UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Tècnica Superior d'Enginyeries
Industrial i Aeronàutica de Terrassa

TRABAJO FINAL DE GRADO

Desarrollo de una placa de control de actitud 2D para un prototipo de CubeSat con pares magnéticos y ruedas de inercia.

MEMORIA

Grado en Ingeniería Electrónica Industrial y Automática

Autor: Morales Krueger, Jordan

Director: Gago Barrio, Javier

Codirector: Lamich Arocas, Manel

Entrega: 22/06/2021

Abstract

The research group DISEN (Distributed Sensor Networks) of the Universitat Politècnica de Catalunya (UPC) in Terrassa, through the PLATHON (Integrated Hardware in the loop simulation PLATform of Optical communications in Nanosatellites) project, is aiming to carry out a simulation of a CubeSat system that collects information from IoT sensors.

This project relies on the study and design of the attitude determination and control system (ADCS) based on its own torque rod technology in order to diminish the cost of its assembly. The CubeSat must vary its position to be able to focus on another CubeSat to communicate via optical communication.

To do this, the CubeSat is introduced into the Air bearing located in the center of a magnetic simulator, that produces a magnetic field similar to the position of the satellite in its low altitude orbit (LEO). The on-board computer (OBC) of the nanosatellite communicates with a computer and exchanges information. Once the magnetic field is obtained with the simulator, the magnetocouple control is activated to rotate the desired attitude. Finally, the attitude measured with the inertial measurement unit (IMU) of the ADCS is sent to the computer to display the results.

Keywords

CubeSat, ADCS, Magnetorquers.

Resumen

El grupo de investigación DISEN (Distributed Sensor Networks) de la Universitat Politècnica de Catalunya (UPC) en Terrassa, a través del proyecto PLATHON (Integrated Hardware in the loop simulation PLATform of Optical communications in Nanosatellites), tiene como objetivo realizar una simulación de un CubeSat sistema que recopila información de los sensores IoT.

Este proyecto se basa en el estudio y diseño del sistema de determinación y control de actitud (ADCS) basado en su propia tecnología de barra de torsión con el fin de disminuir el costo de su montaje. El CubeSat debe variar su posición para poder enfocarse en otro CubeSat para comunicarse mediante comunicación óptica.

Para ello, el CubeSat se introduce en el Air bearing ubicado en el centro de un simulador magnético, que produce un campo magnético similar a la posición del satélite en su órbita de baja altitud (LEO). El ordenador de abordo (OBC) del nanosatélite se comunica con un ordenador e intercambia información. Una vez que se obtiene el campo magnético con el simulador, se activa el control del magnetopar para rotar la actitud deseada. Finalmente, la actitud medida con la unidad de medida inercial (IMU) del ADCS se envía a la computadora para mostrar los resultados.

Palabras Clave

CubeSat, ADCS, Magnetorquers.

Agradecimientos

Me gustaría dedicar unas palabras a todas las personas que me han acompañado en este largo camino.

En primer lugar, me gustaría agradecer a mi familia, que me ha felicitado en los buenos momentos y me ha ayudado a salir adelante en los malos. Ha sido crucial el apoyo recibido en tiempos en los que daba por perdidas muchas cosas, así que por ello doy las gracias.

En segundo lugar, agradecer a mis amigos, tanto a los de siempre como a los que he ido haciendo a lo largo de estos años, por hacerme desconectar cuando los necesitaba y ayudarme dándome tantos buenos momentos.

Por último, me gustaría dar las gracias a los coordinadores del proyecto, concretamente a mi tutor Javier Gago y mi cotutor Manel Lamich, por el tiempo dedicado y la paciencia que han tenido conmigo.

Una vez más, gracias a todos.

Índice general

Índice de tablas

Índice de figuras

INTRODUCCIÓN

Objetivo

La finalidad de este proyecto es el diseño, fabricación e implementación de una placa ADCS (*Attitude and Control Determination System)* para un prototipo de CubeSat con un grado de libertad basado en pares magnéticos y rueda de inercia. Se tendrán que diseñar y fabricar los pares magnéticos y, además, se deberá realizar un estudio de los componentes que forman parte de la placa para su posterior diseño de software y evaluación de funcionamiento experimental, incluyendo una plataforma de visualización de datos mediante gráficas en tiempo real.

Justificación y utilidad

El control de actitud surge por la necesidad de mantener dos CubeSats enfocados entre sí de manera precisa para realizar un intercambio de datos por medio óptico (láser en el emisor y fotodiodo en el receptor). De esta forma se consigue una mayor tasa de envío de información con un consumo relativamente bajo.
Dicho esto, se debe realizar el control del satélite para, conociendo la posición del satélite con el que se desea comunicar, girar un ángulo determinado y, de esta forma, poder realizar la comunicación óptica.

Este trabajo está enfocado en la implementación de cálculos teóricos de trabajos anteriores. En ellos ya se han escogido los diferentes componentes necesarios para la placa (drivers de corriente, IMU, microcontrolador) todos ellos de bajo coste y con el único fin de realizar una simulación hardware *in-the-loop*, una técnica de simulación que consiste en realizar pruebas a tiempo real en sistemas controlados para ver resultados de una forma veraz y concluyente mediante prototipo. Para ello, en otro proyecto paralelamente se fabricará un entorno de simulación en el que se reproducirá el campo magnético de una órbita LEO (*Low Earth Orbit*).

Requisitos

Los principales requisitos que se tienen que cumplir en este proyecto son:

- La PCB deberá incluir todos los componentes electrónicos necesarios para el control, incluyendo 2 magnetopares y una rueda de inercia, ajustándose al tamaño del cubesat (10x10x10cm) y teniendo en cuenta que se deberá dejar espacio para el cableado.

- El layout de los componentes deberá cumplir unas condiciones concretas para satisfacer el comportamiento de la placa y no crear discordancias entre componentes que se puedan interferir.

- La PCB deberá cumplir con las condiciones que exige el fabricante para que su fabricación sea posible.

- El control del sistema estará gobernado por un microcontrolador BluePill, que podrá comunicarse de forma inalámbrica mediante Bluetooth con el ordenador.

- Será necesario el diseño de una interfaz HMI en la que se pueda controlar el modo de trabajo junto con el ángulo consigna y visualizar los resultados gráficamente.

Alcance

Para lograr los objetivos establecidos, se deberá seguir una serie de tareas que definirán la trascendencia de este proyecto. A continuación, se muestra un listado.

- Realización de un estudio teórico del control de actitud de un CubeSat.

- Realización de un estudio teórico de las barras de torsión, concretando un diseño que cumpla con los requisitos del sistema.

- Realización de un estudio teórico y posterior configuración mediante un primer prototipo de los componentes electrónicos de la placa ADCS.

- Diseño del esquemático de la PCB y elección de los componentes discretos del sistema.

- Contactar con el fabricante de la PCB y construcción de la placa.

- Se deberá llevar la placa a un entorno de simulación para comprobar su correcto funcionamiento.

CAPÍTULO 1: ESTADO DEL ARTE

En este capítulo se hará una explicación del concepto de CubeSat, así como los principios en los que se basa y qué tipos de control de actitud existen hoy en día. Además, se hablará sobre qué hace a los CubeSats una tecnología en auge y con tanto futuro.

1.1 CubeSat

Un CubeSat es una estandarización de un nanosatélite (satélite con un peso de entre 1-10 kg) que no supera las dimensiones de 10 centímetros de arista y no pesa más de 1,33 kilogramos. Las especificaciones del CubeSat surgen por primera vez entre la California Polytechnic State University y la Universidad de Stanford, que nacen a raíz de la necesidad de una tecnología al abasto económico de las universidades que permitiera realizar proyectos de investigación espaciales en órbitas LEO (*Low Earth Orbit*).

Los CubeSats poseen la principal ventaja de que, por su simplicidad, disminuyen en gran cantidad su dificultad de diseño por lo que se pueden usar en proyectos de corto plazo respondiendo con alta eficiencia. Gracias también a su tamaño reducido de 10x10x10cm, no supone una carga importante a la hora de su lanzamiento. Se pueden compactar diferentes unidades escalando en incrementos de media o una unidad. En algunos proyectos se han lanzado CubeSats de dos unidades 2U (20x10x10cm) y 3U (30x10x10cm).

Todos estos factores comportan una reducción económica importante respecto a otros satélites, con una media oscilando entre 50.000$ y 100.000$, dependiendo de su complejidad y teniendo en cuenta de que cada vez se reduce más el coste de lanzamiento debido a las nuevas tecnologías.

En cuanto al despliegue de este tipo de satélites, al tener un estándar de 10 cm^2 de área, se lleva a cabo mediante un mecanismo llamado *Poly-PicoSatellite Orbital Deployer* (P-POD). El funcionamiento es básico: se montan los P-POD en el vehículo de lanzamiento y, cuando se recibe una señal proveniente del vehículo de lanzamiento, se despliegan los satélites. A partir de aquí se realizan las diferentes maniobras de control.

Ilustración 1: Poly-PicoSatellite Orbital Deployer (P-Pod). Fuente: [1].

1.2 Sistemas de control de actitud

El control de actitud es el conjunto de maniobras necesarias para realizar el control de un objeto con respecto a un sistema de referencia marcado. Para ello se requieren tres elementos básicos: los sensores para medir las condiciones del entorno, los actuadores para aplicar los torques necesarios para realizar la maniobra correspondiente y los algoritmos que determinan el control de los actuadores basándose en las medidas obtenidas por los sensores.
Así pues, los objetivos principales del sistema de control de actitud son:

- Alcanzar la actitud deseada y estabilizar el sistema en dicha actitud deseada.
- Reducir el impacto de las perturbaciones externas.

Para la determinación del control de actitud existen varios métodos con sus propias ventajas e inconvenientes, que se dividen en dos grupos según el procedimiento:

- **Sensores de actitud relativa:** este control se basa en sensores que producen señales que indican el cambio de actitud de forma constante y, comparando con una actitud inicial conocida y el ritmo de cambio, se calcula el control necesario. Este tipo de sensores cuentan con la desventaja de señales ruidosas producidas por elementos externos a los sensores y que deben ser corregidos para obtener la precisión deseada.
- Giroscopios: los giroscopios detectan la rotación del cuerpo, independientemente de la observación de cuerpos externos. Hay diferentes tipos de giroscopios: el clásico que consiste en una masa que rota sobre sí

misma, el giroscopio láser que se basa en el reflejo de un láser en un campo cerrado y el giroscopio por resonador semiesférico que se basa en el cálculo de vibraciones sobre un objeto semiesférico, parecido a una copa.

- **Sensores absolutos de actitud:** este tipo de sensores determina el control mediante efectos físicos, por lo que no requieren ningún tipo de algoritmo para su funcionamiento. De este tipo de sensores se destacan los siguientes:

- Sensor de horizonte: se basa en la detección de la luz del borde de la atmósfera de la Tierra.

- Girocompás orbital: se usa un péndulo para medir la gravedad local y hacer que su giroscopio se alinee con el Norte de la Tierra mediante el vector spin de esta.

- Sensor solar: se basa, como su nombre indica, en la posición del Sol. Su complejidad depende de los requisitos de la misión.

- Sensor terrestre: como en el sensor solar, el funcionamiento del sensor terrestre se basa en la dirección de la Tierra mediante una cámara infrarroja.

- Star tracker: los Star trackers son dispositivos que miden la posición de las estrellas mediante fotodetectores. Normalmente para este tipo de sensores se incluye una base de datos de campos de estrellas para evitar perturbaciones de fuentes luminosas que puedan causar cálculos de posición erróneos como la luz de los planetas, supernovas, cometas, etc.

- Magnetómetro: el magnetómetro mide la intensidad y dirección del campo magnético terrestre para calcular la posición relativa del satélite.

En cuanto al control de actitud, se puede llevar a cabo con diferentes mecanismos o actuadores como por ejemplo los propulsores, velas solares, giroscopio de control de momento, etc. Entre ellos, los más relevantes en las aplicaciones para CubeSats son los siguientes:

- Ruedas de inercia: las ruedas de inercia o volantes de reacción son dispositivos que se basan en el principio físico de la conservación de momento angular. La rueda de reacción gira cada vez más rápido, por lo que, por la conservación del momento angular, el satélite se moverá de forma proporcional en el sentido opuesto al giro. Este tipo de movimiento solo permite girar el satélite sobre su centro de gravedad y en ningún caso permitirá movimiento de traslación.

La principal ventaja de las ruedas de inercia es su rapidez y precisión a la hora de actuar.

- Pares magnéticos: Los pares magnéticos o magnetorquers son muy usados en órbitas LEO, donde el campo magnético terrestre es significante. Básicamente son bobinas en las que, cuando se les induce una corriente, crean un dipolo magnético que interactúa con el campo magnético terrestre generando un torque útil. Esto permite al satélite moverse únicamente con electricidad. El inconveniente que tienen respecto a las ruedas de inercia son la lentitud con la que actúan y la poca precisión. Además, solo tienen dos grados de libertad al basarse en la interacción con el campo geomagnético. De todas formas, la combinación de este con las ruedas de inercia ayuda a la estabilización del sistema ante la saturación (ver capítulo **5**).

CAPÍTULO 2: DISEÑO Y FABRICACIÓN DE LOS MAGNETOPARES

El presente capítulo se centra en el diseño de los pares magnéticos que se van a usar para la simulación. El procedimiento será hacer un estudio teórico con el objetivo de calcular el número de vueltas del devanado para su posterior fabricación.

