pi y el arduino micro. Además, es capaz de mandar errores y detener todo el sistema en caso de fallo grave.

DistanceSensor: Es la encargada de manejar el sensor de distancia. En caso de haber más de uno, habría que instanciarla una vez por cada sensor.

FileProcesor: Es la clase encargada de leer y procesar los ficheros Json que contienen los vuelos. Verifica si son correctos y guarda la información necesaria para que el resto del sistema pueda usar los datos sin necesidad de volver a acceder al archivo.

Finalmente tenemos el main y ConfigaVariables, donde se implementa la lógica básica de cómo funciona el sistema y donde está el ejecutable del código.

ConfigVariables: Pese a no ser una clase si no un archivo de python, se ha optado por incluirlo en la explicación y en el diagrama de clases. Durante la ejecución hay muchas variables que dependerán de la configuración del dron. Este archivo permite ajustarlo a la configuración del dron del usuario en especifico. EL código de arduino también cuenta con una sección de variables de configuración para el mismo uso.

Main: Aquí se encuentra toda la lógica del programa. Cuando se hace uso del ejecutable, es la clase que se lanza.

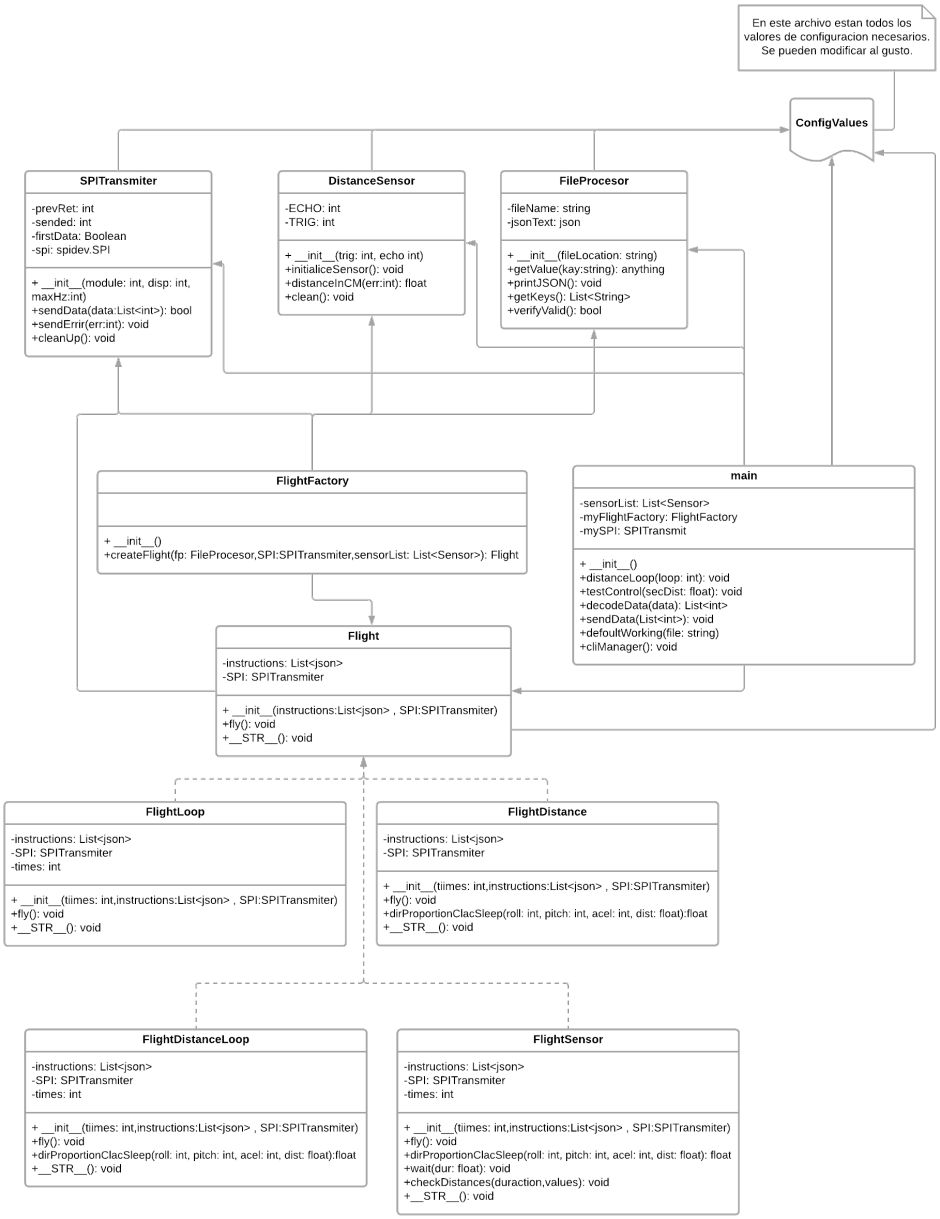

Figura 21: Diagrama de clases

Cabe destacar el uso del patrón factory para crear los vuelos. También

se podrían haber utilizado singletons para el FileReader, el SPITransmiter y el FilrProcresor, creando una clase superior para la lista de sensores, pero se ha optado por no hacerlo. Aunque finalmente parece que hubiera sido más cómodo hacerlo. Queda pendiente como posibilidad de mejora en caso de ampliar el proyecto.

6.4. Diagramas de Secuencia

A continuación se muestran los diagramas de los métodos principales. Ya que el código de la mayoría de los métodos es muy sencillo se ha optado por no añadir los diagramas con el fin de no hacer la lectura del documento más tediosa.

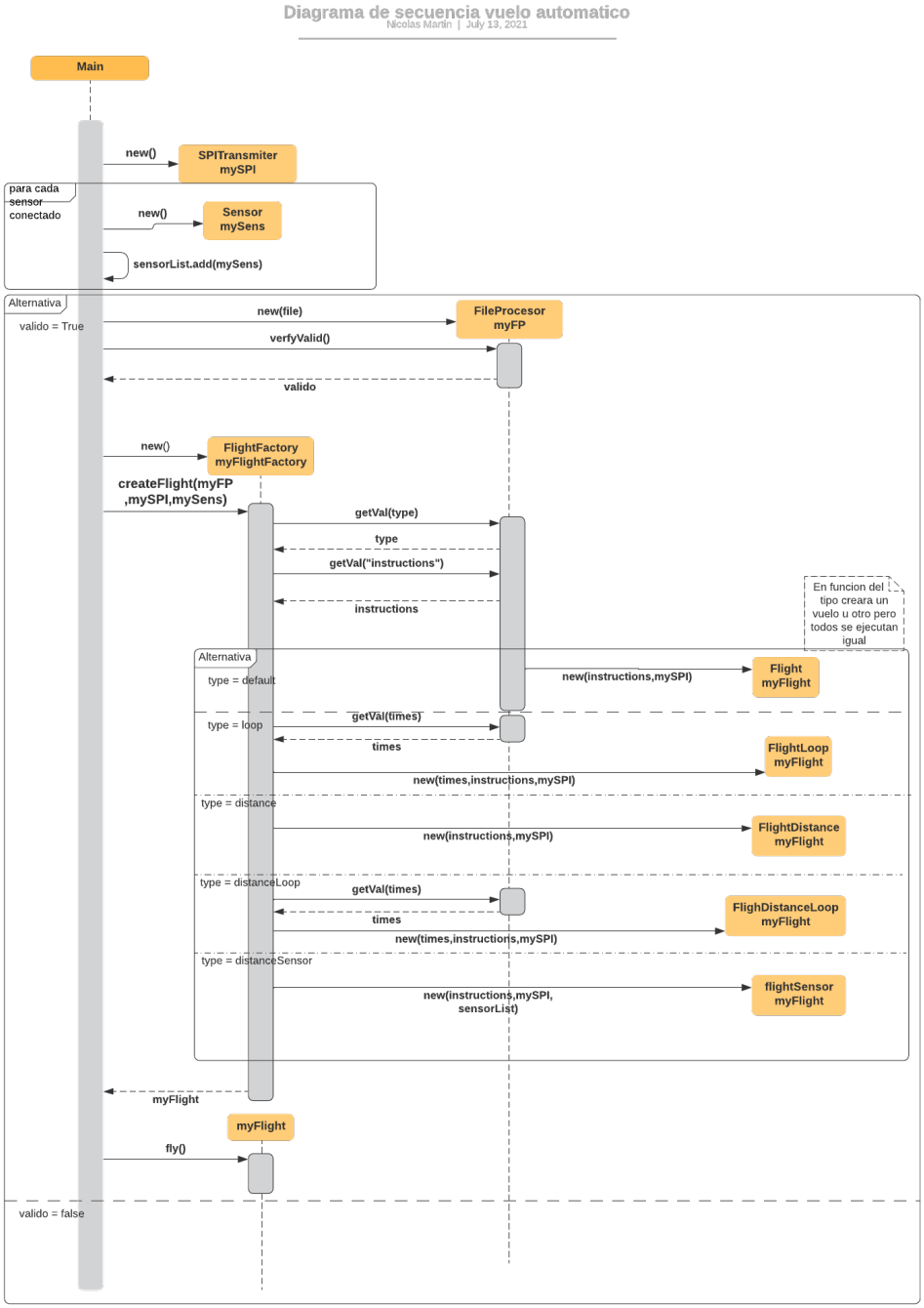

Figura 22: Diagrama de secuencia Main

Este es el diagrama de la secuencia principal. En este diagrama se incluye el método fly que dependerá de la clase del vuelo, pero como ejemplos se explican a continuación el método fly de las clases Flight y FlightSensor ya que son el más sencillo y el más complejo respectivamente.

El primer diagrama es de la clase *Flight*, este método es muy sencillo. Envía la serie de instrucciones y espera el tiempo indicado. Una vez transcurrido este tiempo, envía el nuevo conjunto de instrucciones. En el caso del segundo diagrama de la clase *FlightSensor*, es un poco más complejo. El método necesita medir la distancia que va a recorrer para calcular el tiempo que va a esperar. Además utiliza multiporcessing para poder medir la distancia mientras espera para mandar el siguiente conjunto de instrucciones. Esto último no resultaba necesario, pero ya que raspberry pi y python tienen la capacidad de implementarlo, se toma como una oportunidad de aprendizaje.