2.1 Fundamento teórico

Tal y como se ha explicado en el apartado (**1.2**) los magnetorquers son bobinas que, cuando se les induce un corriente, crean un dipolo magnético que interactúa con el campo magnético terrestre generando un torque que hace que gire el CubeSat de manera controlada dependiendo de la intensidad suministrada. El dipolo magnético generado puede ser calculado según la siguiente fórmula [**2**]:

$$m = N\,I\,A \tag{2.1}$$

Donde *N* es el número de vueltas de la bobina, I es la corriente eléctrica que recorre el hilo conductor (A) y A es el área de la sección transversal del bobinado (m²). El dipolo generado, proporcional a la intensidad de la corriente suministrada, genera un torque al interactuar con el campo magnético terrestre siguiendo la expresión:

$$\vec{\tau} = \vec{m} \times \vec{B} \tag{2.2}$$

Donde τ es el torque generado (Nm), m es el momento dipolar magnético (Am²) y B el campo magnético externo terrestre (T). El módulo del torque generado sería:

$$\tau = m\,B\,\sin(\theta) \tag{2.3}$$

De aquí podemos deducir que el par máximo se consigue cuando $\tau = 90º$ ya que el seno de 90 grados es 1. De esta manera, se debe alimentar a los magnetorquers de manera que formen un dipolo siempre normal al vector campo magnético terrestre.

Es importante destacar que el torque generado será perpendicular al eje de la bobina y el vector del campo magnético generado, como se muestra en la siguiente figura:

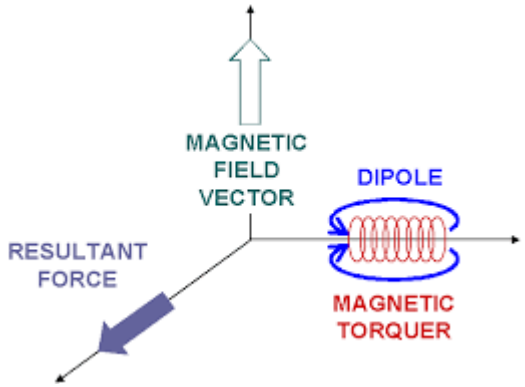


Ilustración 2: Principio de funcionamiento de un par magnético. Fuente: [3]

Otro factor importante a tener en cuenta es la permeabilidad magnética. La permeabilidad magnética es la capacidad de un material de atraer las líneas de campo magnético. Esta propiedad que poseen algunos metales nos permitirá incrementar de manera sustancial el dipolo magnético generado. En el proyecto de Oriol Laserna, *Design of a cubesat attitude 2D control based on the Earth's magnetic field* [**4**], se escoge una barra de ferrita como núcleo de la bobina, con una permeabilidad magnética (μr) de 125.

2.2 Cálculo del número de vueltas

La expresión del momento generado por el dipolo magnético contenido en el núcleo se describe por la ecuación [**5**]:

$$\vec{m} = N\,I\,A\left[1 + \frac{\mu_r - 1}{1 + N_d\,(\mu_r - 1)}\right] \tag{2.4}$$

Donde N es el número de vueltas (incógnita), I es la intensidad que recorre por el hilo conductor (A), A es el área de la bobina (m²), μres la permeabilidad magnética relativa y N_d el factor de desmagnetización (descrito en la fórmula 2.5). Para el cálculo del factor de desmagnetización necesitamos saber las dimensiones del núcleo. En el proyecto de Oriol Laserna se escoge la barra de ferrita con una longitud de 9 cm y un diámetro de 8 mm. No obstante, para el prototipo se decide usar una barra del mismo material pero de 4,5 cm de longitud y 8 mm de diámetro [**6**]. Teniendo en cuenta la dimensión del cable de cobre que se usará (0,25 mm de diámetro) y la longitud de la barra de ferrita, se calcula que habrá que hacer entre unas 8-10 capas de devanado para conseguir el par magnético máximo deseado, de 0,2 Nm. Así, suponiendo el caso en el que se bobinan 8 capas de hilo, se obtiene un resultado final de 12 mm de

diámetro. Teniendo en cuenta que el bobinado no estará perfectamente alineado, añadimos un factor de error de precisión de 2.5, por lo que cada capa actuaría con un grueso de 0,625 mm aproximadamente teniendo en cuenta el margen de error.

$$N_d = \frac{4\left[\ln\left(\frac{l}{r}\right)-1\right]}{\left(\frac{l}{r}\right)^2-4\ln\left(\frac{l}{r}\right)} = \frac{4\left[\ln\left(\frac{4{,}5\times10^{-2}}{6{,}5\times10^{-3}}\right)-1\right]}{\left(\frac{4{,}5\times10^{-2}}{6{,}5\times10^{-3}}\right)^2-4\ln\left(\frac{4{,}5\times10^{-2}}{6{,}5\times10^{-3}}\right)} = 0.093045 \quad (2.5)$$

Obtenemos un factor de desmagnetización N_d de 0,093045 siendo l la longitud del núcleo de ferrita (m) y r su radio (m). A partir de este cálculo, simplificamos la fórmula (2.4), de manera que obtenemos lo siguiente:

$$m = N\,I\,\vec{A}\,(1+\beta) \quad (2.6)$$

Donde el factor βtiene un valor expresado en la siguiente ecuación:

$$\beta = \frac{\mu_r-1}{1+N_d(\mu_r-1)} = \frac{124}{1+0{,}093045\cdot124} = 9,97 \quad (2.7)$$

Finalmente, mediante la siguiente expresión, obtenemos el valor del número de vueltas para un par máximo. Para ello, supondremos una intensidad máxima de entre 100 y 200 mA, ya que tendremos una alimentación de 5 voltios ideales y debemos tener en cuenta las pérdidas del circuito. Se escoge un valor conservador, de unos 125 mA, obteniendo el siguiente resultado para el número de vueltas:

$$N = \frac{\vec{m}}{I\,A\,(1+\beta)} = \frac{0{,}2}{0{,}125\cdot\pi\left(6{,}5\times10^{-3}\right)^2(1+9{,}97)} = 1098,85 \quad (2.8)$$

Obtenemos un resultado de 1098,85. Actuando de manera lógica, redondeamos a unas 1100 vueltas.

N = 1100 vueltas

Una vez sabemos el número de vueltas necesario para conseguir las especificaciones deseadas, procedemos a su fabricación. Para ello, usamos la barra de ferrita de 4,5 cm de largo y 8 mm de diámetro, junto con el hilo de cobre de 0,25 mm de diámetro.

Ilustración 3: Barra de ferrita junto con el hilo de cobre. Fuente: [6][7]

Los magnetorquers se construyen finalmente en el *Fablab* de la ESEIAAT. Se devanan 8 capas obteniendo aproximadamente 1100 vueltas y una resistencia de unos 14Ω. El resultado se muestra en la siguiente imagen:

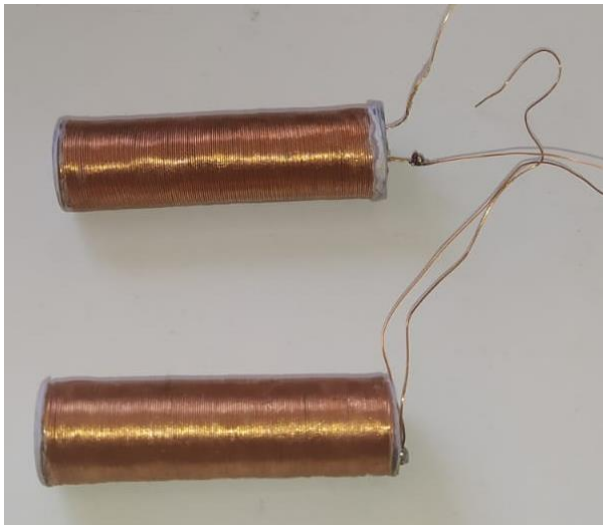


Ilustración 4: Resultado final de los magnetorquers.

CAPÍTULO 3: ESTUDIO DEL SISTEMA Y PROTOTIPO

En este capítulo se realizará un estudio completo del sistema de actitud, en el que se detallará el funcionamiento de cada subsistema por separado con el fin de entender el conjunto.
Se hará una distinción de las etapas tanto de comunicación como de actitud y control, haciendo un análisis de cada uno de ellos.
A continuación, se mostrarán los pasos seguidos para realizar el prototipo antes de hacer el diseño de la PCB, profundizando tanto en hardware como software.
Para el hardware, se hará una explicación del conexionado eléctrico para el correcto funcionamiento del sistema (alimentación, comunicación, etc.).
En cuanto al software, se mostrarán los programas que realizan el control del sistema, así como la configuración previa de todos los elementos que la necesitan para su uso (calibración, asignación de pines, establecimiento de conexión, etc.).

3.1 Control del sistema

La placa ADCS estará gobernada por un microcontrolador, que se encargará de comunicarse con todos los elementos y así recibir la información necesaria para controlar el sistema. En este trabajo se dispone de una BluePill, basada en el microcontrolador STM32F103C8T6 de 32 bits y tecnología ARM «(*Advanced RISC Machine*) o máquina RISC avanzada, donde RISC significa *Reduced Instruction Set Computer*» [**8**]
El microcontrolador BluePill, también referido comúnmente como STM32, dispone de 2 bus de comunicaciones I2C, SPI y 3 UART.

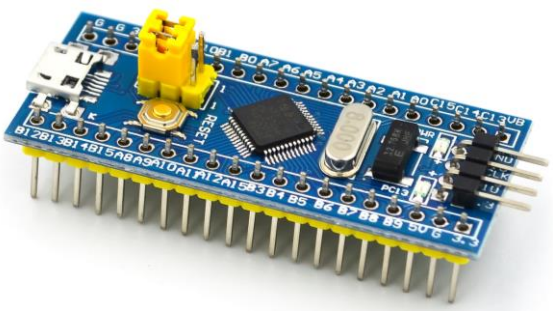


Ilustración 5: Microcontrolador BluePill. Fuente: [9].

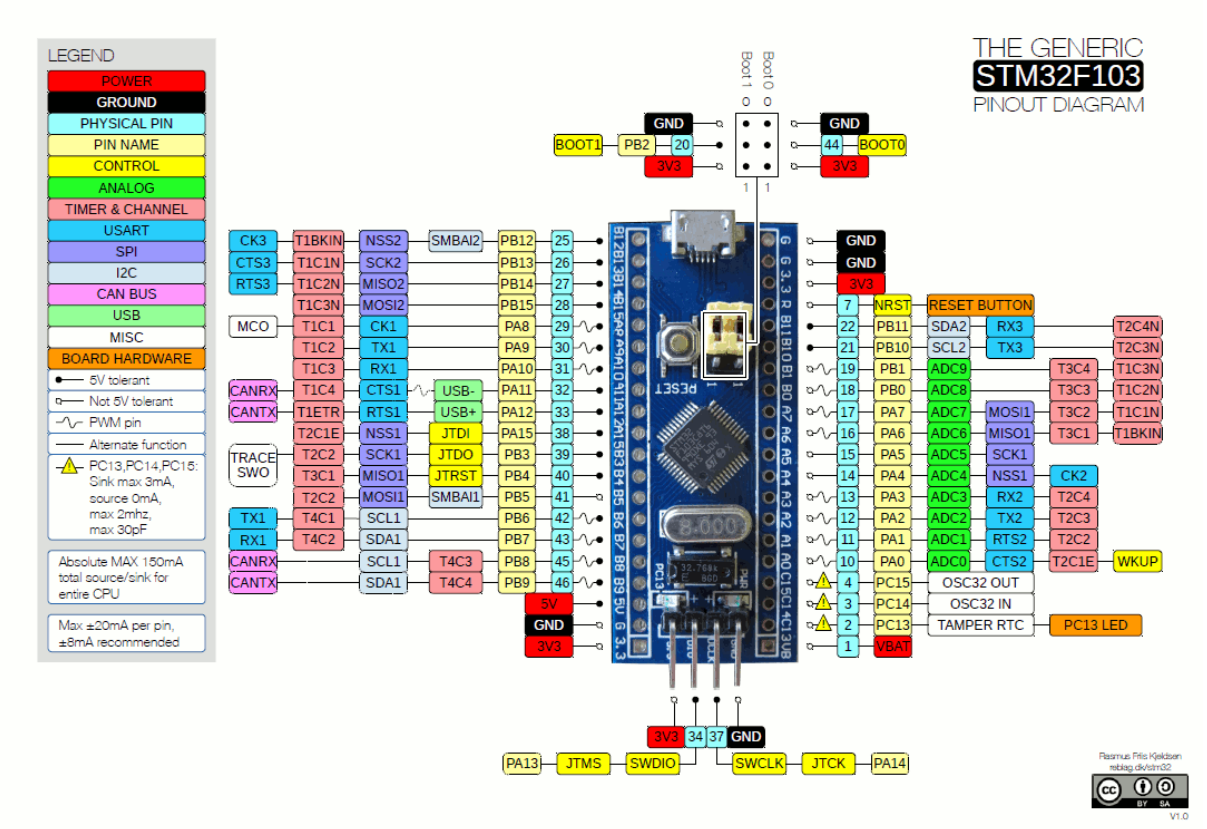


Ilustración 6: Pinout del microcontrolador BluePill. Fuente: [10].

El datasheet del microcontrolador BluePill se encuentra en [**11**] para su consulta.

3.2 Subsistema de actitud

3.2.1 Módulo GY-9250

El módulo GY-9250 es la unidad de medición inercial (*IMU*, del inglés *Inertial Measurement Unit)* y está basado en el sensor MPU-9250. Es el encargado de sensar los datos de actitud del satélite mediante tres funcionalidades incluidas en el mismo módulo: giroscopio, acelerómetro y magnetómetro de tres ejes, lo que permite un rastreo de movimiento con 9 grados de libertad (9 DoF).

En cuanto a la alimentación, el módulo dispone de un regulador de tensión a 3,3V para poder conectarse al pin de salida de 5V de los microcontroladores. Se comunica mediante I2C, pero hay librerías que permiten su uso mediante comunicación SPI.

La MPU-9250 tiene distintos rangos de trabajo para cada medida. Para el acelerómetro, se puede ajustar con rangos de entre ±2g, ±4g, ±8g, ±16g. El giroscopio

dispone de rangos de ±250Grad/Seg, ±500Grad/Seg, ±1000Grad/Seg y ±2000Grad/Seg.
Por último, el magnetómetro puede usarse con un rango de ±4800μT Para más información sobre el módulo, se puede consultar [**12**] para el datasheet del producto.

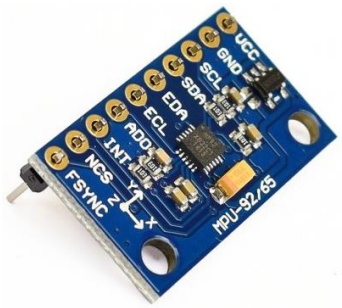


Ilustración 7: Módulo de medición universal modelo MPU-9250. Fuente: [13]

Es importante comentar que el módulo dispone de un DMP (*Digital Motion Processor*). Básicamente se trata de un procesador que permite fusionar los datos del acelerómetro y el giroscopio para liberar el microcontrolador de cálculos a la hora de determinar los ángulos de Euler.

En cuanto a la comunicación con el microcontrolador, el módulo GY-9250 es capaz, como ya se ha mencionado, de comunicar mediante I2C o SPI. Por razones de velocidad y consumo, se ha decidido usar el protocolo SPI. Para realizar la conexión, se conecta el pin MOSI (*Master Output Slave Input*) de la Bluepill al pin SDA de la IMU para la transmisión de datos. El pin MISO (*Master Input Slave Output*) al pin ADO de la IMU. SCK es la señal de reloj, NSS1 determina el dispositivo esclavo y lo habilita para la transmisión de datos y FSYNC (*Frame Synchronization*) es la entrada de sincronización de cuadro que, al no usarse, se debe conectar a masa. A continuación, se muestra un esquema junto con una tabla donde se resumen las conexiones mencionadas.

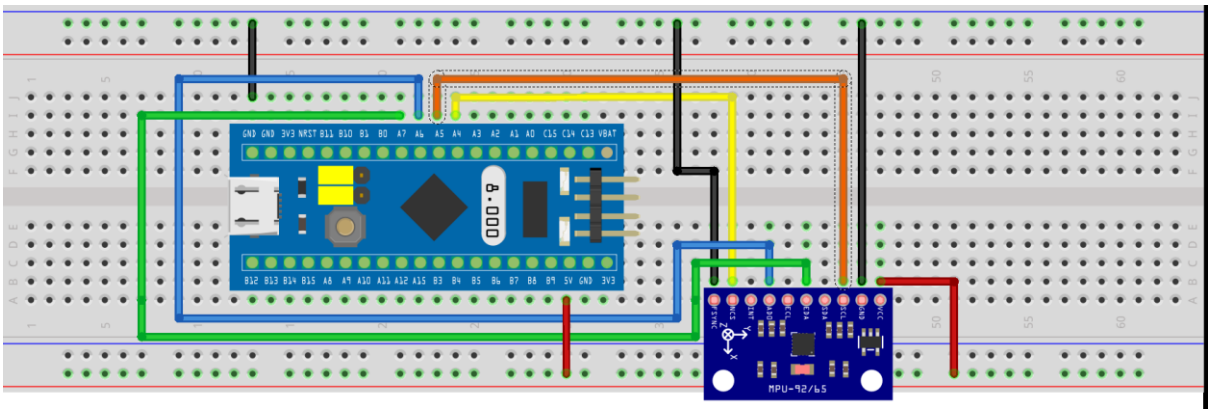


Ilustración 8: Esquema de conexionado de la IMU en protoboard.

En la imagen observamos un total de 7 conexiones. En color rojo se muestra la conexión de alimentación a 5V. Los dos cables en negro (GND y FSYNC) se conectan a tierra. En verde vemos la conexión de datos (MOSI), en azul el pin MISO a ADO, transmitiendo los datos provenientes del esclavo (IMU), en amarillo el pin selector de esclavo y en naranja la señal de reloj.

Señal	Pin	Nombre	IMU	Color
GND	GND	GND	GND	Negro
Vcc (5V)	5V	5V	5V	Rojo
MOSI	17	A7	SDA	Verde
MISO	16	A6	ADO	Azul
SCK1	15	A5	SCL	Naranja
NSS1	14	A4	NCS	Amarillo
FSYNC	GND	GND	GND	Negro

Tabla 1: Conexiones entre BluePill e IMU.

Una vez tenemos las conexiones realizadas, procedemos a configurar el software del módulo. Como se ha especificado anteriormente, la MPU-9250 es capaz de determinar valores de velocidad de giro en radianes por segundo, aceleración en m/s², campo magnético en micro teslas y temperatura en grados centígrados. Sin embargo, para determinar la posición del satélite, es necesario hacer uso de los ángulos de Euler. Los ángulos de Euler son un conjunto de tres coordenadas angulares que describen la orientación de un cuerpo en tres dimensiones respecto tres ejes

ortogonales. La rotación respecto el eje vertical del cuerpo describe la guiñada (*yaw* en inglés), el eje transversal el cabeceo (pitch) y el longitudinal el alabeo (roll).

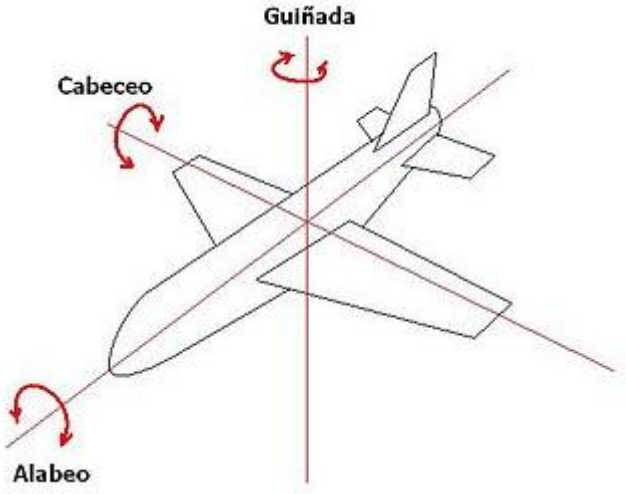


Ilustración 9: Ángulos de Euler. Fuente: [14]

Mediante estos tres ángulos obtenidos mediante el DMP podemos, de esta manera, determinar la orientación del satélite. El principal inconveniente es que las librerías disponibles actualmente en internet para extraer los datos del DMP están hechos para comunicar mediante I2C y el fabricante Invensense no tiene público el algoritmo para extraer estos datos. Esto nos obliga a usar filtros complementarios, los cuales filtran las señales registradas en bruto (*raw measurements* en inglés).

El filtro complementario actúa como un filtro de paso alto para los datos del giroscopio y como filtro de paso bajo para la señal del acelerómetro. El filtro complementario regula cualquier señal de ruido externa que pueda medirse erróneamente. Esto significa que si, por ejemplo, se mide una señal demasiado alta, se percibe como una perturbación y se filtra la señal de modo que devuelva un valor esperado. Dicho filtro se rige por la siguiente fórmula:

$$\theta = A \cdot (\theta_{prev} + \theta_{gyro}) + B \cdot \theta_{accel} \tag{3.1}$$

Donde A y B son dos valores que suman 1. Típicamente se escogen valores de 0,98 y 0,02, respectivamente. Por último, es importante mencionar qué es el gimbal lock o bloqueo de cardán. El bloqueo de cardán es la pérdida de un grado de libertad cuando dos de los ejes de un sistema de rotación se alinean de forma paralela.

En el programa MPU9250_SPI.ino(), creado originalmente por Andrés Gómez y Miguel Reurer, se extraen los ángulos de Euler de la IMU mediante el filtro complementario mencionado y se trata el Gimbal Lock. En el apéndice **A** se muestra el código completo explicado.

3.3 Subsistema de comunicación

El sistema de actitud es, como bien se ha explicado, el encargado del control del satélite, transmitiendo las órdenes necesarias a los actuadores para llevar el control de la posición del satélite respecto a la Tierra basándose en los sensores que conforman la placa. De todas formas, es muy importante establecer contacto directo con la placa de comunicaciones para recibir órdenes del centro de control o para devolver la información proveniente de los sensores. Para ello, el sistema de actitud debe ser capaz de intercambiar información con el OBC (*On Board Computer*), encargado de transmitir procesar y empaquetar toda la información que se envía o se recibe.

3.3.1 Planteamiento inicial

Inicialmente, se decidió simular dicho proceso de intercambio de comunicación entre placa ADCS y OBC mediante I2C usando la BluePill como controlador de actitud y un Arduino Nano como OBC, para así comunicar con la placa de comunicaciones de Juan Palomares.
En este sistema inicial, el Arduino Nano (como OBC) estaría conectado a un ordenador en el que se recibirían las órdenes tales como el modo de trabajo y el ángulo consigna mediante el puerto serie. Esta información sería transmitida al sistema de control del satélite (BluePill). La BluePill, calcularía las corrientes que debiera enviar a los magnetopares y devolvería dicha información junto con las medidas del sensor MPU-9250 al OBC. El OBC empaquetaría esta información y la enviaría mediante Bluetooth a un segundo ordenador, en el que se procesaría y se mostraría la información recibida.
Para la comunicación I2C entre Arduino Nano y BluePill se usaría el segundo bus I2C disponible en la BluePill a partir de la figura **6** para dedicar únicamente el primer bus I2C al control de los drivers y, de esta manera, desaturar de información y aprovechar todas las conexiones disponibles.
En cuanto a las conexiones de salida del Arduino Nano, se usarían los pines PC4 y PC5 para SDA y SCL, respectivamente, extraídas del pinout de la siguiente figura:

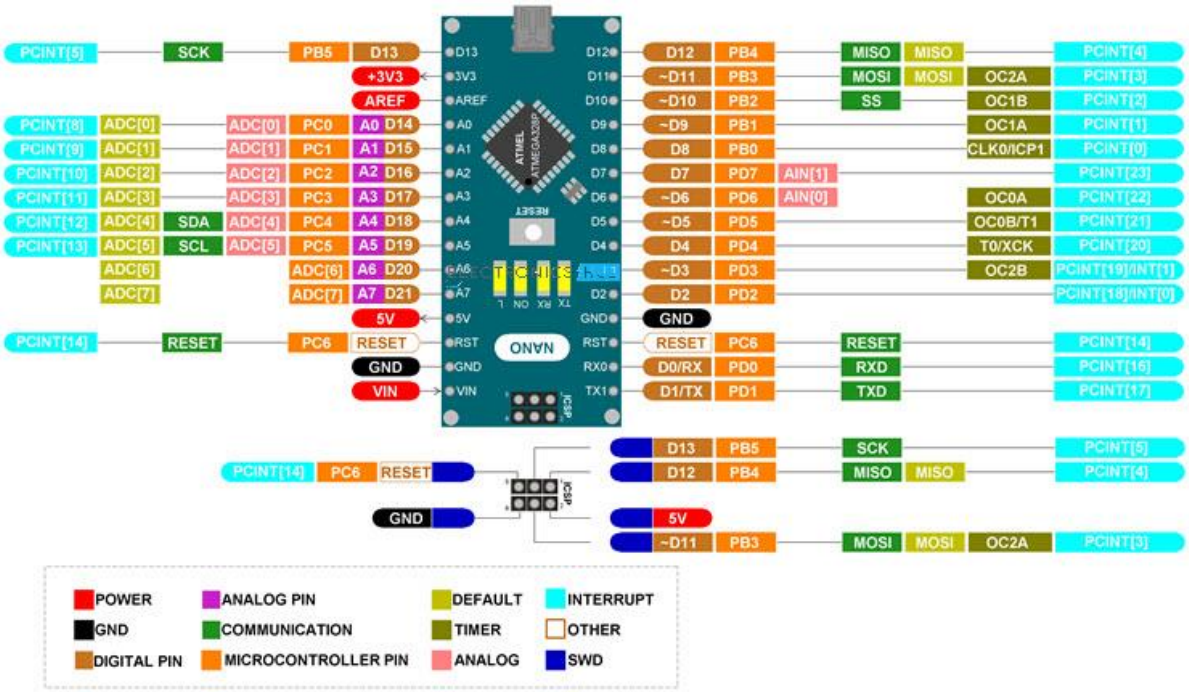


Ilustración 10: Pinout del Arduino Nano. Fuente: [15]

Como se observa, el Arduino Nano, escogido en el proyecto de Juan Palomares por razones de facilidad de programación y un tamaño ligeramente inferior con respecto la BluePill, no posee 2 bus I2C.

Para consultar el datasheet del Arduino Nano, el enlace está disponible en [**16**].

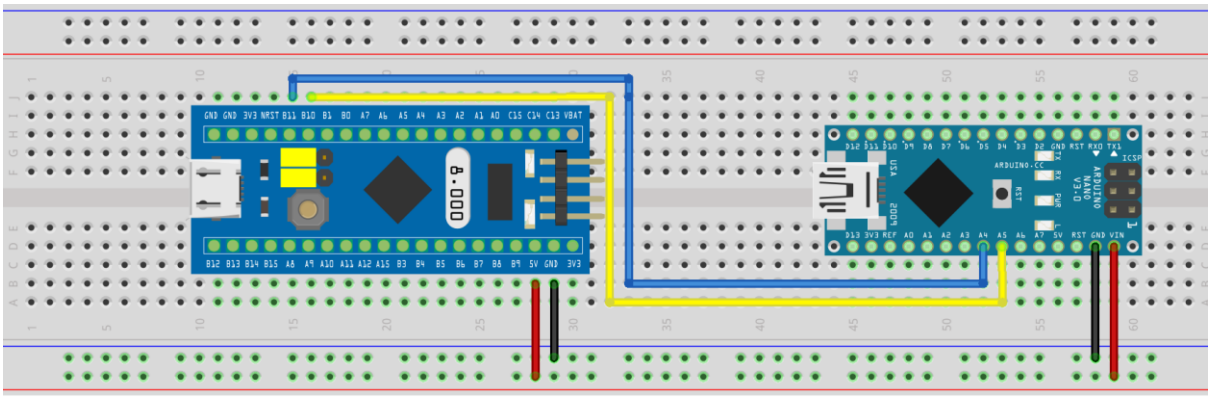


Ilustración 11: : Esquema de conexionado entre BluePill y Arduino Nano.