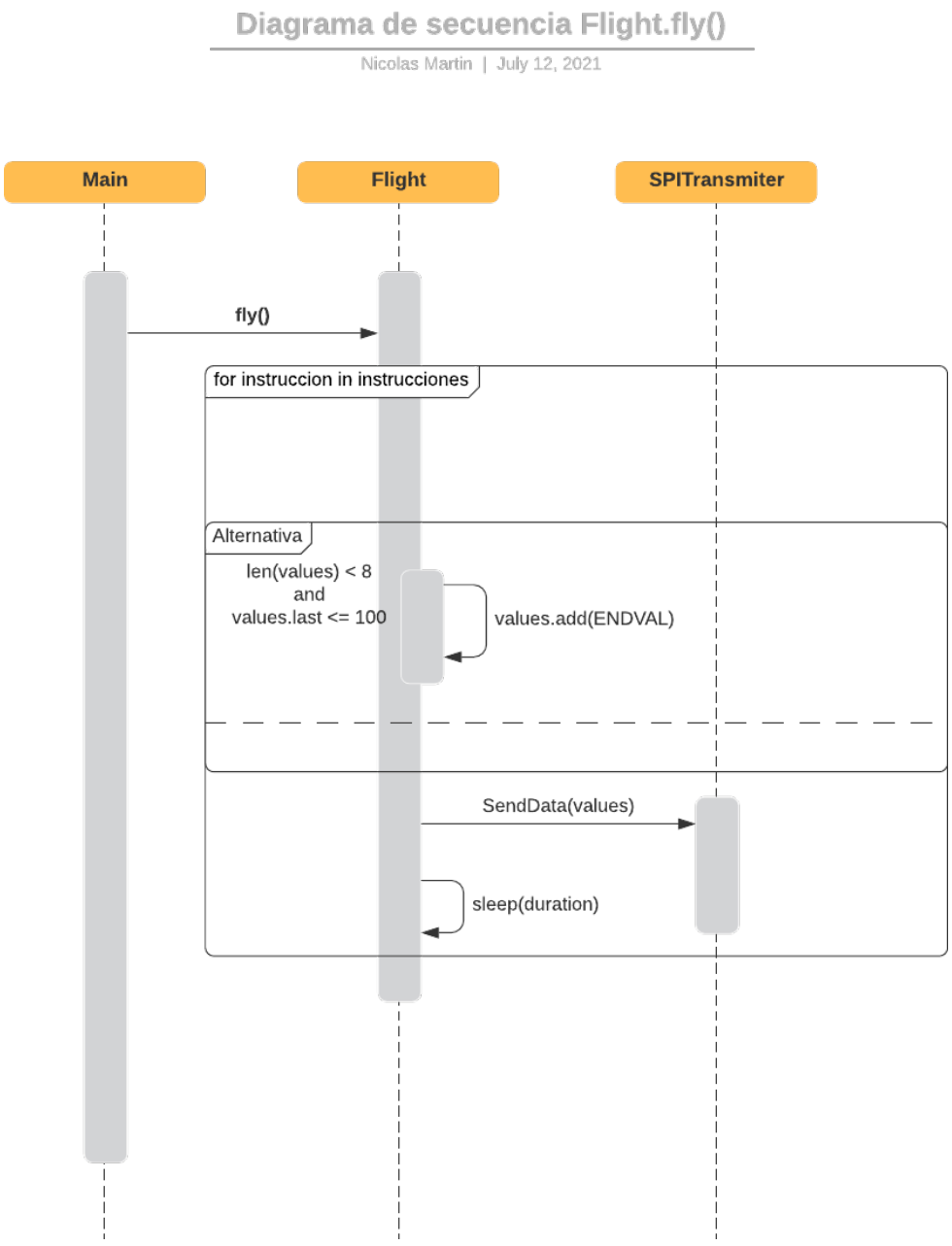

Figura 23: Diagrama de secuencia Flight.fly()

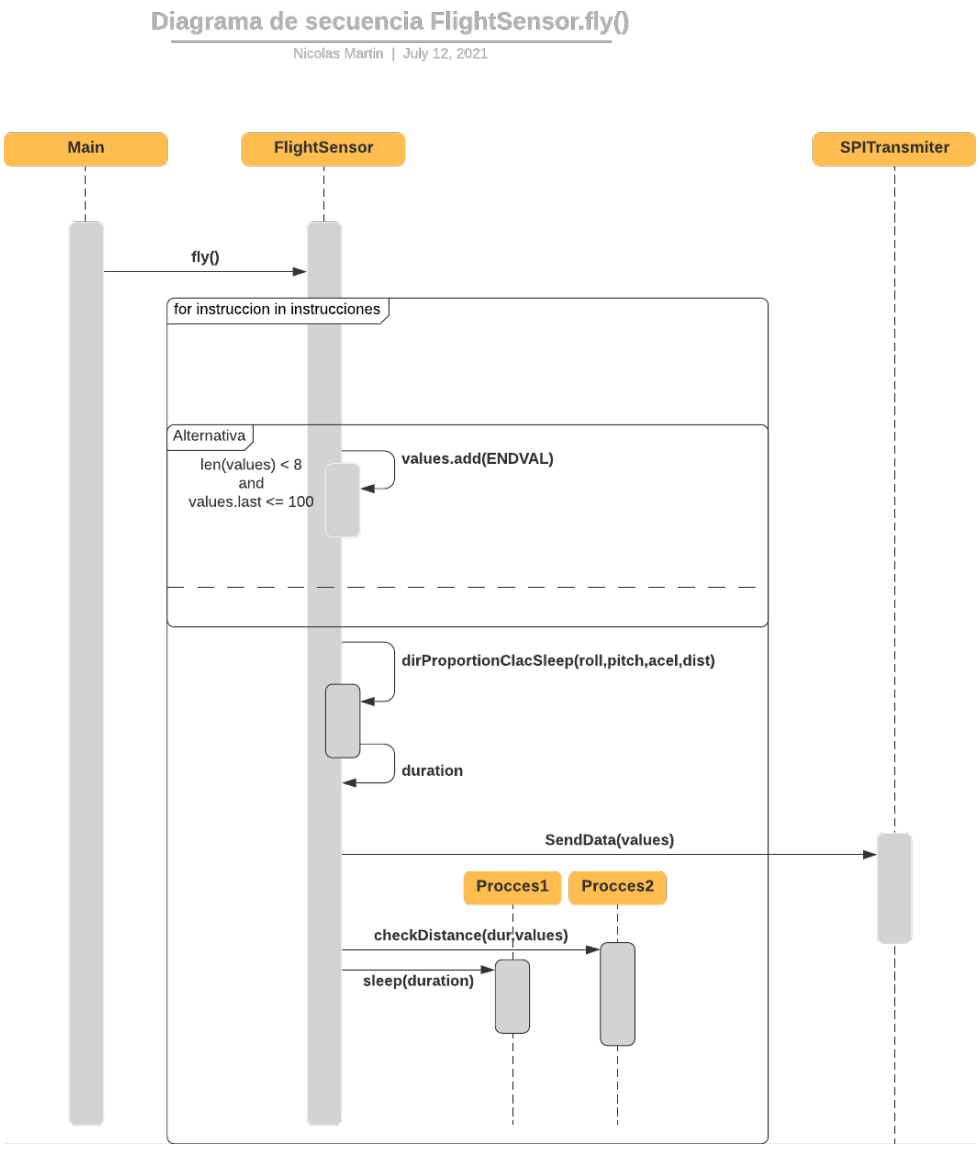

Figura 24: Diagrama de secuencia FlightSensor.fly()

Por último tenemos los métodos *envarDatos()* y *distanceCM()* que se utilizan a lo largo del código. Todos los vuelos llaman al método eviarDatos(), aunque también se puede hacer directamente desde el main como se muestra en el diagrama.

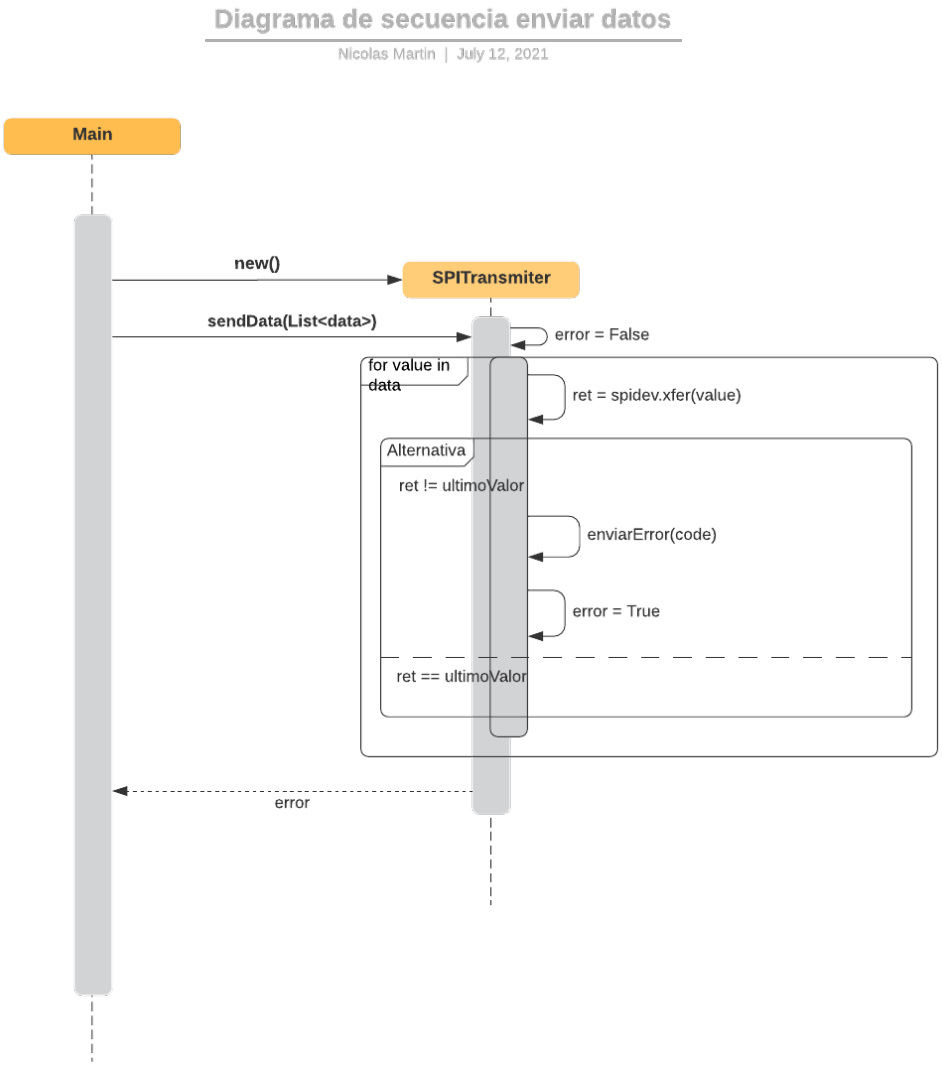

Figura 25: Diagrama de secuencia de enviarDatos()