En la figura [**3.7**] se aprecian las conexiones entre el microcontrolador encargado del control del satélite (BluePill) y el OBC (Nano). Las conexiones de alimentación son los cables rojos para 5V y negro para GND. En cuanto al I2C, el cable azul se usa para la transmisión de datos SDA y el amarillo la señal de reloj o SCL. A continuación, se muestra una tabla-resumen de las conexiones necesarias.

BluePill			**Arduino Nano**			
Señal	**Pin**	**Nombre**	**Señal**	**Pin**	**Nombre**	**Color**
GND	GND	GND	GND	GND	GND	Negro
Vcc (5V)	5V	5V	5V	5V	5V	Rojo
SDA	22	PB11	SDA	PC4	18	Azul
SCL	21	PB10	SCL	PC5	19	Amarillo

Tabla 2: Conexiones entre BluePill y Arduino Nano.

En cuanto al software, para el procedimiento de transferencia de datos se plantearon 2 posibilidades. En la primera (y más intuitiva), se consideraba que el Ardunio Nano actuaría como maestro para solicitar datos a la BluePill y enviarlos al módulo Bluetooth, que actuaría como esclavo, para transferir los datos al segundo ordenador. El inconveniente que se encontró fue que, para ello, la BluePill debería actuar como esclavo, pero a la vez como maestro para dar órdenes a los drivers de corriente, por lo que debería ir cambiando el rol de maestro a esclavo sucesivamente y complicaría mucho su programación. La segunda opción (más sencilla) era que la BluePill actuara

como maestro y enviar información al Arduino Nano como esclavo. A continuación, se muestra el código de la BluePill como maestro para comunicar con el segundo bus I2C con el Arduino Nano explicado en diferentes imágenes:

```
52 //I2C2
53 TwoWire Wire2(2, I2C_FAST_MODE); //Se declara el segundo bus I2C (PB11, PB10)
```

```
60 void setup()
61 {
62
63   //Puertos Serie
64   Serial.begin(9600);   // inicializamos puerto serie
65   Wire.begin();         // inicializamos el bus i2c
66   Wire2.begin();        // inicializamos el 2 bus i2c
```

En estas dos primeras imágenes de código, se declara el bus I2C y se inicializa. A continuación, se declaran dos uniones, que permitirán tanto recibir como enviar la información en forma de array de bytes ya que el protocolo de comunicación I2C requiere este tipo de datos. Esta información luego deberá ser descodificada. Para esta parte, se ha extraído el código del proyecto de Juan Palomares “Sensores.ino”, incluido en el anexo **A**.

```
23 //Se crea una unión para pasar los datos "float" a "array de bytes" y así poderlos enviar por i2c
24 union {
25   byte array[4]; // Se declara un array de 4 bytes
26   float FloatNumber;
27 } FloatByteRX;
28
29 //Se crea una unión para pasar los datos provenientes del buffer de recepción ("array de bytes") a float
30 union {
31   byte array[4];
32   float FloatNumber;
33 } FloatByteTX;
34
35 byte bufferRX[60], bufferTX[60];
36 float DatosOtroSat[4]; //Se define un array con todos los datos que se obtendrán del otro satélite.
```

Observamos en el código anterior que se crean dos uniones: una será para el buffer de recepción (RX) y la otra para el de transmisión (TX).

A continuación, se muestra el subprograma de codificación y envío de datos (02_Send.ino) y el de recepción y decodificación de datos (01_Request.ino).

Función de recepción

```
void Request() {

  Wire2.requestFrom(8, 4);    //Se piden 4 bytes al Arduino esclavo con dirección 0x08
  int i = 0; //contador de cada byte que llega
  while (Wire2.available()) {//Este while acaba cuando se acaban los datos a recibir.
    bufferRX[i] = Wire2.read(); //Los bytes que se obtienen se guardan en el buffer de recepción
    i = i + 1;
  }
  //Decodificación de los datos recibidos
  for (int j = 0; j < 1; j++) { //Mediante este "for" se recibe 1 "float" que se espera obtener
    for (int i = 0; i < sizeof(float); i++) {  //sizeof(float) devuelve el tamaño de un float (4 bytes)
      FloatByteRX.array[i] = bufferRX[i + j * 4]; //Los 4 bytes que representan 1 float se guardan en la unión de recepción
    }
    DatosOtroSat[j] = FloatByteRX.FloatNumber; //Se convierten los 4 bytes a "float" y se guarda el valor
  }

  angulo = DatosOtroSat[0];

}
```

El subprograma “01_Request.ino” es el encargado de solicitar y recibir la información al Arduino esclavo, decodificarla y guardarla en variables conocidas. Se recibe un array de bytes codificado por el programa del esclavo y se decodifica para obtener los valores deseados sabiendo el formato en el que llega. En este caso se espera obtener solamente un dato que es el ángulo consigna tipo *float*. El tamaño de un dato de tipo *float* es de 4 bytes, por lo tanto, en la instrucción “*Wire2.requestFrom(8, 4)*” se solicitan 4 bytes de datos al esclavo con dirección 0x08. En la instrucción “*while (Wire2.avaiable())*” se reciben todos los datos que llegan, si es que los hay. La dirección 0x08 se establece en el programa del esclavo (Arduino Nano). Este programa se encuentra en el anexo de forma explicada en el programa “Ard_nano.ino”.

Función de transmisión

```
void Send() {

  int j = 0; //Se inicializa un contador para que el buffer no se vaya sobreescribiendo conforme se codifican nuevos valores
  //Codificación de la intensidad en x [j=0]
  FloatByteTX.FloatNumber = Ix; //El valor float de la unión de transmisión ahora será "Ix"
  for (int i = 0; i < sizeof(float); i++) { //Se guardan todos los bytes que codifican "Ix" en el buffer de transmisión
    bufferTX[j] = (FloatByteTX.array[i]); //Se escriben los bytes que codifican "Ix" en el buffer de transmisión
    j += 1; //El siguiente byte se guardará en la siguiente posición del "array" del "bufferTX"
  }

  //Codificación de la intensidad en y [j=1]
  FloatByteTX.FloatNumber = Iy;
  for (int i = 0; i < sizeof(float); i++) {
    bufferTX[j] = (FloatByteTX.array[i]);
    j += 1;
  }

  //Codificación de la intensidad del campo magnético [j=2]
  FloatByteTX.FloatNumber = HTx;
  for (int i = 0; i < sizeof(float); i++) {
    bufferTX[j] = (FloatByteTX.array[i]);
    j += 1;
  }

  //Transmisión de los datos al Arduino esclavo
  Wire2.beginTransmission (8); //Inicio de la transmisión con el Arduino esclavo con dirección 0x08
  for (int i = 0; i < j; i++) { //Este for recorre todo el buffer de transmisión desde la primera posición hasta la última con valor
    Wire2.write(bufferTX[i]); //Se envía el byte en cuestión
  }
  Wire2.endTransmission(); //Finalización de la transmisión

}
```

El subprograma “02_Send.ino” es el encargado de codificar la información que se quiere enviar al esclavo y enviarla. El proceso es inverso al del subprograma “01 _request.ino”. Se usa la unión de transmisión declarada al inicio del programa, guardando los valores de cada variable en el buffer de transmisión. Si observamos la imagen, vemos cuatro bloques de código bien diferenciados. En los tres primeros se codifican los valores de tres variables, correspondientes a las intensidades del driver y al campo magnético terrestre. En el último bloque se envía el array de bytes al esclavo con dirección 0x08 que, en este caso, tendrá 12 bytes correspondientes a tres variables de tipo *float.* Este programa no es el definitivo, por lo que el nombre de las variables que se usa ni la cantidad de información enviadas es la correspondiente a la versión final del programa.

3.3.2 Planteamiento final

En un sistema real, la placa de control debería comunicar con el ordenador de abordo y recibir las instrucciones desde este, pero para simplificar la simulación y tener una placa completamente independiente en la que se puedan hacer pruebas sin necesidad del OBC, lo que haría el sistema más cómodo, se decidió usar un módulo bluetooth que sirviera para recibir las instrucciones desde el ordenador. De esta forma, el propio controlador de actitud también se encarga de la recepción y envío de datos. Se reciben entonces las instrucciones a través de un ordenador mediante bluetooth y se devuelve la información del sistema de actitud mediante el mismo. Para ello, se usará el módulo bluetooth HC-05, similar al HC-06 usado en proyectos anteriores, pero con la capacidad de actuar tanto de esclavo como de maestro.

3.3.3 Módulo HC-05

El módulo HC-05, es un módulo Bluetooth que comunica mediante puerto serie y tiene un alcance de hasta 10 metros. Se puede usar para conectar 2 microcontroladores o incluso a un teléfono móvil o un ordenador. El módulo necesita una alimentación de entre 3.6V y 6V DC. El datasheet del componente se encuentra en [**17**].

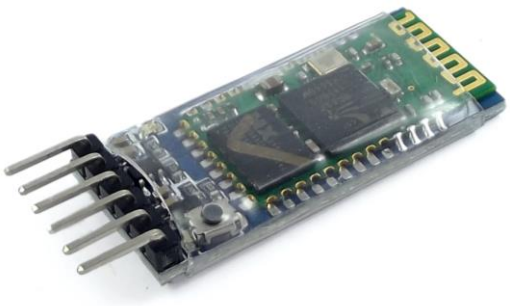


Ilustración 12: Módulo Bluetooth HC-05. Fuente: [18]

El HC-05 se conecta mediante UART (*Universal Asynchronous Receiver-Transmitter*) directamente a los pines seriales del microcontrolador. Una ventaja que posee la BluePill sobre otros microcontroladores es que los niveles altos de voltaje en las conexiones RX y TX son de 3,3V, por lo que no hará falta un divisor de voltaje como en otros microcontroladores en los que niveles altos de las conexiones UART son de 5V, lo cual quemaría el módulo.
A continuación, se muestra en la figura **13** la conexión completa del módulo Bluetooth en la protoboard.

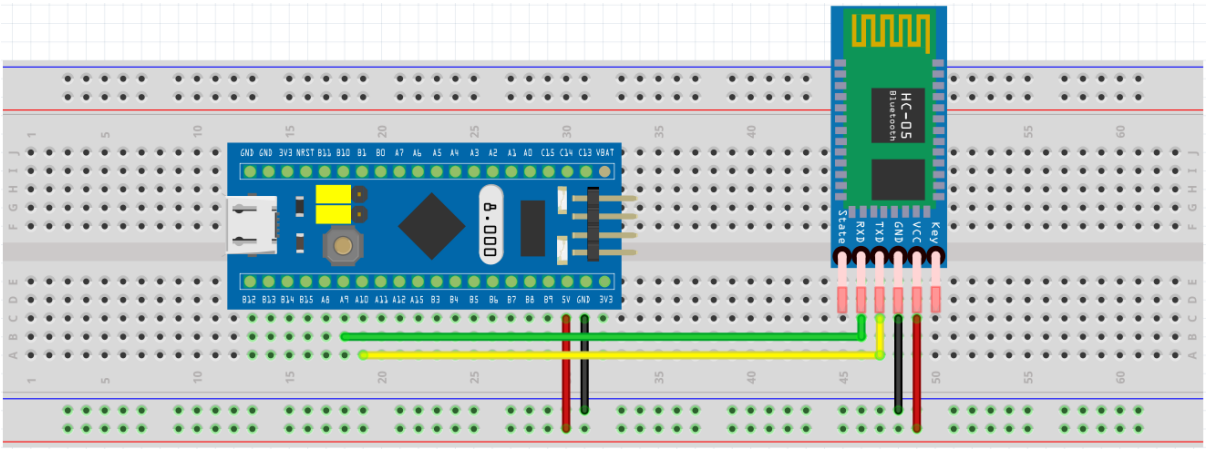


Ilustración 13: Esquema de conexionado del módulo Bluetooth HC-05 en protoboard.

Señal	Pin	Nombre	HC-05	Color
GND	GND	GND	GND	Negro
Vcc (5V)	5V	5V	5V	Rojo
RX	43	PA10	TX	Amarillo
TX	42	PA9	RX	Verde
Enable	-	-	Enable	-
Key	-	-	Key	-

Tabla 3: Conexiones entre BluePill y HC05.

En la tabla **3** se muestra un resumen de las conexiones entre el HC-05 y la BluePill. Si observamos bien, vemos que la conexión RX de un dispositivo se conecta con la TX del otro y viceversa. Esto se debe a que el canal emisor de uno debe ir al receptor del otro.

Para usar el módulo, primero se tuvo que configurar la velocidad de transmisión y cambiar el nombre. El HC-05 viene con una configuración predeterminada, que es la siguiente [**19**]:

Nombre: H-C-2010-06-01

Rol: Esclavo

Contraseña: 1234

Velocidad de transmisión: 9600 baudios

Una vez configurado, se estableció una velocidad de transmisión de 115200 baudios y se cambió el nombre, quedando así los parámetros del módulo una vez configurados:

Nombre: ADCS

Rol: Esclavo

Contraseña: 1234

Velocidad de transmisión: 115200 baudios

Para realizar la configuración, se tuvieron que introducir los comandos AT (abreviatura de "*Attention*"). Los comandos AT, introducidos mediante el puerto serie, "avisan" al módulo Bluetooth de que se le está dando una instrucción, y se introducen con el formato "AT+*Instrucción*" [**20**].

Para introducir los comandos al módulo, se usó el siguiente programa, extraído de Github [**21**].

```
1  //Función para la configuración del Bluetooth
2  //Autor: bitwiseAr
3
4  void setup() {
5  Serial.begin(9600);
6  Serial1.begin(38400); //Establece el puerto serie bluetooth a 38400 baud
7
8  }
9
10 void loop() {
11
12 if (Serial1.available())
13 {
14 Serial.write(Serial1.read()); //Lee puerto serie BT y envía a Arduino
15 }
16
17 if (Serial.available())
18 Serial1.write(Serial.read()); //Lee puerto serie Arduino y envía a BT
19
20 }
```

Observamos que se inicializan dos puertos serie, uno a 9600 baudios y el otro a 38400 baudios, que corresponde al puerto serie virtual del módulo bluetooth. Para

configurarlo es necesario establecer esta velocidad para que responda a los comandos AT. Otro paso necesario para poder configurarlo es presionar el botón que contiene los 5 primeros segundos de encendido, hasta que parpadee cada segundo. Esto activará el modo de configuración y significa que ya lo podemos configurar.

A continuación, se muestra en la imagen **14** los pasos seguidos para la configuración de HC-05 para usarlo tal y como se ha explicado:

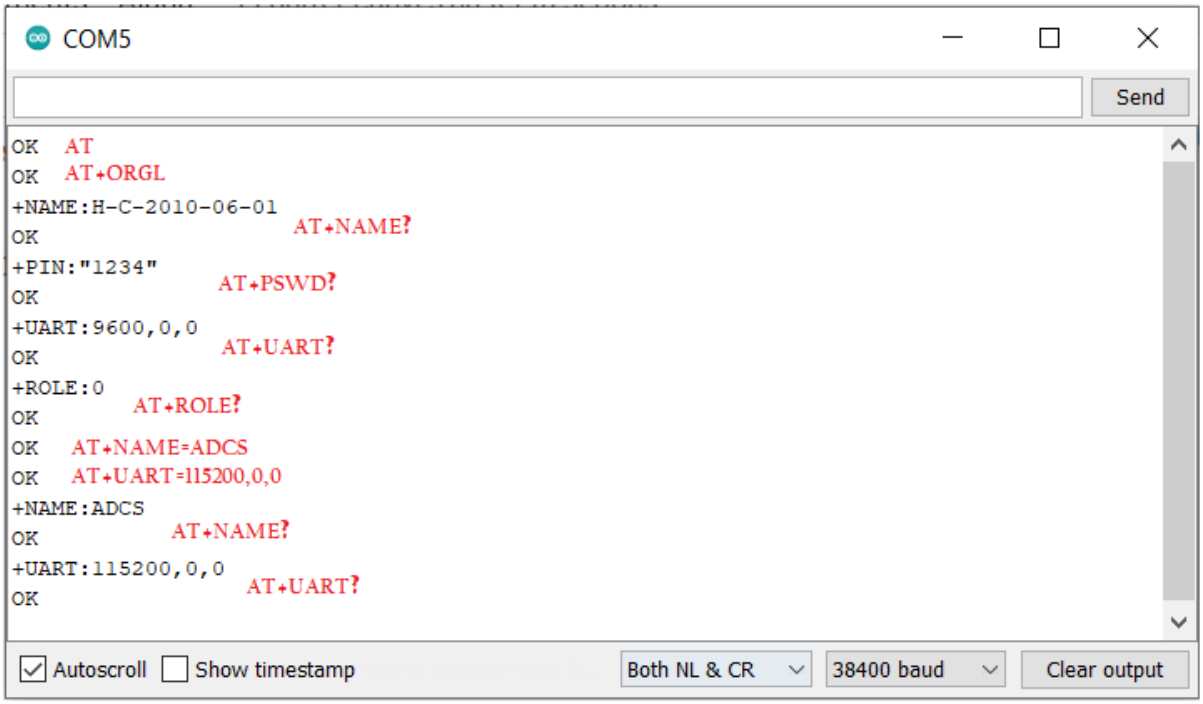


Ilustración 14: Configuración del módulo HC-05 mediante comandos AT.

Para saber qué función tiene cada comando, en la tabla **4** se muestra un resumen ordenado de los comandos usados:

Paso	Comando	Función
1	AT	Comprobar conexión con el módulo.
2	AT+ORGL	Establecer estado de fábrica.
3	AT+NAME?	Consulta nombre.
4	AT+PSWD?	Consulta contraseña.
5	AT+UART?	Consulta velocidad de conexión.
6	AT+ROLE?	Consulta rol (esclavo=0, maestro=1).
7	AT+NAME=ADCS	Establece nombre a “ADCS”.
8	AT+UART=115200,0,0	Establece velocidad de transmisión a 115200 baudios.
9	AT+NAME?	Consulta nombre.
10	AT+UART?	Consulta velocidad de transmisión.
11	AT+RESET	Cambiar de modo configuración a modo usuario

Tabla 4: Comandos usados para la configuración del módulo HC-05.

Para el envío y recepción de datos, se usa un subprograma llamado “bluetooth()”. Este subprograma ordena la información a enviar en un formato entendible para la interfaz en Visual Studio y decodifica la información recibida en un formato conocido.

```
void bluetooth() { //Este subprograma envia y recibe los datos del puerto serie bluetooth

  if (Serial1.available() > 0) //Mientras haya datos disponibles en el puerto serie
  {
    String datos_serie = Serial1.readString(); // Se lee un string donde los datos vienen con el formato dato1;dato2;...;datoN
    String modo_st = getValue(datos_serie, ';', 0); //Se guardan los datos en las variables de tipo String
    String angulo_st = getValue (datos_serie, ';', 1); //ídem
    modo = modo_st.toFloat(); //Se pasa el valor de String a Float
    angulo_init = angulo_init_st.toFloat(); //ídem
  }
```

En la imagen anterior se muestra el código en el que se decodifica la información proveniente de la interfaz. Se reciben, concretamente, dos valores que corresponden al ángulo consigna y al modo de trabajo. Para ello, se hace uso de otro subprograma “string getValue(datos, separador, índice)”, extraído de [**22**], en el que se introducen como argumentos la trama de datos recibida, el símbolo que separa un dato de otro (en este caso se escogió punto y coma “;”) y el índice de la variable y se devuelven los datos separados en variables conocidas.

En el programa de la interfaz, explicado en el capítulo (**6**), se explica que el formato enviado a la BluePill es “modo;ángulo”, por lo que el índice de la variable modo es 0 y el del ángulo es 1. En el anexo **A** se muestra el subprograma “04_getValue.ino”.

En cuanto al envío de datos, lo que se hace es juntar todas las variables que se quieren enviar a la interfaz en un String con los datos separados por “;”. Para diferenciar cada envío de datos, el String empieza con el símbolo de dólar “$”, por lo que el programa de la interfaz en Visual Studio, entiende que llega una trama nueva cada vez que ve el símbolo del dólar. Así pues, la estructura que sigue el envío de datos es “$dato1;dato2;dato3;dato4…;datoN;”, como se observa en la imagen siguiente extraída del mismo subprograma bluetooth():

```
24  String s = '$' + String(Vx) + ';' + String(Vy) + ';' + String(Vz) + ';' + String(Ax)
25  Serial.println(s);
26  delay(500);
```

La trama de datos es más larga que lo que se muestra en la imagen, pero es suficiente para entender su funcionamiento. El código completo, como ya se ha indicado, se encuentra en el anexo **A**. En la línea 26 añadimos un delay de medio segundo para dar tiempo al buffer de transmisión de incluir todos los datos.

3.4 Actuadores

3.4.1 Módulo ROB-15451

El módulo ROB-15451 de SparkFun Electronics es un SCMD (*Serial Controlled Motor Driver*) controlado por un PSoC (*Programmable System on Chip*) modelo CY8C4245AXI-483 de la marca Cypress Semiconductors . Se trata de un driver de corriente diseñado para comunicar por I2C (aunque también hay la posibilidad de usar UART) capaz de controlar hasta 2 motores DC de forma independiente ya que dispone de 2 puertos de salida.
El driver requiere una alimentación de entrada con una tensión comprendida entre 3V y 11V con máximo absoluto de 12V. Para alimentar el circuito lógico del driver, se precisa de una tensión de 3,3V.
En cuanto al control de los drivers, estos poseen unos puertos Qwiic, que tienen entrada para conectores JST de 4 pines polarizados, de manera que siempre se conectarán correctamente. Estos conectores poseen la principal ventaja de que conectándolos con el conector Qwiic no hará falta soldar los pines SDA y SCL y además permite crear una *Daisy Chain*, por lo que se podrán conectar entre sí más de un driver a la vez, únicamente asignando una dirección física diferente de I2C a cada driver. Para hacer esto, habrá que soldar los pines de la parte inferior del ROB-15451.

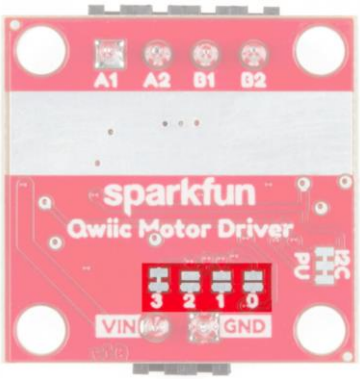


Ilustración 15: Parte inferior del driver ROB-15451. Fuente: [23]

En la imagen **15** podemos observar los pines que se usan para asignar las direcciones a cada driver, es importante saber que la dirección por defecto es la 0x5D, en la que viene soldado el primer jumper.
A continuación, se muestra una tabla en la que se resumen las diferentes direcciones disponibles y sus funcionalidades.

Pattern	Mode	User Port	User Address	Expansion Port
0000	UART at 9600	UART	N/A	Master
0011	I²C	I²C	0x58	Master
0100	I²C	I²C	0x59	Master
0101	I²C	I²C	0x5A	Master
0110	I²C	I²C	0x5B	Master
0111	I²C	I²C	0x5C	Master
1000	**I²C**	**I²C**	**0x5D**	**Master**
1001	I²C	I²C	0x5E	Master
1010	I²C	I²C	0x5F	Master
1011	I²C	I²C	0x60	Master
1100	I²C	I²C	0x61	Master
1101	UART at 57600	UART	N/A	Master
1110	UART at 115200	UART	N/A	Master
1111	N/A	Reserved	N/A	N/A

Tabla 5: Patrones para asignación de dirección de los drivers. Fuente: [23]

Como podemos observar en la tabla **5**, las direcciones 0000, 1101, 1110 están reservadas a conexiones UART de 9600, 57600 y 115200 baudios respectivamente, mientras que las otras direcciones mostradas se pueden usar para conectar mediante I2C. Como ya se ha mencionado, la dirección por defecto (marcada en negrita en la tabla) es la $(0x5D)_{16}$ para el usuario y el patrón $(1000)_2$ para asignar los jumpers.

El funcionamiento del driver de corriente ROB-15451 se basa en un puente H. El puente H se utiliza para cambiar el sentido de giro de un motor DC, pudiendo así acelerar o desacelerarlo de forma trivial. En este proyecto en lugar de un motor se conectarán los magnetopares y la rueda de inercia. Esto hará que, conmutando los transistores del circuito de cierta manera, explicada a continuación, podemos controlar el sentido de la intensidad que pasa por la bobina y controlar el dipolo magnético generado. A continuación, la descripción del funcionamiento del puente H.

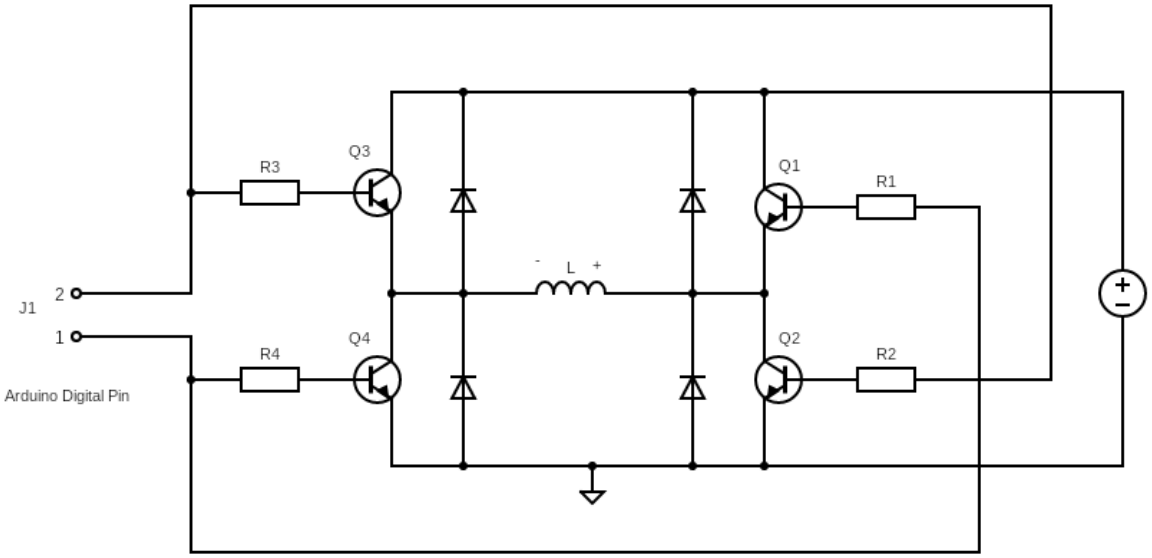


Ilustración 16: : Esquema general del puente en H.