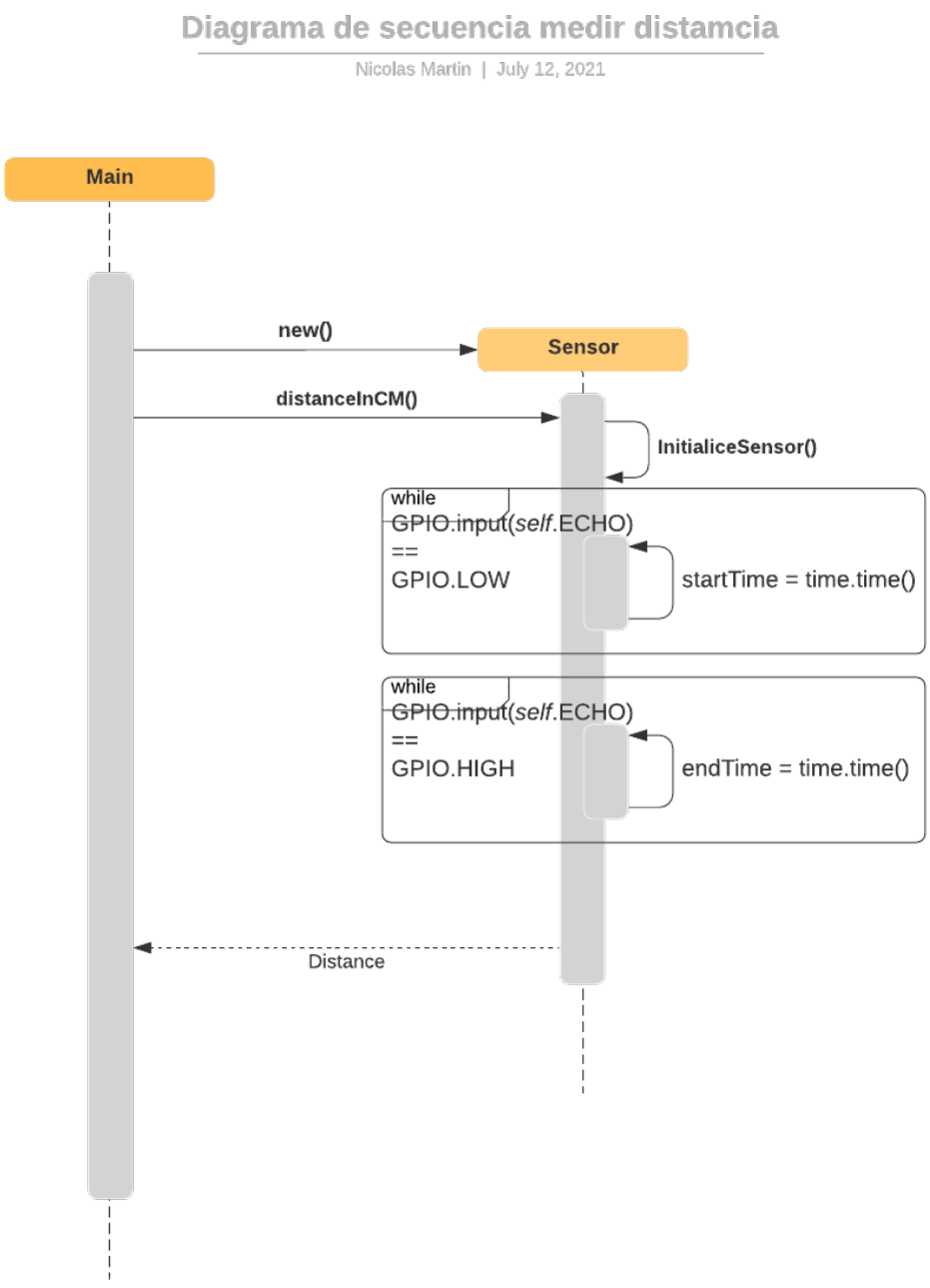

Figura 26: Diagrama de secuencia medir distancia

7. Desarrollo

En este sección se expone el proceso de desarrollo completo, con las pruebas y la investigación realizada para llegar al resultado final del proyecto.

7.1. Selección del hardware y la arquitectura

Es crucial que todos los componentes elegidos para este proyecto funcionen de manera correcta, por eso se han analizado las opciones más viables económicamente para ver cuales podrían funcionar. Se han valorado 4 puntos principales: el precio, el peso, la memoria y la complejidad a la hora de programar.

Lo primero a tener en cuenta es con qué opciones contamos en cuanto al enfoque del proyecto. Para lo cual es necesario tener un concepto genérico del funcionamiento de un dron, en específico, de los controles y el vuelo.

Los drones disponen de 2 piezas principales para su manejo:

- El **receptor de radio** se encarga de recibir las instrucciones del mando. En esencia informa al dron de qué tiene que hacer. Existen distintos tipos de señales a la hora de trasmitir la información entre el receptor de radio y el controlador de vuelo. Los principales son PPM y PWM.
- El **controlador de vuelo** procesa las señales que recibe del receptor de radio y se las envía al ESC. Además, en general, cuenta con un barómetro y un acelerómetro para nivelar y estabilizar el dron.

Visto esto, existen dos opciones para poder desarrollar el proyecto. Diseñar un controlador de vuelo programable y simular un receptor de radio interno por medio de instrucciones guardadas previamente. O comprar cualquier controlador de vuelo del mercado e implementar un dispositivo capaz de generar señales simulando a un receptor de radio.

Programar el controlador de vuelo	**Comprar el controlador de vuelo**
Menos coste económico	Menos potencia de cálculo necesaria
Más libertad a la hora de transimitir la información	Código más ligero
Más libertad de programación	Más tiempo invertido en el autopiloto
Código más pesado	Mejor calidad de Controlador de vuelo
Requerimientos de tarjeta programable más exigentes	Hardware especializado para su uso
Mástiempo programando fuera de los necesario	Más fácil de montar en otros drones
Menos recursos en la web	Más caro
Reduce la capacidad de uso en otros drones	Máslimitado por las opciones del mercado
Peor calidad de Controlador de vuelo	Opciones de transmisión de datos limitadas
Hardware extra necesario (acelerómetro, barómetro)	Opciones de vuelo limitadas por el Controlador

Cuadro 3: Diferencias entre programa y comprar el controlador

En un principio podría parecer que programarlo todo reduciría el peso y el tamaño del dron, pero hay que tener en cuenta que, con el aumento de los recursos necesarios para los nuevos cálculos, la tarjeta programable aumentará su peso. Además, los controladores de vuelo vienen con un barómetro y un giroscopio incorporado, para poder realizar todos los cálculos. Para lograr el objetivo de este proyecto tendríamos que añadir esas piezas como hardware extra.

Al no tener una pieza especializada para el control del vuelo, la calidad y estabilidad del dron disminuirá. También es cierto que el piloto automático podría aumentar su calidad al gestionarlo todo directamente.

Comprar el controlador produciría el efeto contrario, simplificaría el código y mejoraría la calidad del controlador de vuelo, además de eliminar el código y el hardware extra. En este caso, lo que se pierde es un poco de libertad a la hora de programar ya que el usuario quedaría sujeto a cómo funciona el controlador.

Con las ventajas y desventajas de cada una de las opciones en mente, finalmente se optó por comprar el controlador de vuelo. De esta forma, el código será más sencillo además de mucho más fácil de incorporar a todo tipo de drones.

Se necesita que el controlador de vuelo tenga giroscopio, barómetro y acelerómetro. Además, sería recomendable que utilizase cómo mínimo PPM a la hora de revivir las señales, ya que es un sistema más estandarizado y mucho más moderno que PWM. Teniendo en cuenta que el precio medio de un controlador de vuelo con piloto automático como el pixhawk 4 es de 150 euros, y la intención de este proyecto es abaratar el sistema, se establece invertir un máximo 60 euros en el controlador de vuelo.

Entre las opciones de disponibles están el F405-CTR de Matek que, por su tamaño y peso es el más conveniente, y del que ya se dispone de experiencia previa de uso. El Holybro Kakute F7 es también una muy buena opción para realizar este proyecto. Es el controlador más de moda en los drones de carreras, por su estabilidad, pero su precio es superior, lo que impediría ajustarse al presupuesto del proyecto. Entre otras opciones, GEPRC GEP-12A-F4 es un controlador de vuelo más económico, también muy popular en los drones de competición.

Controlador de vuelo	Precio	Calidad	Peso
Holybro Kakute F7	100 euros	Excelente	12g
F405-CTR	50 euros	Muy buena	23g
GEPRC GEP-12A-F	45 euros	Correcta con problemás de estabilidad	25g

Cuadro 4: Opciones de controlador de vuelo

Finalmente se ha optado por el F405-CTR por ser uno de los más populares del mercado, además de disponer de software donde poder ejecutar las simulaciones y ajustarse a todas las necesidades del proyecto.

En cuanto a la arquitectura, se han analizado distintas opciones. Trabajar con un solo PLC con mayores limitaciones o utilizar una combinación de varios PLCs para obtener lo mejor de cada uno de ellos.

Caracteristicas en cuenta	Un PLC sencillo (arduino, esp8266)	Un PLC potente (xilinx, siemens)	Multiples PLCS
Precio	Más barato	Mucho más caro	Ligeramente más caro
Peso	Más ligero	Mucho más pesado	En general en el punto medio entre el potente y el simple
Programabilidad	Mucho más limitado si el PLC es simple	Solo limitaciones del lenguaje	Más libre de lenguaje y con menos limitaciones de potencia
Limitaciones	Muchas limitaciones	Muy pocas limitaciones	Muy pocas limitaciones
Cableado	Casi sin cableado	Casi sin cableado	Necesita cableado extra
Compativilidad	Más limitado por falta de pines	Sin limitaciones	En un punto medio entre el potente y el simple

Cuadro 5: Opciones de PLC

Como podemos ver en la tabla anterior, diseñar un sistema compuesto es bastante equilibrado: solventa el exceso de precio y el exceso de peso y salva las limitaciones de comprar un solo PLC. También proporciona un aspecto muy interesante en cuanto a la oportunidad aprendizaje al tener que resolver la comunicación entre los PLCs.

A continuación se explica la decisión de escoger como dispositivos para el proyecto un arduino micro y un raspberry pi.

Para la selección del arduino se han tenido en cuenta dos factores. Se podría haber escogido prácticamente cualquier dispositivo de arduino pero el micro es, además del más ligero de los que trabaja con 5 voltios, el más barato. Se ha valorado el ESP8266, que dispone de wifi, pero sus salidas son de 3 voltios y está más orientado a la recepción y emisión de señales wifi, además no dispone de pines con la capacidad para generar señales PPM o PWM con suficiente frecuencia. Cabe señalar que arduino micro tampoco tiene capacidad de generar PPM de calidad, pero con PWM se puede funcionar correctamente.

Por ultimo, la raspberry pi se eligió por su potencia y su memoria, además de que permite poder trabajar con múltiples lenguajes. En

esencia es un ordenador capaz de manejar un sistema operativo, así que era ideal para que el código fuera sencillo, eficiente y personalizable. Teniendo todo esto en cuenta podemos ver un pequeño diagrama con la arquitectura que vamos a seguir.

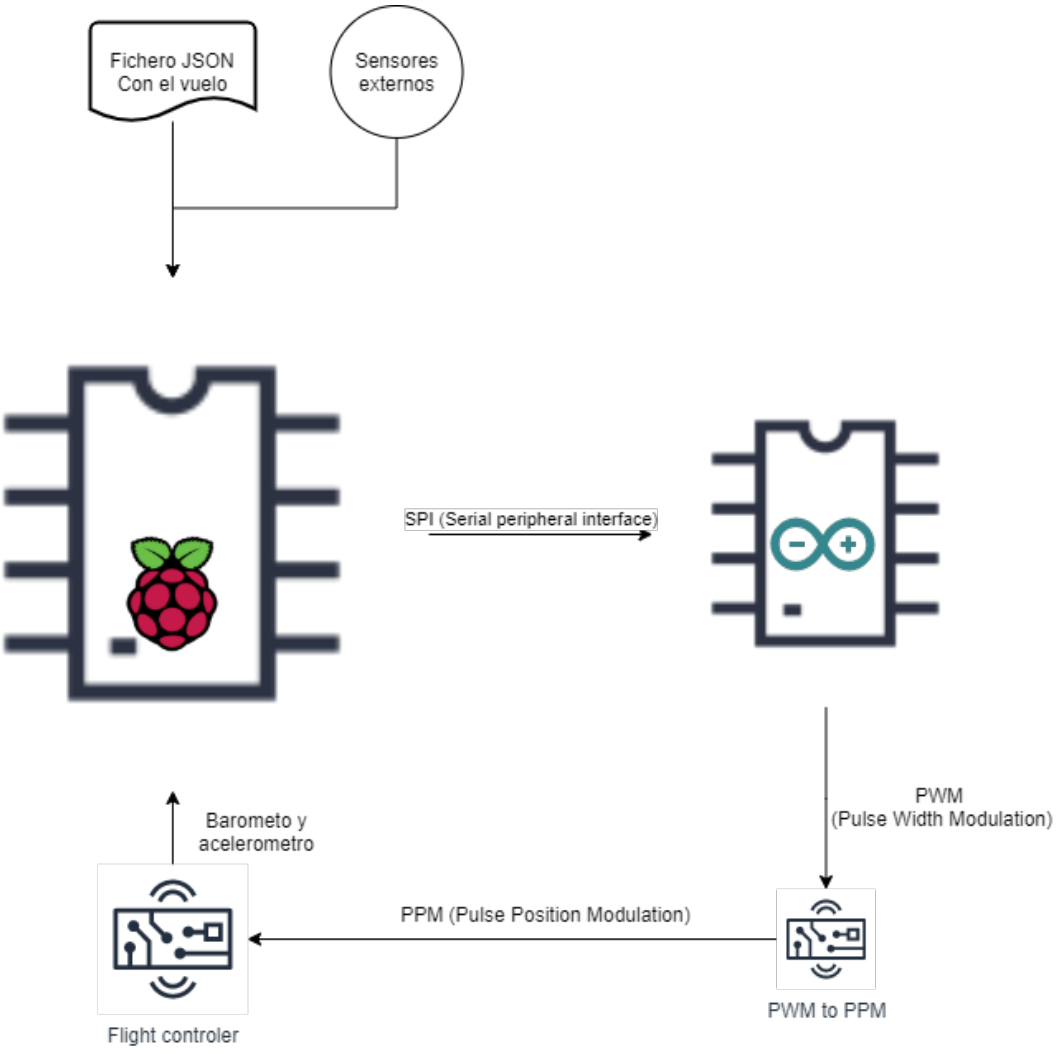

Figura 27: Arquitectura del proyecto

Es un proceso sencillo, la raspberry pi lee el vuelo guardado en un fichero JSON, procesa esta información obteniendo los datos de los sensores y le envía las instrucciones al arudino micro. El arduino micro recibe estas instrucciones por medio de comunicación SPI, genera señales de PWM y las envía al Controlador de vuelo. Debido a las limitaciones de frecuencia, los pines de arduino micro no son capaces de generar señales PPM. Para salvar esta limitación, se añade un conversor de PWM a PPM incrementando 7 euros al precio final del sistema.

7.2. Comunicación entre dispositivos

Posteriormente a la elección de los dispositivos, se lleva a cabo un análisis de las distintos protocolos que se pueden emplear para realizar la transferencia de la información. Esta transferencia de información se ha de realizar de la raspberry pi al arduino micro y del arduino al controlador de vuelo. Con el fin de adquirir un mayor conocimiento sobre la comunicación entre los dispositivos, se ha optado por hacer cada una de las comunicaciones con protocolos distinto. En este apartado se explica al detalle la investigación realizada con las pruebas para cada una de las comunicaciones.

7.2.1. Raspberry-arduino

Con respecto a la comunicación entre la raspberry pi y el arduino micro existen múltiples protocolos de transmisión de información, todo depende de cuan complejo se desee el sistema. En este caso, se prioriza aplicar un protocolo que funcione con cables, ya que, en general, son más rápidos y estables, descartando opciones como el wifi o la radiofrecuencia.

Arduino micro es el elemento más limitante a la hora de elegir el protocolo de comunicación, siendo sus opciones principales los siguiente métodos: comunicación serial, comunicación I2C, comunicación SPI y, por último, UART.

El primer paso ha sido realizar las pruebas para obtener el mejor método de comunicación. Para estas pruebas se han tenido en cuenta tres factores: la velocidad de trasmisión, el tiempo de lectura y el volumen de datos que se puede transmitir de una vez.

Características en cuenta	Velocidad de transferencia	Tiempo de lectura	Tamaño máximo del envío	Extra	Conexiones
Serial	115200 bits/s	1 segundo	1 byte + buffer	Cómodo de programar y cablear	USB
I2C	400 kb/s estables 3.6 Mb/s teóricos	0.0000025 segundos 0.00000028 segundos teóricos	1 byte	Inestable a más de 400kHz	2 pines
SPI	1Mb/s estables 8Mb/s teóricos	0.000001 segundos 0.00000013 segundos teóricos	1byte	El cableado no debe ser muy largo	4 pines
UART	115200 bits/s	0.0000087segundos	1byte	Solo dos cables	2 pines

Cuadro 6: Diferencias entre métodos de comunicación

Como se puede observar en la tabla anterior, Serial y UART son protocolos considerablemente más lentos a la hora de transferir datos. Además en arduino la comunicación serial tarda 1 segundo desde que se inicia la lectura del buffer, donde está la información, hasta que finaliza la lectura. Por esto las opciones se reducen a I2C y SPI.

Tras una investigación exhaustiva, a la hora de elegir entre las últimas dos opciones, se ha observado que I2C permite trabajar con una longitud de cables más larga antes de comenzar a dar errores, pero en general da errores de forma más propensa. Además, SPI consume una menor cantidad de energía, algo que es muy importante en un dron. Es por esto que finalmente se usa SPI.