El circuito debe su nombre a la forma característica en la que se disponen los transistores formando la letra H.

Como se observa en la figura **16** el funcionamiento se basa en cuatro transistores NPN que actúan como interruptores. Para dominar el sentido de la intensidad que pase por el magnetopar se deben conectar los transistores en cruz, porque si los conectamos de un mismo lado, se crea un cortocircuito. Por esta razón se colocan los 4 diodos que observamos en paralelo a los transistores. Estos protegen de la corriente de retorno que puede causarse por un posible mal uso del driver o la propia corriente de descarga que pueda crear la bobina al desconectarla.

Para explicar el proceso se muestra como ejemplo el funcionamiento del driver L298n, que se basa en el mismo proceso, pero el control lógico se lleva a cabo mediante conexiones en los pines digitales del microcontrolador, mientras que en el ROB-15451 se controla mediante software y el proceso no es, quizás, tan visual a la hora de explicar el funcionamiento. Las principales diferencias entre el driver L298n y el ROB-15451 son que el primero soporta tensiones más altas, de hasta 35V, lo cual conlleva un tamaño significativamente más grande respecto el ROB-15451 y las conexiones de entrada que controlan la lógica del circuito. En el L298n, la lógica se conecta físicamente conectando los pines de habilitación a los pines digitales del Arduino, mientras que en el ROB-15451, se establece una comunicación por I2C con el Arduino y se dirige la dirección mediante comandos, explicados más adelante.

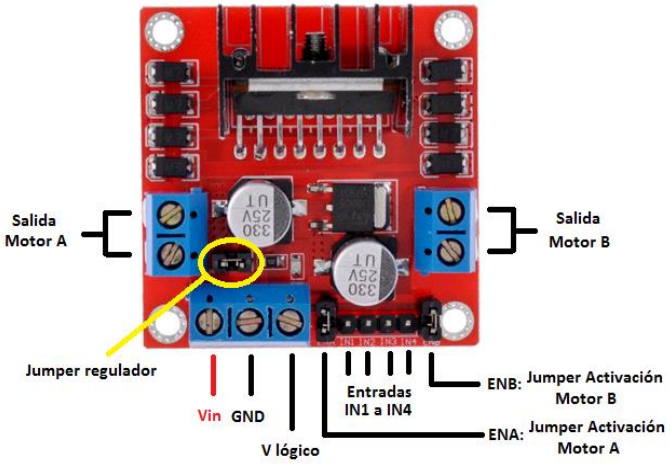


Ilustración 17: Vista superior del driver ROB-15451. Fuente: [23]

Dicho esto, en la siguiente explicación se referirá como J1 a los pines que controlan el driver (marcados como Entradas IN1 a IN4 en la imagen **17**). La conexión J1 está dividida en 2 pines J2.1, conectada a los transistores Q3 y Q2 y J2.2 a Q1 y Q4 (observable en la imagen **16**), es decir, en cruz tal y como se ha especificado anteriormente. A continuación, se explicará el proceso de control del driver y los diferentes modos de trabajo posibles.

- J1.2=1

Cuando el pin J1.2 está conectado a la alimentación de la lógica, se cierra el circuito de forma que pasará corriente por la base de Q3 y Q2, permitiendo así que circule corriente de colector a emisor desde la fuente de alimentación. En este caso, vemos como el corriente primero recorrería Q3 y entraría por el negativo de la inductancia, yendo hacia Q2 y finalmente a tierra.
En este caso el vector de campo magnético sería hacia fuera de la hoja "⊙" y el vector dipolo magnético en el sentido de la corriente y perpendicular a la superficie lateral de la bobina

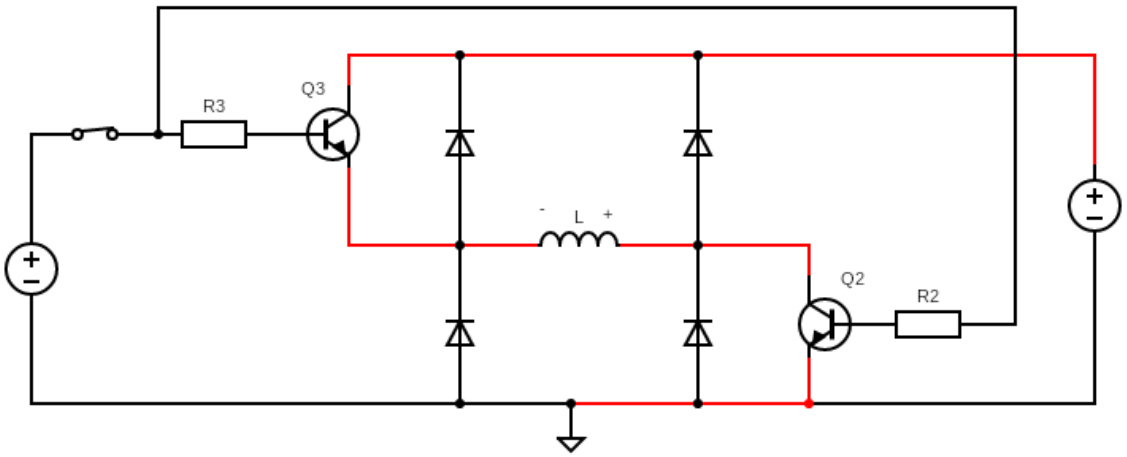


Ilustración 18: El corriente del magnetorquer circula de negativo a positivo.

- J1.1=1

Cuando J1.1 está activo se cierra el circuito de forma inversa a la explicación anterior. En esta ocasión circulará corriente por la base de los transistores Q1 y Q4, permitiendo así que pase corriente de colector a emisor. En este caso, vemos como el corriente primero pasaría por Q4 y por el positivo de la inductancia, yendo hacia Q1 y finalmente a tierra.
El vector de campo magnético sería hacia dentro de la hoja "⊗" y el vector dipolo magnético en el sentido de la corriente (positivo hacia negativo).

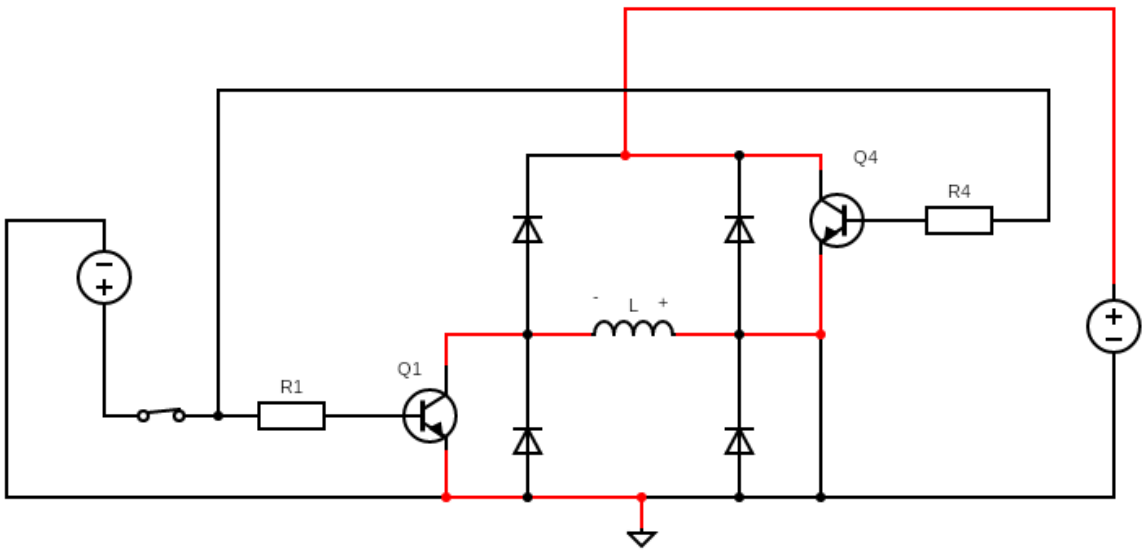


Ilustración 19: El corriente del magnetorquer circuila de positivo a negativo.

$B \rightarrow$	J1.1	J1.2	Q1	Q2	Q3	Q4
$\otimes$	1	0	1	0	0	1
$\odot$	0	1	0	1	1	0

Tabla 6: Conexiones para controlar ambos sentidos de corriente.

A continuación, se muestra el código de Arduino explicado en el que se configuran los drivers y se lleva a cabo el control.

```
1  #include <Wire.h>
2  #include <SCMD.h>
3  #include <SCMD_config.h>
4
5
6  SCMD MT; //creamos un objeto Magnetorquer que controla los 2 MT
7  void setup() {
8    Serial.begin(9600);  // inicializamos puerto serie
9    Wire.begin();        // inicializamos el bus i2c
10
11   //  Configuracion magnetorquers
12   MT.settings.commInterface = I2C_MODE;
13   MT.settings.I2CAddress = 0x5D; // Configuramos patrón "1000" en el driver para dirección 0x5D
14
15   delay(500);
16
17   // Inicializamos driver y habilitamos los outputs
18   Serial.print("Magnetorquer driver reading ID = 0x");
19   Serial.println(MT.begin(), HEX);
20
21   delay(500);
22
23   while ( MT.begin() != 0xA9 ) // Si no devuelve la dirección 0xA9 la comunicación no ha sido correcta
24   {
25     Serial.println( "ID Mismatch in magnetorquer driver. Trying again..." );
26     delay(5000);
27   }
28
29   Serial.println( "ID matches 0xA9" );
30
31   Serial.println("Waiting for magnetorquer configuration to complete...");
32   while (MT.ready() == false ); //
33   Serial.println("Magnetorquer configuration done.");
34
35   while ( MT.busy() );
36   MT.enable(); //Habilitamos el objeto MT
37
38 }
39
40 void loop() {
41
42  MT.setDrive( 0,0, 255); //(Objeto, Sentido, Velocidad)
43
44 }
```

En el anexo **A** se pueden consultar el código completo y librerías usadas (Driver_Hbridge.ino).

Del código mostrado se debe destacar la línea 42 en la que se lleva a cabo la instrucción de control del driver. Podemos ver que la estructura de la instrucción es (Objeto, Sentido, Velocidad). Objeto se refiere al número correspondiente a cada objeto de tipo MT, en el caso de tener un segundo motor o magnetopar, introduciremos un 1. Sentido es la instrucción que controla la lógica del circuito explicada anteriormente. Puede tomar valores 0 o 1 (uno para cada sentido). El parámetro Velocidad realiza el control de la intensidad suministrada al motor. Puede tomar valores de 0 a 255, correspondientes a los niveles de fuerza que puede entregar

el driver al motor, siendo 0 el mínimo y 255 el máximo. Esto se consigue mediante señales PWM (*Pulse-Width Modulation*).

Por último, las conexiones necesarias para conectar el driver de corriente son las que ya se han explicado. Se necesita alimentar el driver con 5V y GND. Para la comunicación con el microcontrolador, se conectan los cables de la señal de reloj (SCL) y el bus de datos (SDA) a los pines B6 y B7 de la BluePill, respectivamente. Se alimenta el circuito lógico del driver con 3,3V con la salida 3V3 de la BluePill y GND. Finalmente, en las conexiones de salida conectaremos el magnetorquer.

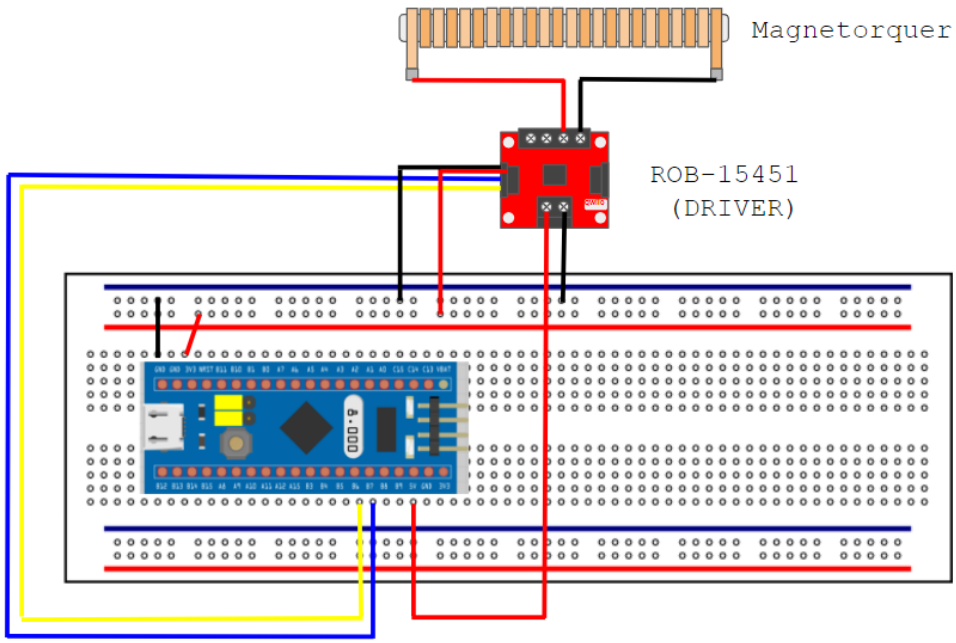


Ilustración 20: Esquema de conexionado del driver ROB-15451 en protoboard.

Señal	Pin	Nombre	ROB-15451	Color
GND	GND	GND	GND	Negro
Vcc (3,3V)	3V3	3V3	3V3	Rojo
SDA	43	B7	SDA	Azul
SCL	42	B6	SCL	Amarillo
Vin (5V)	5V	5V	5V	Rojo

Tabla 7: Conexiones entre ROB-15451 y BluePill.