La siguiente tabla es una pequeña porción de las simulaciones que se han realizado para medir la velocidad. En caso de tener interés:

Serial		Uart		I2C		UART	
Transmision	Tiempo Lectura	Transmisi	Tiempo Lectura	Transmision	Tiempo Lectura	Transmisi	Tiempo Lectura
115195	1	400087	2,49946E-06	999918	1,00008E-06	115206	8,6801E-06
115208	1	399926	2,50046E-06	1000020	9,9998E-07	115203	8,68033E-06
115197	1	399942	2,50036E-06	1000033	9,99967E-07	115203	8,68033E-06
115199	1	399966	2,50021E-06	1000068	9,99932E-07	115207	8,68003E-06
115203	1	399940	2,50038E-06	1000003	9,99997E-07	115210	8,6798E-06
115195	1	399992	2,50005E-06	999995	1,00001E-06	115201	8,68048E-06
115199	1	400066	2,49959E-06	1000021	9,99979E-07	115191	8,68123E-06
115209	1	399999	2,50001E-06	999970	1,00003E-06	115205	8,68018E-06
115210	1	400059	2,49963E-06	999963	1,00004E-06	115205	8,68018E-06
115191	1	400014	2,49991E-06	1000038	9,99962E-07	115190	8,68131E-06
115199	1	399995	2,50003E-06	1000020	9,9998E-07	115193	8,68108E-06
115196	1	400039	2,49976E-06	1000067	9,99933E-07	115194	8,68101E-06
115197	1	399973	2,50017E-06	999951	1,00005E-06	115203	8,68033E-06
115203	1	400069	2,49957E-06	999952	1,00005E-06	115201	8,68048E-06
115204	1	399918	2,50051E-06	1000088	9,99912E-07	115195	8,68093E-06
115204	1	400092	2,49943E-06	1000045	9,99955E-07	115207	8,68003E-06
115195	1	400034	2,49979E-06	999970	1,00003E-06	115198	8,68071E-06
115202	1	399973	2,50017E-06	999925	1,00008E-06	115206	8,6801E-06
115210	1	399914	2,50054E-06	1000024	9,99976E-07	115208	8,67995E-06
115190	1	399904	2,5006E-06	1000068	9,99932E-07	115193	8,68108E-06
115210	1	400055	2,49966E-06	999992	1,00001E-06	115197	8,68078E-06
115201	1	400083	2,49948E-06	1000081	9,99919E-07	115202	8,6804E-06
115193	1	399943	2,50036E-06	999934	1,00007E-06	115207	8,68003E-06
115209	1	399996	2,50003E-06	1000010	9,9999E-07	115191	8,68123E-06
115193	1	399937	2,50039E-06	1000083	9,99917E-07	115202	8,6804E-06
115203	1	400043	2,49973E-06	1000072	9,99928E-07	115196	8,68086E-06
115194	1	399981	2,50012E-06	999930	1,00007E-06	115196	8,68086E-06
115196	1	399930	2,50044E-06	1000007	9,99993E-07	115196	8,68086E-06
115194	1	399933	2,50042E-06	999928	1,00007E-06	115192	8,68116E-06
115193	1	400095	2,49941E-06	1000078	9,99922E-07	115191	8,68123E-06
115208	1	399950	2,50031E-06	999949	1,00005E-06	115191	8,68123E-06
115201	1	400040	2,49975E-06	999982	1,00002E-06	115207	8,68003E-06
115192	1	400097	2,49939E-06	1000088	9,99912E-07	115192	8,68116E-06
115204	1	400014	2,49991E-06	999934	1,00007E-06	115198	8,68071E-06
115192	1	400067	2,49958E-06	999918	1,00008E-06	115196	8,68086E-06
115200	**1**	**400002**	**2,49999E-06**	**1000004**	**9,99996E-07**	**115200**	**8,68061E-06**

Figura 28: Tabla de pruebas

7.2.2. Arduino-controlador

Es importante señalar que el controlador de vuelo F405-SE dispone de 4 protocolos a la hora de recibir la señal de radio: SBUS, FFPORT, CRSF y PPM. Teniendo en cuenta que entre estos 4 protocolos no hay ninguna diferencia significativa se opta por PPM que es el más popular. Debido a que la frecuencia de los pines de arduino micro es demasiado baja para generar este tipo de señales se necesita un conversor, que incrementa en 5 euros el costo final del proyecto. De este modo, Arduino generará pulsos de PWM que posteriormente se convertirán en PPM por medio del conversor.

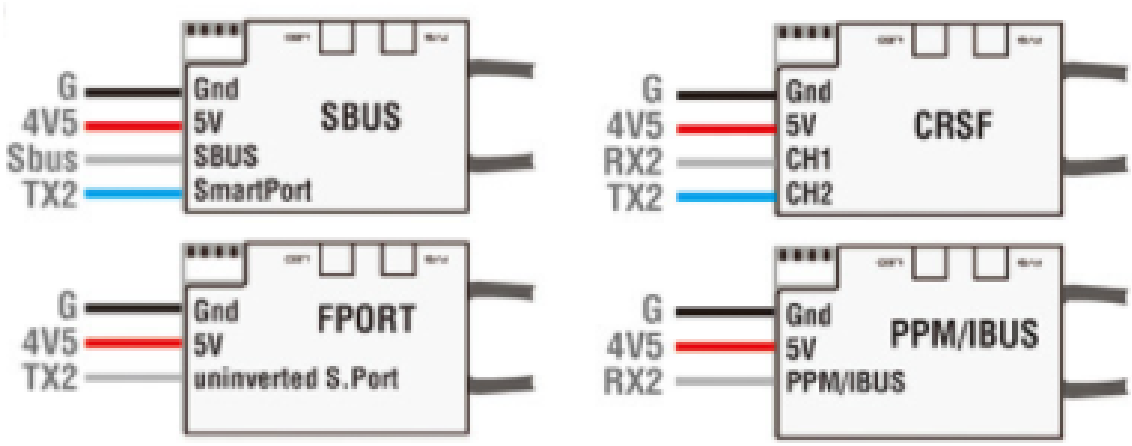

Figura 29: Imagen de las entradas de radio.
Recuperado de http://www.mateksys.com

En cuanto a la experimentación, en este paso solo se ha requerido comprobar la velocidad máxima de actualización en la que funciona el sistema. Con un pequeño programa que pone todos los motores a máxima potencia y luego disminuye la potencia hasta 0 y la aumenta de nuevo. Con este bucle se prevé obtener picos regulares en la simulación cada 200 milisegundos, de modo que puedan distinguirse los picos.

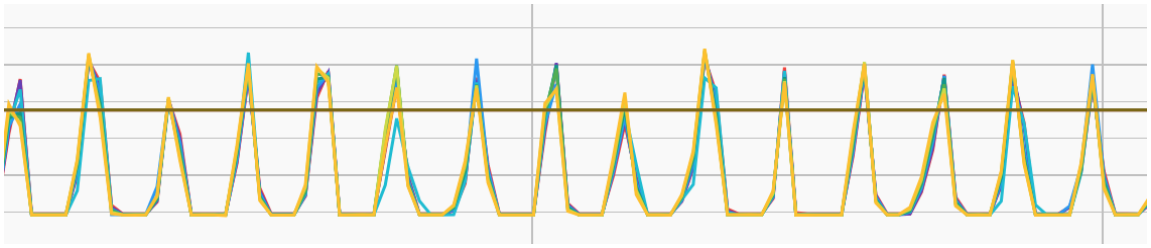

Figura 30: Diagrama de las señales 0ms

El valor, en el eje 'Y', de estos gráficos va de 1000 a 2000, siendo el mínimo y el máximo de potencia respectivamente; en el eje 'X' se señala el tiempo transcurrido. Al observar el diagrama resultante, parece que todo funciona correctamente, con un poco de inestabilidad pero los picos están presentes. El programa de simulación está generando un punto cada 50 milisegundos, al incrementar la velocidad a 10 milisegundos estos errores evidencian mejor.

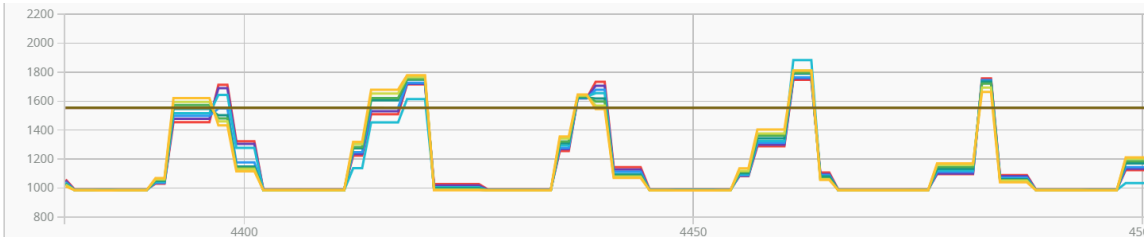

Figura 31: Diagrama de las señales 0ms ralentizado

Teniendo esto en cuenta, y viendo que es muy inestable en estas frecuencias, se prueba con un retardo de 10ms entre una cada comunicación. Este retado se ha elegido porque es la periodo mínimo al que funciona el simulador.

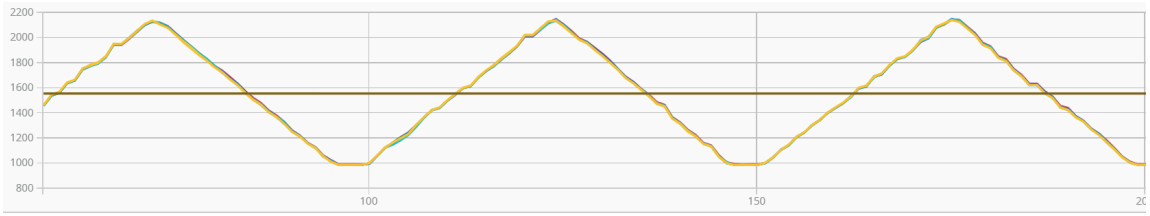

Figura 32: Diagrama de las señales 10ms

Como puede verse en el gráfico siguientes, pese a que las irregulares han desaparecido casi por completo, se siguen dando en ocasiones. Pese a que con las irregularidades que se dan en esta frecuencia de actualización no deberían presentarse problemas, por motivos de seguridad y de estabilidad, se opta por emplear 20ms de retardo.

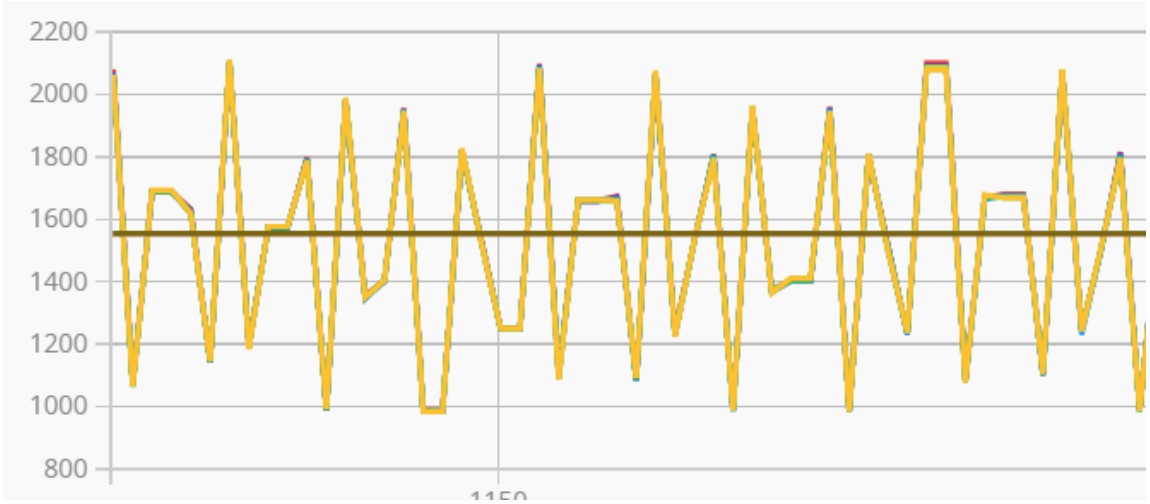

Figura 33: Irregularidad en la señal

Con el fin de comprobar y garantizar que no se produzcan estas irregularidades se programa una sesión de dos hora, mucho más larga que un vuelo promedio. En esta sesión no se presentan estas irregularidades ni en una sola ocasión. El proceso se ha repetido 3 días más para

garantizar que el resultado no ha sido fruto de la casualidad. En las 8 horas de simulación no ha aparecido ninguna irregularidad más.

7.3. Sensores de distancia

A la hora de manejar un dron de forma remota, es recomendable contar con algún sistema de detección de objetos, para prevenir una colisión. Dentro de las opciones, las tres más interesantes son: los sensores de ultrasonidos, los sensores láser y la visión artificial.

A continuación se detallan las características de cada uno de ellos, y cómo se ha tomado la decisión de elegir un sensor de ultrasonidos. En la siguiente tabla se resumen las diferencias.

Visión Artificial	Sensores ultrasonidos	Sensores láser
Permite analizar situaciones complejas.	La medición de distancia abarca un área suficiente.	Mediciones estables de hasta 8 metros.
Puede medir distancia a objetos fuera de la trayectoria.	Muy estable en distancias inferiores a cuatro metros.	Son muy pequeños y generalmente baratos.
Permite añadir funciones como el reconocimiento de imágenes.	Tienen pocas interferencias.	Pueden medir la distancia a intervalos muy cortos
Interesante como primer contacto con la visión artificial.	Poca capacidad de computo necesaria.	Medición de distancia a un área muy pequeña.
Problemas para medir distancias a objetos como paredes.	Mediciones más lentas.	La luz puede producir interferencias.
Necesidad de bastante potencia de computo.	Generalmente mas grandes que los sensores láser.	Se han de conectar a pines especiales en la raspberry.
Necesidad de hardware más caro.		

Cuadro 7: Diferencias entre sistemas de medición

7.3.1. Visión artificial

La visión artificial es una tecnología que se ha popularizado mucho en los últimos años con el incremento de potencia de los ordenadores. La raspberry pi no es el dispositivo más ideal para emplear visión artificial, pero es capaz de hacer cosas sencillas como medir distancias o reconocer objetos dentro de una imagen.

Para medir distancias por medio de imágenes, se usa una proceso conocido como depth mapping. Este proceso convierte las imágenes de dos cámaras ligeramente separadas entre si en un depth map, una imagen que indica por medio de colores la distancia al objeto.

Esta opción es muy interesante para un dron ya que además de cumplir la función de medir distancias, permite añadir nuevas funciones como por ejemplo: predecir trayectorias de objetos en la imagen o reconocer y localizar objetivos en pantalla. Además, no hay necesidad de que las cámaras sean normales, se podrían usar cámaras térmicas para medir la temperatura o cámaras de mejor calidad que permitan, no solo medir distancias sino también sacar fotografías.

Pero como ya se ha especificado en la tabla, cuando se ha analizado esta opción se han detectado muchas desventajas, todas ellas relacionadas con el modo en que funciona el depth mapping. Para poder abordarlas, se resumen a continuación su funcionamiento. Se requieren dos cámaras iguales, enfocando en la misma dirección pero separadas por una distancia conocida. Cada una de estas cámaras emitirá una imagen ligeramente distinta. Gracias a que conocemos la distancia que las separa, se puede triangular la distancia del dron a cada punto de la imagen. En esencia funcionan de igual manera que los ojos. La siguiente imagen ilustra este concepto.

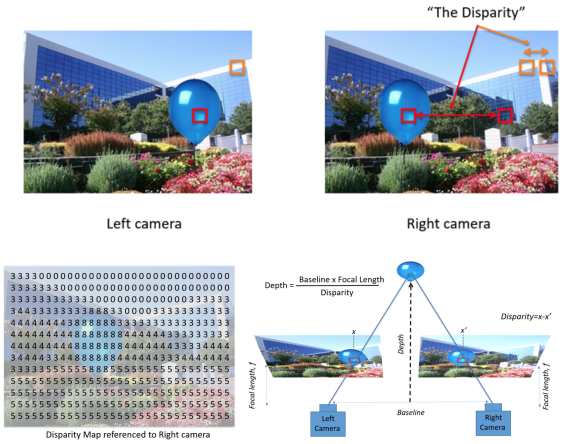

Figura 34: DepthMapping.
Recuperado de hthttps://dev.intelrealsense.com/docs/stereo-depth-cameras-for-phones

Una vez explicado su funcionamiento, se pueden observar los diversos problemas que generaría. No se ha elegido esta opción, además de por el extra de necesitar dos cámaras, por el problema de medir la distancia del dron a objetos que son muy similares en toda su superficie. También implicaría al usuario emplear muchas horas para ajustar las cámaras al dron y que la distancia se mida correctamente.

7.3.2. Sensores láser

Este tipo de sensor ha quedado descartado casi de inmediato teniendo en cuenta que pueden ser muy inestables en situaciones de alta luminosidad, pero en caso de interés, aquí hay una pequeña explicación.

Es casi igual que el sensor de ultrasonido solo que, en lugar de generar un pulso ultrasónico emite un pulso láser de luz infrarroja. Es muy similar, pero, debido a que la velocidad de la luz es mucho mas rápida,

puede hacer más mediciones en un segundo.

Además del inconveniente de la alta sensibilidad lumínica, a diferencia del pulso ultrasónico que cubre un área, este tipo de sensores limitan la medición de la distancia a un ángulo más estrecho. Esto hace que la más mínima inclinación el dron afecte a la distancia que se está midiendo. De este modo, es más probable que el objeto con el que puede producirse el peligro de colisión quede fuera del área abarcado en la medición.

7.3.3. Sensores de ultrasonidos

Estos sensores tienen un funcionamiento muy simple, cuando el sistema lo indica, emite un pulso de ultrasonidos, y cuando ese pulso rebota contra un objeto y regresa, genera un flanco ascendente en uno de sus pines.

Para medir la distancia solo se tiene que contabilizar cuánto tiempo tarda el pulso en llegar al objeto y regresar. Ese tiempo se divide entre la velocidad del sonido para obtener la distancia de ida y vuelta. Y esa distancia se divide entre dos.

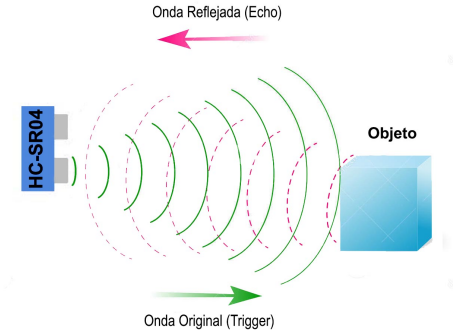

Figura 35: Sensor de ultrasonidos.
Recuperado de hthttps://www.zonamaker.com/arduino/modulos-sensores-y-shields/ultrasonido-hc-sr04

La única desventaja de este tipo de sensores es no que no permiten, a diferencia de las otras dos opciones, medir a intervalos tan reducidos de tiempo ya que funcionan por sonido. Pero los intervalos que no se pueden medir son de centésimas de segundo, lo que no supone un problema para el uso que se le pretende dar en el presente proyecto.

Teniendo en cuenta estas características se opta, como se ha mencionado, por emplear un sensor de ultrasonidos para incorporar al sistema de autopilotaje la detección de objetos.

7.4. Guardado de los vuelos

Por último, se ha de decidir cómo guardar los vuelos. A la hora de discernir la mejor opción, se ha valorado principalmente que fuera fácil de procesar para el piloto automático. También se ha priorizado que permita guardar el valor de cada uno de los motores cómodamente. Por último, aunque con menos relevancia que los factores anteriores, se ha

buscado un método que permita crear y guardar instrucciones de vuelo de forma sencilla sin necesidad de desarrollar una aplicación específica.