3.5 Sistema inicial

El diseño explicado en el apartado (**3.3.1**) se resume en el bloque de la siguiente figura **21**, en el que los módulos están marcados en negro, los subsistemas en azul y las conexiones entre cada módulo en rojo.

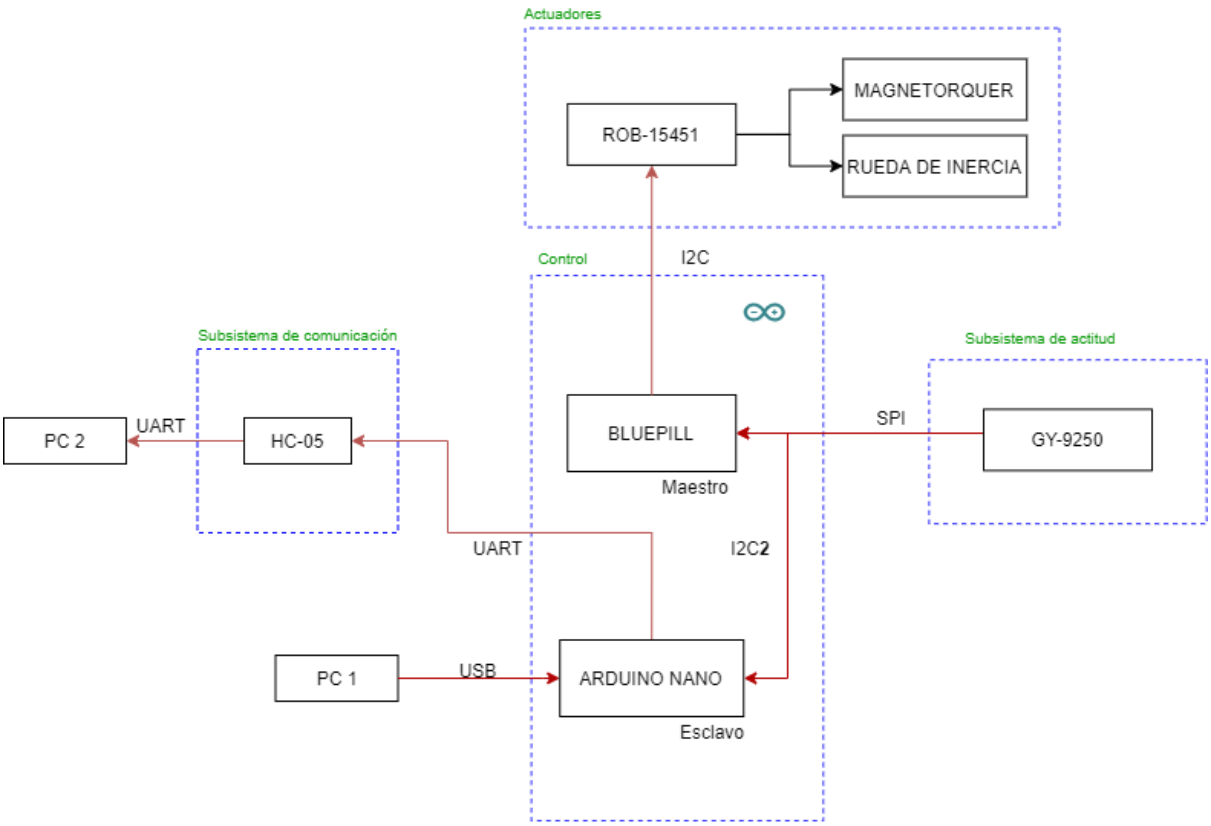


Ilustración 21: Diagrama de conexiones propuesto inicialmente.

3.6 Sistema final

El diseño explicado en el apartado (**3.3.2**) se resume en el bloque de la siguiente figura **22**.

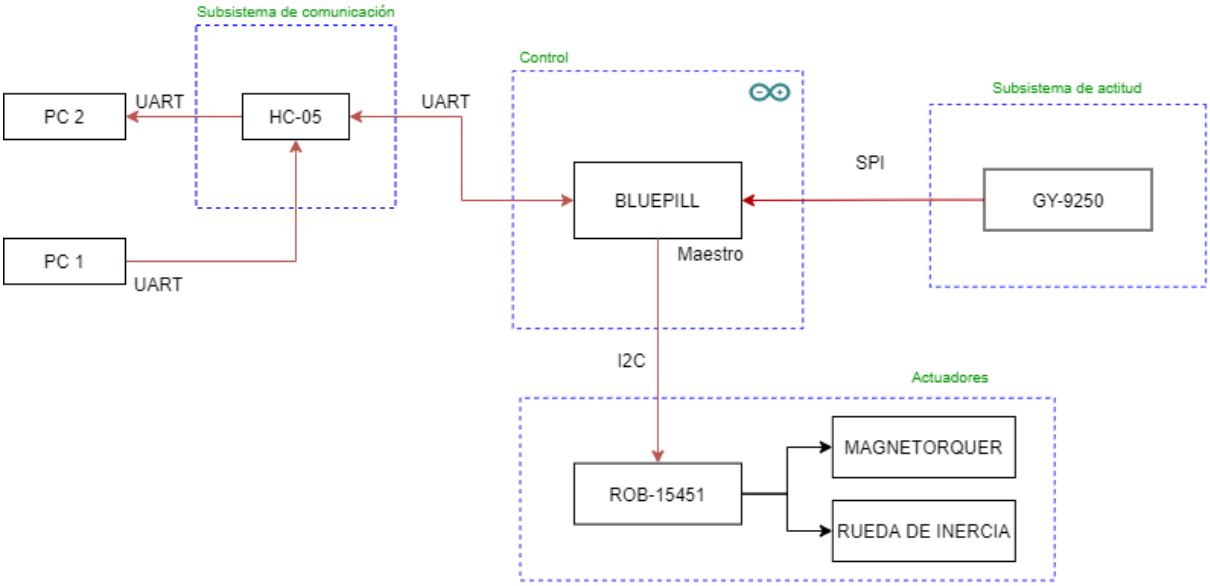


Ilustración 22: Diagrama de conexiones del sistema final.

CAPÍTULO 4: DISEÑO DE LA PCB

El objetivo principal de este capítulo es realizar el diseño de la placa PCB (Printed Circuit Board) a partir del conexionado previamente probado en la protoboard. Para ello, se usará el programa de diseño KiCad, ya usado en proyectos anteriores a este.

La placa tendrá como objetivo unir todos los elementos necesarios para realizar el control de actitud del CubeSat y para ello deberemos incluir los siguientes elementos: Bluepill, IMU, 2 drivers de corriente ROB-15451, 2 magnetorquers, un motor, el módulo bluetooth HC-05 y la rueda de inercia.
Deberemos incluir también un pin socket para el bus de potencia y otro de comunicación siguiendo la estructura de otras placas que hay en el laboratorio TIEG.

Además de estos componentes, se tendrán que usar elementos pasivos en el circuito. Uno de ellos son los condensadores de desacoplo. Los condensadores de desacoplo se usan como reservas de energía que se oponen ante cambios inesperados de voltaje o también para absorber cualquier pico de corriente que pueda haber. Para ello, se escogen dos tipos de condensadores que se conectarán en paralelo a los pines de alimentación de cada componente de la placa que lo requiera. Uno de ellos se coloca para absorber altas frecuencias de ruido y de respuesta rápida y otro para frecuencias bajas y con más capacidad. Para altas frecuencias se usa un condensador SMD de 100 nF. Junto a este se conecta un condensador cerámico de 1 μF.

Por último, se considera relevante tener la posibilidad de medir la corriente de salida de cada driver para ver qué intensidad de corriente se suministra a los magnetorquers y la rueda de inercia. Para conseguir esto, se tendrán que colocar pines de test en la salida de cada driver, de la siguiente forma:

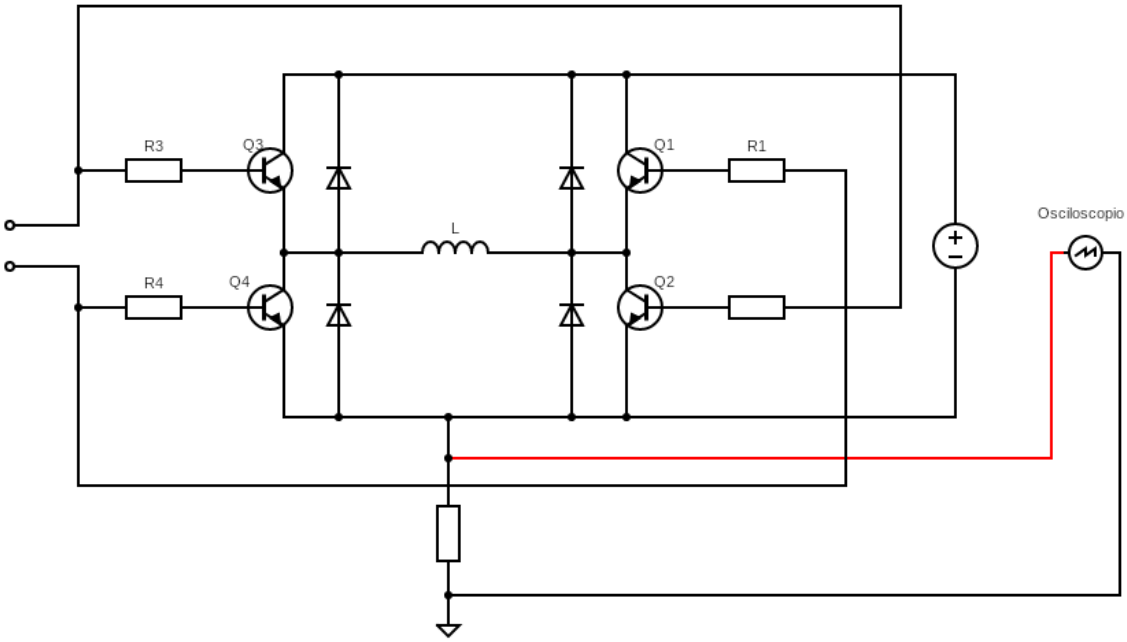


Ilustración 23: Puente H con resistencia para medir corriente de salida.

Observamos una resistencia en la salida del puente. Esta nos permitirá medir la corriente de salida de cada driver para ver, por ejemplo, el corriente máximo suministrado a las bobinas o a la rueda de inercia.

4.1 Esquemático

El esquemático es el primer paso para el diseño de la PCB. En él, se definen todas las conexiones entre componentes de la placa mediante símbolos. Se realiza un esquema eléctrico para que, posteriormente, se distribuyan los componentes en la placa.

4.1.1 Símbolos

Como se ha mencionado anteriormente, el primer paso para el diseño de la PCB es realizar el esquema eléctrico del sistema mediante símbolos. El software de KiCad dispone de una amplia librería de símbolos ordenados por su función, como por ejemplo filtros, amplificadores, relés, reguladores, componentes básicos, etc. Sin embargo, es prácticamente imposible que en la librería se incluyan todos los componentes electrónicos existentes, por lo que KiCad dispone de la opción de añadir o incluso crear tus propios. Para este diseño, la librería por defecto no dispone de

símbolos para la BluePill ni para la MPU-9250. Para la BluePill, se incluye una librería que contiene tanto el símbolo de esquemático, como el footprint y el modelo 3D, descargada de GitHub [**24**].
En cuanto a la IMU, se diseña el símbolo mediante el editor de librerías de KiCad. A continuación, se muestra el diseño creado:

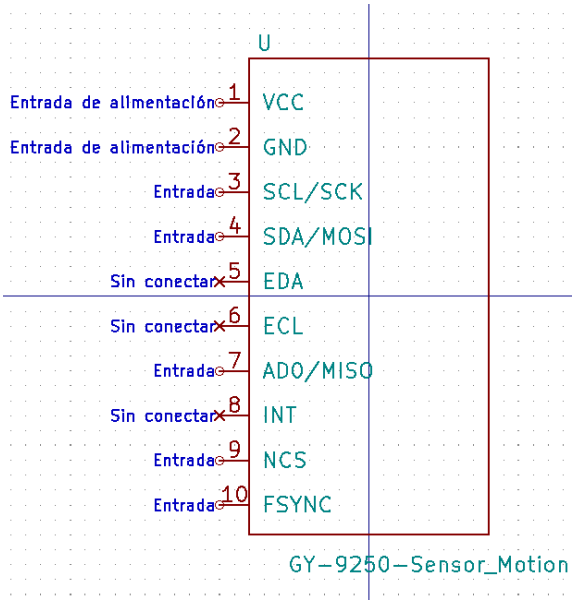


Ilustración 24: Símbolo del módulo GY-9250 creado en KiCad.

Observamos todas las salidas y entradas del sensor. La IMU dispone de diez pines de conexión, de los cuales se usan siete. De todas formas, se decidió crearlos todos, ya que no afecta al resultado final. El símbolo es de tipo “U” ya que forma parte de la familia de circuitos integrados.

En cuanto al módulo Bluetooth, se decide usar el símbolo de un conector vertical de 1x06 pines, ya que no se considera necesario incluir un símbolo específico. A continuación, se muestra el símbolo usado.