Dicho esto, los vuelos se tendrán que guardar en algún tipo de fichero de texto legible tanto para una máquina como para un ser humano, esto descarta el formato binario y hexadecimal. Los ficheros Json y XML son muy populares hoy en día y dos muy buenas opciones para este proyecto. También existe la opción de guardar los vuelos en un fichero de texto plano, pero debido a que Json y XML son en esencia ficheros de texto con un formato especifico, se descarta esta opción desde un principio.

Entre Json y XML no existe ninguna diferencia apreciable. Dado que python trabaja también con Json y que resulta más cómodo, por experiencia y criterio personal se opta por utilizar Json.

Con respecto al almacenamiento de los vuelos dentro del propio fichero, debe ser un formato sencillo de escribir y fácil de leer. El primer paso consiste en decidir los datos imprescindibles que debe contener un vuelo. Con el objeto de simplificar la configuración es importante que cuente con el menor número de atributos. Es por esto que sólo se trabajo con los siguiente atributos:

- **type:** Se trata del tipo de vuelo. Según el tipo que sea tendrá una implementación distinta en el código. Un usuario del programa podría añadir una nueva implementación y crear un nuevo tipo de vuelo si lo desea.
- **times:** Es un atributo que solo se emplea por vuelos en bucle. Sirve para indicar cuántas veces queremos que se repita el vuelo.
- **instructions:** Se trata de una lista de las instrucciones. Ya que

Json permite guardar otros Json como atributos de los mismos, las instrucciones serán una lista de Json.

- **duration/distance:** Es un atributo de cada instrucción. En caso de que se utilice una implementación que funciona por distancia se usa *distance* y si funciona por duración, *duration*. Ambos sirven para determinar cuánto tiempo se han de mantener los valores del data.

- **values:** Es otro atributo de cada instrucción que guarda los valores de cada uno de los canales del dron. Estos valores son un porcentaje, por lo que van de 0 a 100, e indican el nivel de potencia que se desea.

  Dentro de los valores, en general siempre se enviarán tantos valores como canales tenga el dron (8 en este caso), pero se han programado además unos valores especiales que se pueden indicar para programar el vuelo mas cómodamente. Estos valores son los siguientes:

  - **252:** Indica una parada, detendrá todos los motores al momento, en caso de querer evitar una colisión. Es muy útil para una situación de emergencia, pero también se puede utilizar al aterrizar el dron, para detener los motores.
  - **253:** Indica un descenso progresivo, detendrá todos los movimientos y reducirá la potencia para que el dron comience a descender lentamente.
  - **254:** No tiene mucho uso a la hora de programa un vuelo, pero se utiliza cuando se envía un valor incorrecto, para advertir al arduino micro del problema. Sirve para indicar un error.

- **255:** Se emplea cuando solo se desea actualizar alguno de los canales. Sirve para que el arduino micro sepa que se ha concluido la instrucción. Por ejemplo, si el usuario sólo desea actualizar los dos primeros canales, en lugar de reescribir 8 valores, se escriben los dos primeros y se añade este valor especial. De esta forma se mantienen los 6 siguientes canales y se actualizan los dos primeros.

En la siguiente figura se puede observar un ejemplo de vuelo programado. Analizándolo se puede ver que es un vuelo de tipo "loop", y que se repetirá 5 veces. Así, los primeros tres segundo se enviarán los valores "[10,20,30,40,50,60,70,80]"y cuando transcurra el tiempo indicado enviará los siguientes valores. También se pueden ver los valores especiales como 252 o 255, explicados en la lista anterior.

```
{
  "type": "loop",
  "times": 5,
  "instructions": [
    {
      "duration": 3,
      "values" : [10,20,30,40,50,60,70,80]
    },{
      "duration": 3,
      "values" : [80,70,60,50,40,30,20,10]
    },{
      "duration": 3,
      "values" : [252]
    },{
      "duration": 3,
      "values" : [80,70,255]
    }
  ]
}
```

Figura 36: Ejemplo de vuelo programado

7.5. Implementación

Una vez explicado el proceso de selección de cada una de los componentes, se expone en funcionamiento el conjunto. La implementación ha sido algo bastante sencillo, ya que gracias a la exhaustiva investigación de cada una de las partes se ha adquirido una comprensión detallada de su funcionamiento.

Lo primero a realizar durante la implementación son las clases funcionales: SPITransmiter y DistanceSensor. Estas clases se encargan, por medio de librerías propias de Python para la raspberry, de manejar los pines para los sensores y la comunicación a la placa de arduino micro. Las clases funcionales son sencillas, por eso se explican brevemente.

7.5.1. DistanceSensor

Esta clase utiliza la librería RPi.GPIO, de uso público, obtenida directamente de www.raspberrypi.org. Esta librería sirve para facilitar la utilización de los pines.

A la hora de instanciar esta clase se le pasan a la constructora, como parámetros, los pines de trig y echo del sensor respectivamente. A continuación, el propio sistema automáticamente inicializa el sensor y lo deja listo para medir. Para instanciar una clase se utiliza el siguiente código, donde 16 es el pin al que esta conectado el trig del sensor y el 18 el del echo:

```
self.mySens = sensor.distanceSensor(16, 18)
```

El método principal de la clase es distanceInCM, que devuelve la distancia en que está midiendo el sensor correspondiente en centímetros.

Es un método muy sencillo de implementar, pero en caso de querer entrar más en detalle sobre el funcionamiento aquí esta el algoritmo:

Algorithm 1 distanceInCM

```
procedure DISTANCEINCM
    Initilice Sensor
StartTime:
    if GPIO.input(self.ECHO) = GPIO.LOW then
        startTime ← time.time()
        goto StartTime.
EndTime:
    if GPIO.input(self.ECHO) = GPIO.HIGH then
        endTime ← time.time()
        goto EndTime.
    distancia ← (velocidad sonido * (endTime - startTime))/2
    return distancia.
```

En resumen, lo que hace distanceInCM es registrar el tiempo preciso que es cuando se lanza el pulso de ultrasonidos. Una vez el sensor ha recibido de vuelta el pulso, marca el tiempo que es, y con esos tiempos calcula la distancia que hay hasta el objeto.

7.5.2. SPITransmiter

Esta clase, como DistanceSensor, utiliza la librería RPi.GPIO. Además también utiliza la librería spidev para gestionar las conexiones SPI. La librería es también de uso público y se puede obtener de pypi.org/project /spidev/.

Para esta clase, la constructora requiero de tres parámetros. El primero es módulo de SPI, y será cero o uno, ya que la raspberry pi dispone

de dos módulos distintos. El segundo es el dispositivo, como es el que manda instrucciones será cero. El tercero es la frecuencia, se puede modificar, pero no es recomendable por las limitaciones de arduino micro. Todos los parámetros son opcionales y sus valores por defecto son módulo = 0, dispositivo = 0 y máxima frecuencia = 1MHz. Un ejemplo de como instanciar esta clase:

```
self.mySPI = SPIT.SPITransmiter(maxHz = 500000)
```

A diferencia de los sensores, que cada uno necesita una instancia de la clase; con un solo SPITransimiter podemos comunicarnos con el arduino micro ya que solo habrá uno.

El método principal de esta clase es bastante más complejo, por lo que se ha considerado de interés detallar más su funcionamiento. Con la ayuda del método "xfer()" de la librería mencionada, la transferencia es sencilla. Este método envía un valor y devuelve el valor que arudino micro recibe en la anterior transferencia. Entonces se guarda el valor enviado y se compara con el siguiente que se recibe. Si son iguales significa que está funcionando correctamente y si no lo son, se trata de un error y se muestra un mensaje de error. Los mensajes de error se envían desde su propio método.

Aquí esta el pseudocódigo del método para ver un poco más en detalle la explicación. Data es el único parámetro de este método y se le llama hasta que devuelve True con la siguiente instrucción.

```
while (not self.mySPI.sendData(decData))
```

Algorithm 2 sendData

```
procedure SENDDATA
    sendedNoErr ← True
    for val in data do
        lastReturned = spidev.xfer(val)
        if prevRet! = sendedand notfirstData then
            sendError()
            sendedNoErr ← False
            break
        sended ← val
        firstData ← False
        sleep(0.01)
    return sendedNoErr.
```

7.5.3. Vuelos, ficheros y main

Para estas clases no se hará mucho hincapié en el funcionamiento, ya que están explicadas en la sección "Diagramas de secuencia"xswa.

Las clases de vuelos se encargan de gestionar la lógica de los vuelos. Ya que cada una de ellas tiene un funcionamiento distinto pero con los mismos parámetros de entrada y de salida, son muy sencillas de utilizar. Lo que si que es importante conocer es que se crean desde la clase FlightFactory que se encarga, de forma automática, de reconocer qué tipo de vuelo hay en el fichero.

El lector de fichero se encarga de leer el fichero Json que corresponda, verificar que está bien, con todos los campos necesarios, y almacenar el contenido.

Por último, el main contiene el menú y la consola de comandos y se encarga de iniciar cada una de las funcionalidades del código.

7.5.4. Unión de las distintas partes del código

Lo primero para hacer funcionar el código, es es crear un SPITransmiter e instanciar un DistanceSensor para cada uno de los sensores que se dispongan.

Después de instanciar esas clases, se verificara que el Json sea correcto, y en caso de serlo, la clase FlightFactory obtendrá el tipo de vuelo que se quiere crear, llamando al método "createFlight" y pasando como parámetros el FileProcessor, el SPITransmiter y la lista de sensores.

Una vez generado el vuelo, se llama al método "fly" del mismo y se inicia el vuelo. Un vez finalizado el vuelo, en caso de estar funcionando por medio de menú, se le envía al usuario de nuevo a la pantalla principal.

8. Pruebas y conclusiones del sistema

Con el fin de evitar errores inesperado se prepara un plan de pruebas para cuando el proyecto este finalizado. Estas pruebas se acaba dividendo en tres partes, pruebas sobre la lectura del fichero, pruebas del envió de datos y pruebas del sistema completo.

8.1. Pruebas sobre la lectura del fichero

Con estas pruebas se planea comprobar que, si se producen fallos en el fichero de vuelo, independientemente del tipo y gravedad de esos fallos, el sistema no avanzará a la siguiente etapa del proceso.

En la siguiente tabla se encuentran tanto las pruebas como los resultados de la sección del plan de pruebas que corresponde al fichero.

Definición	Resultado Esperado	Resultado Obtenido
Falta del atributo Tipo	Error	Error
El atributo Tipo contiene un valor no válido	Error	Error
El fichero Json no existe	Error	Error
Falta el atributo Instrucciones	Error	Error
Una instrucción en el atributo Instrucciones carece del campo Data	Error	Error
Una instrucción en el atributo Instrucciones carece del campo Duration cuando no se necesita	Correcto	Correcto
Una instrucción en el atributo Instrucciones carece del campo Distance cuando no se necesita	Correcto	Correcto
Una instrucción en el atributo Instrucciones carece del campo Duration cuando se necesita	Error	Error
Una instrucción en el atributo Instrucciones carece del campo Distance cuando se necesita	Error	Error
Falta el campo Iimes cuando se necesita	Error	Error
Existe el campo Times cuando no se necesita	Error	Error
El fichero es correcto	correcto	correcto

Cuadro 8: Plan de pruebas lectura de fichero

Viendo estas pruebas podemos concluir que el sistema de lectura de fichero es bastante robusto, aunque existen situaciones que no se están contemplando, por ejemplo que los datos no sean válidos o que la duración o distancia sean negativas. Estas situaciones podrían perjudicar al sistema en determinadas ocasiones. Idealmente próximas versiones del código podrían prevenir este tipo de errores.

8.2. Pruebas en el envío de datos

En lo que respecta a el envió de datos,las pruebas son bastante sencillas, constando solo de tres distintas:

- **Que los datos enviados no sean válidos:** Aunque al final el sistema permite enviar datos fuera de los márgenes válidos, se espera que el sistema no falle. Probando esto, en la simulación se observa que puede generar alguna anomalía, pero, aun con valores fuera de los límites, el sistema no se estropea.
- **Que se envíen datos correctamente:** En este caso esperamos que al enviar los datos, los cambios se vean reflejados en la simulación. En el caso de la prueba, el resultado obtenido coincide con el esperado, salvo que los valores sean incorrectos.
- **Que se envíen datos incorrectamente:** Cuando sucede que los datos no se envía correctamente, aparece un mensaje de error, tal y como está previsto.

Tras la realización de las pruebas, se puede decir que el sistema de vuelo no generará problemas durante el envió de las instrucciones si son correctas. Aunque como ya hemos mencionado previamente, añadir al procesador de los ficheros la capacidad de ver si los valores son correctos evitaría estos problemas.

8.3. Pruebas del sistema completo

Del sistema completo se comprueba que las funcionalidades de los vuelos actúan como se ha previsto. Para garantizar la seguridad del sistema se preparan unas pruebas que abarcan la mayor cantidad de casos posibles. En la siguiente lista están las pruebas que se han realizado para cada una de las funcionalidades:

- **El vuelo no es correcto**: En este caso el resultado esperado es que no se inicie el vuelo.
- **El vuelo es correcto con una sola instrucción**: En este caso el resultado esperado es que se inicie el vuelo y una vez finalizada la instrucción, se muestre un mensaje.
- **El vuelo es correcto con más de una instrucción**: Se espera que las instrucciones se ejecuten en orden.
- **El vuelo es correcto y una instrucción contiene algún valor especial**: Se espera que las instrucciones se ejecuten en orden.
- **El vuelo es correcto y es de bucle**: Se espera que, en caso de que la funcionalidad de sea la adecuada, las instrucciones se ejecuten en orden y se repitan el número de veces que se indica.
- **El vuelo es correcto y usa sensores**: Se espera que si el sensor detecta una posible colisión, detenga el movimiento del dron en la dirección del sensor. Un sensor frontal no detendrá un movimiento en otra dirección que no sea hacia delante.

Todas las pruebas anteriores se pasan con éxito.

8.4. Conclusiones del proyecto

En general el resultado obtenido funciona como se esperaba y tal y como se planeó. Se han cumplido la mayoría de objetivos principales y algunos de los secundarios. Además, ha constituido un gran oportunidad de aprendizaje. Se han completado aproximadamente el 90 % de los objetivos.

En lo que respecta al presupuesto, ha habido un incremento de precio de los materiales de entorno al 7 % por gasto de envío. Además, el precio del conjunto se ha incrementado un euro por la necesidad de conversores de 3 a 5 Voltios.

Durante el desarrollo del proyecto, se han encontrado algunos de los problemas que ya se han mencionado en la sección de riesgos, pero que, gracias a una buena planificación, no han supuesto un gran contratiempo. Además, gracias a un buen uso de las herramientas disponibles, se ha conseguido mejorar la calidad del software y crear un código simple y fácil de entender. Se considera un valor añadido haber diseñado cinco funcionalidades de vuelo distintas. Por último, se ha observado que el sistema es capaz de ejecutar los vuelos con éxito en el 100 % de las ocasiones, y los sistemas de emergencia controlan el dron correctamente cuando el vuelo falla

Pese a que originalmente este proyecto fuera a orientarse hacia la creación de un dron completo, ha acabado siendo un trabajo de investigación. Se han probado minuciosamente muchas de las opciones disponibles para cada parte del sistema. Se ha obtenido un producto completo por debajo del presupuesto. Además, se ha logrado diseñar un sistema sencillo y ligero que se podrá instalar en la mayoría de los drones, tal y como se ha planeado desde el inicio.

9. Posibles mejoras futuras

En caso de retomar este proyecto, se proponen en esta sección algunas mejoras que podrían incorporarse.

Uno de las principales carencias de este sistema es la de una aplicación que permita programar los vuelos visualmente. Esto no se ha realizado dado que supondría prácticamente el diseño de otro proyecto completo y no es imprescindible para el funcionamiento. Pero podría ser de gran ayuda para el usuario.

Mediante el sistema diseñado, en caso de que el dron fallara, aterrizará en el lugar en que se encuentre, pero habría sido interesante poder recuperar el control del dron. Además, esto se podría lograr en cualquier momento, dándole a un botón destinado a retomar el control del dron.

Añadir nuevas funcionalidades de vuelo e introducir avances en las actuales, también son mejoras que se podría incorporar. Más allá de las nuevas funcionalidades, se podría implementar una página web que permita manejar el dron de manera remota y subir los vuelos desde la misma.

Por último y como ya se ha mencionado, sería interesante actualizar el sistema de validación de los ficheros Json con el fin de evitar que se introduzcan valores no válidos. También se podrían verificar campos adicionales que el usuario podría añadir. Sumado a esto, constituiría también una mejor convertir SPITransmiter, DistanceSensor y FileProcesor en Singletons.

Referencias

[1] treyes4: DIY ARDUINO FLIGHT CONTROLLER, https://www.instructables.com/DIY-ARDUINO-FLIGHT-CONTROLLER/ (Accedido a 24 de julio de 2021)

[2] unknown: Project YMFC-32 - The STM32 quadcopter, http://www.brokking.net/ymfc-32_main.html (Accedido a 24 de julio de 2021)

[3] FScreations, 23 de diciembre de 2018: DIY RC Transmitter and Receiver for RC plane,helicopter,drone,car etc using arduino, http://www.brokking.net/ymfc-32_main.html (Accedido a 24 de julio de 2021)

[4] Kris Winer, Greg Tomasch, 23 de agosto de 2018: SuperFly Hackable ESP8266 Flight Controller, https://hackaday.io/project/160674-superfly-hackable\ -esp8266-flight-controller (Accedido a 24 de julio de 2021)

[5] Oscar Liang, 7 de abril de 2021: DIY PWM to PPM Converter for 2.4GHz Receiver using Arduino, https://oscarliang.com/build-pwm-ppm-converter-arduino-2-4ghz-recei (Accedido a 24 de julio de 2021)

[6] Ben, 6 de junio de 2014: Three Ways To Read A PWM Signal With Arduino, https://www.benripley.com/diy/arduino/ three-ways-to-read-a-pwm-signal-with-arduino/ (Accedido a 24 de julio de 2021)

[7] Roger Connor, 9 de marzo de 2018: Three Ways To Read A PWM Signal With Arduino, https://airandspace.si.edu/stories/editorial/

predator-drone-transformed-military-combat (Accedido a 24 de julio de 2021)

[8] Saúl García, 6 de julio de 2020: Three Ways To Read A PWM Signal With Arduino, https://blog.330ohms.com/2020/07/09/como-conectar-arduino\-y-raspberry-pi-por-comunicacion-spi/ (Accedido a 24 de julio de 2021)

[9] ardupilot.org: ardupilot.org, https://ardupilot.org (Accedido a 24 de julio de 2021)

[10] Saúl García, 6 de julio de 2020: Three Ways To Read A PWM Signal With Arduino, https://blog.330ohms.com/2020/07/09/como-conectar-arduino\-y-raspberry-pi-por-comunicacion-spi/ (Accedido a 24 de julio de 2021)

[11] raspberrypi.org: Documentación librería GPIO, https://www.raspberrypi.org/documentation/usage/gpio/ (Accedido a 24 de julio de 2021)

[12] pypi.org: Librerias python, https://pypi.org (Accedido a 24 de julio de 2021)

[13] mateksys.com: Documentacion controlador de vuelo, http://www.mateksys.com (Accedido a 24 de julio de 2021)

[14] Anders Grunnet-Jepsen, John N. Sweetser, John Woodfill,2020: Stereo depth cameras for mobile phones, https://dev.intelrealsense.com/docs/stereo-depth-cameras-for-phones (Accedido a 24 de julio de 2021)

[15] Raúl Diosdado: Sensor de ultrasonidos HC-SR04, https://www.zonamaker.com/arduino/modulos-sensores-y-shields/ultrasonido-hc-sr04 (Accedido a 24 de julio de 2021)