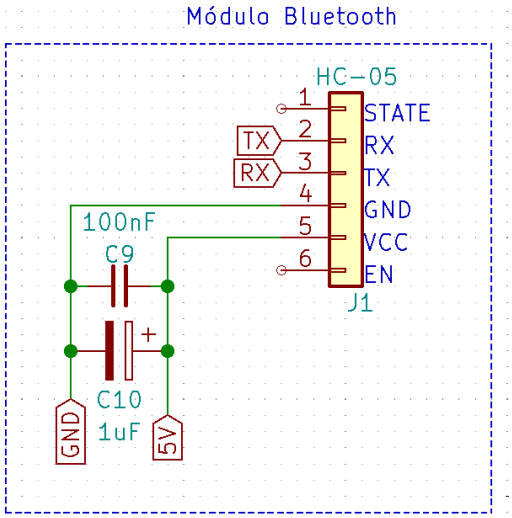


Ilustración 25: Conexión del módulo Bluetooth en el esquema de KiCad.

Para los drivers en KiCad no se dispone de símbolos ni huellas ya que el modelo ROB-15451 usa conectores Qwiic. Este conector sirve tanto como para realizar la conexión de I2C entre la Bluepill y el driver como para conectar más drivers en cadena. Dicho esto, para el diseño de la conexión de los drivers a la placa, se usa un conector hembra de 1x04 de la librería de KiCad. Para los conectores de potencia de los drivers, tanto de salida como de entrada, se usan conectores con un espaciado entre pines de 3.6mm de la librería de KiCad.

Finalmente, se usan dos símbolos también ya incluidos en la librería de KiCad para los buses de comunicación y alimentación. Se escogen, teniendo en cuenta el diseño de las placas anteriores como ya se ha mencionado, conectores hembra de 6x02 y 3x02 respectivamente.

4.1.2 Conexionado

En este proceso se conectan los símbolos incluidos en el apartado anterior. Básicamente, se trata de esquematizar las conexiones ya probadas en el prototipo del sistema del apartado (**3.5**).

Es importante destacar que en este paso se incluyen los elementos discretos que no se consideran en el sistema del prototipo. En el bus de alimentación, se conectará el pin de 5V con el plano de alimentación y los pines GND con el plano de masa. Los pines de 15, 9.6 y 3.3 voltios no se conectan a ningún elemento del sistema de control y servirán únicamente como puente entre placas. En el apéndice **C** se muestran las conexiones detalladamente.

4.2 Enlace de huellas

El siguiente paso a seguir para el diseño de la PCB, será asignar las huellas a cada símbolo del esquema. Las huellas son las representaciones de los elementos del conexionado, al igual que los símbolos. La diferencia es que las huellas tienen en cuenta las dimensiones que ocupan en la placa, para poder distribuir las conexiones del paso anterior. A continuación, se muestra la asignación de huellas a cada componente:

```
BluePill1 - BluePill_STM32F103C : BluePill_breakouts:BluePill_STM32F103C
      C1 -              1uF : Capacitor_THT:C_Disc_D3.0mm_W1.6mm_P2.50mm
      C2 -            100nF : Capacitor_Tantalum_SMD:CP_EIA-2012-12_Kemet-R
      C3 -            100nF : Capacitor_Tantalum_SMD:CP_EIA-3528-15_AVX-H
      C4 -              1uF : Capacitor_THT:C_Disc_D3.0mm_W1.6mm_P2.50mm
      C5 -            100nF : Capacitor_Tantalum_SMD:CP_EIA-2012-12_Kemet-R
      C6 -              1uF : Capacitor_THT:C_Disc_D3.0mm_W1.6mm_P2.50mm
      C7 -              1uF : Capacitor_THT:C_Disc_D3.0mm_W1.6mm_P2.50mm
      C8 -            100nF : Capacitor_Tantalum_SMD:CP_EIA-2012-12_Kemet-R
      C9 -            100nF : Capacitor_Tantalum_SMD:CP_EIA-2012-12_Kemet-R
     C10 -              1uF : Capacitor_THT:C_Disc_D3.0mm_W1.6mm_P2.50mm
  Comm.1 -            Comm. : Connector_PinSocket_2.54mm:PinSocket_2x06_P2.54mm_Vertical
      J1 -            HC-05 : Connector_PinSocket_2.54mm:PinSocket_1x06_P2.54mm_Vertical
      J2 -             Vout : Connector_Wire:SolderWire-0.1sqmm_1x04_P3.6mm_D0.4mm_OD1mm
      J3 -             MT X : Connector_PinHeader_2.54mm:PinHeader_1x02_P2.54mm_Vertical
      J4 -             Vout : Connector_Wire:SolderWire-0.1sqmm_1x04_P3.6mm_D0.4mm_OD1mm
      J5 -            Vin 1 : Connector_Wire:SolderWire-0.5sqmm_1x02_P4.6mm_D0.9mm_OD2.1mm
      J6 -             MT Y : Connector_PinSocket_2.54mm:PinSocket_1x02_P2.54mm_Vertical
      J7 -            Vin 2 : Connector_Wire:SolderWire-0.5sqmm_1x02_P4.6mm_D0.9mm_OD2.1mm
      J8 -           Vout 1 : Connector_PinSocket_2.54mm:PinSocket_1x04_P2.54mm_Vertical
  Qwiic1 -            Qwiic : Connector_PinSocket_2.54mm:PinSocket_1x04_P2.54mm_Vertical
      R1 -            0 Ohm : Resistor_SMD:R_1206_3216Metric_Pad1.30x1.75mm_HandSolder
      R2 -            0 Ohm : Resistor_SMD:R_1206_3216Metric_Pad1.30x1.75mm_HandSolder
     TP1 -        TestPoint : TestPoint:TestPoint_THTPad_1.5x1.5mm_Drill0.7mm
     TP2 -        TestPoint : TestPoint:TestPoint_THTPad_1.5x1.5mm_Drill0.7mm
     TP3 -        TestPoint : TestPoint:TestPoint_THTPad_1.5x1.5mm_Drill0.7mm
     TP4 -        TestPoint : TestPoint:TestPoint_THTPad_1.5x1.5mm_Drill0.7mm
      U2 -          GY-9250 : Connector_PinSocket_2.54mm:PinSocket_1x10_P2.54mm_Vertical
    Vcc1 -              Vcc : Connector_PinSocket_2.54mm:PinSocket_2x03_P2.54mm_Vertical
```

Ilustración 26: Enlace de huellas.

En la imagen anterior observamos que los diez primeros elementos son los capacitores. “Comm.1” se refiere al bus de comunicaciones. Las conexiones J1 a J8 se resumen en la tabla **8** “Qwiic1” hace referencia al conector Qwiic al que se le ha asignado un conector hembra de 1x04 con un espaciado entre pines de 2,54 mm. R1 y R2 son las resistencias SMD de 0Ω explicadas en el capítulo (**4**) junto con TP1 a TP4, que representan los pines de testeo. U2 hace referencia a la IMU, a la que se le asocia un conector hembra de 1x10 con espaciado entre pines de 2,54 mm. Finalmente “Vcc1” es el bus de alimentación.

Tipo de componente	Referencia
J1	Módulo Bluetooth HC-05
J2	Salida del Driver de los magnetorquers X e Y
J3	Conexión de salida del magnetorquer del eje X
J4	Salida del Driver de la rueda de inercia y magnetorquer del eje Z
J5	Vin del driver de la rueda de inercia y magnetorquer del eje Z
J6	Conexión de salida del magnetorquer del eje X
J7	Vin del driver de los magnetorquers X e Y
J8	Conexión de salida de la rueda de inercia y magnetorquer del eje Z

Tabla 8: Resumen de conexiones de J1 a J8.

4.3 Diseño final de la PCB

El último paso antes de generar los archivos de fabricación de la placa será diseñar la disposición de los elementos dentro de la PCB. Para ello, se tienen que tener en consideración algunos criterios a seguir, necesarios para el correcto funcionamiento del sistema.
Lo primero será delimitar el contorno de la PCB. El tamaño será el mismo que el de la placa del proyecto de Juan Palomares**,** en el que se definen unas dimensiones de 93 x 91 mm. La diferencia es que esta vez se incluirán espacios de 3 mm en los costados para poder pasar el cableado de una placa a otra. También se deberá tener en cuenta que, para que la rueda de inercia funcione correctamente, esta debe situarse en el centro de gravedad del satélite, es decir, el centro. Otra limitación es que, para tener el máximo espacio posible para la rueda y conseguir así el mayor momento de inercia posible, se debe dejar la cara inferior completamente libre.
Una vez definidas las limitaciones de espacio, se debe considerar, por orden de prioridad, el lugar óptimo para la disposición de cada elemento. Para ello, se seguirá un orden de manera que primero se ocupan los espacios exteriores de la placa. En la parte exterior y superior colocaremos un magnetorquer. En la parte exterior derecha, el segundo magnetorquer en perpendicular al anterior. En el lado inferior izquierdo, lo más lejano posible a los magnetorquers, se coloca la IMU para que la medida del campo magnético se vea afectada lo mínimo posible durante las maniobras. Los dos drivers, que deben ir lo más juntos posible, se colocan en la parte inferior de la placa. La BluePill se coloca en la parte izquierda de la PCB, por limitación de espacio. El

módulo Bluetooth se coloca justo encima del motor. Finalmente, se colocan los sockets de comunicaciones y de potencia en la parte derecha, donde se encuentra el espacio restante. En el anexo **C** se muestra el plano de la PCB diseñado por Miguel Reurer, donde se señalan los taladros necesarios junto con las cotas dimensionales. En cuanto a los condensadores de desacoplo, se colocan lo más cerca posible de los pines de alimentación para filtrar el máximo ruido posible.

Por último, una vez se colocan todos los componentes, se realizan las conexiones mediante las pistas. Para simplificar considerablemente el ruteado de pistas, se crearán dos capas extra. Una capa será el plano de alimentación y otra capa será el plano de masa. Con ello, obtendremos un diseño de PCB con un total de cuatro capas. Las capas superior e inferior se destinarán al ruteado de pistas. Las pistas tendrán un tamaño de 0,6 mm para obtener una placa más robusta, junto con unas vías de 1 mm y 0,6 mm de taladro.

Una vez finalizado el proceso, nos aseguramos que el diseño cumple las restricciones mínimas del fabricante. Para ello, visitamos la página web de JLCPCB [**25**] y observamos que se cumplen todas. Las más importantes a tener en cuenta son el ancho mínimo de pista (0,127 mm), la distancia mínima entre pistas (0,254 mm), la dimensión mínima de las vías (0,4 mm de taladro) y el tamaño mínimo de los taladros (0,2 mm), entre otros.

4.3.1 Diseño del soporte para los magnetopares

Para conseguir estabilidad en los magnetorquers, se diseña un soporte en el que se fijan mediante tornillos de métrica 2 (M2). De esta manera, los magnetopares irán collados a la placa ya que es importante que ningún componente se mueva cuando se realicen las maniobras. Para ello, se usa el software Solidworks donde se modela el soporte en 3D, que se basa en una especie de anillo de 13 mm de diámetro en el que se introducen los magnetorquers y se ajustan al introducir un tornillo pasante.

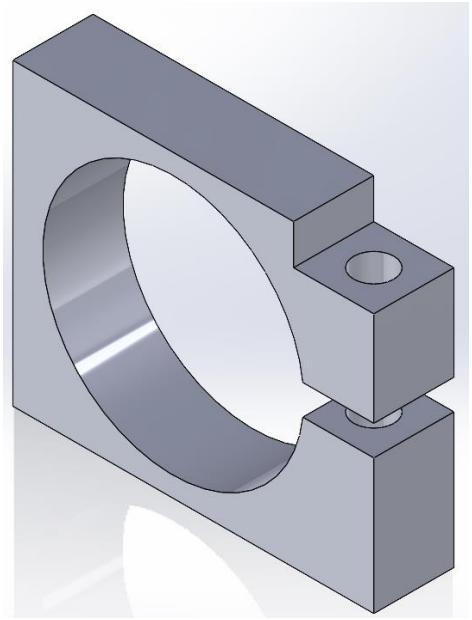


Ilustración 27: Modelo 3D del soporte del magnetorquer diseñado.

En la imagen **27** observamos el diseño final del soporte. Una vez finalizado, se guarda el archivo en formato. STL para su impresión 3D. En el anexo **C** se muestran los planos diseñados por Miguel Reurer con todas las medidas.

Para comprobar que los componentes no chocan entre ellos, se usa la herramienta de visualización 3D de KiCad. En ella podemos observar con tamaños reales mediante modelos 3D de los componentes, como será el resultado final una vez se monte la placa. A continuación, se muestran imágenes del resultado.

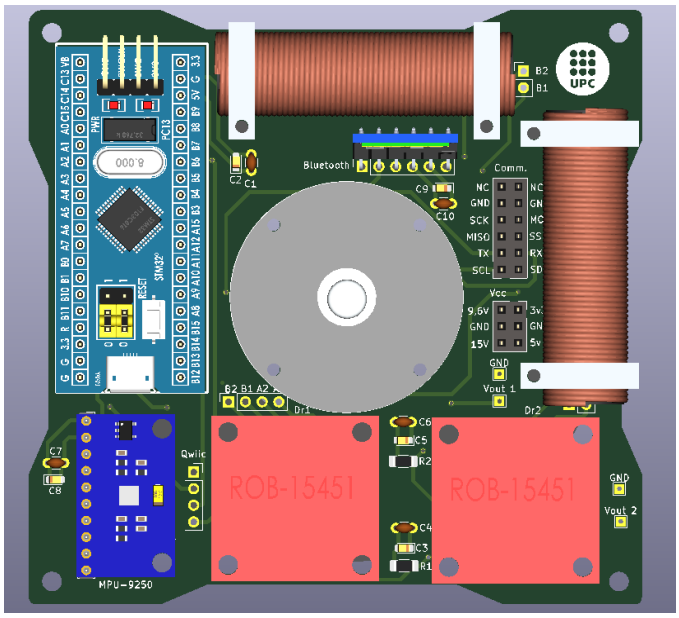


Ilustración 28: Vista de alzado del modelo 3D de la placa ADCS.

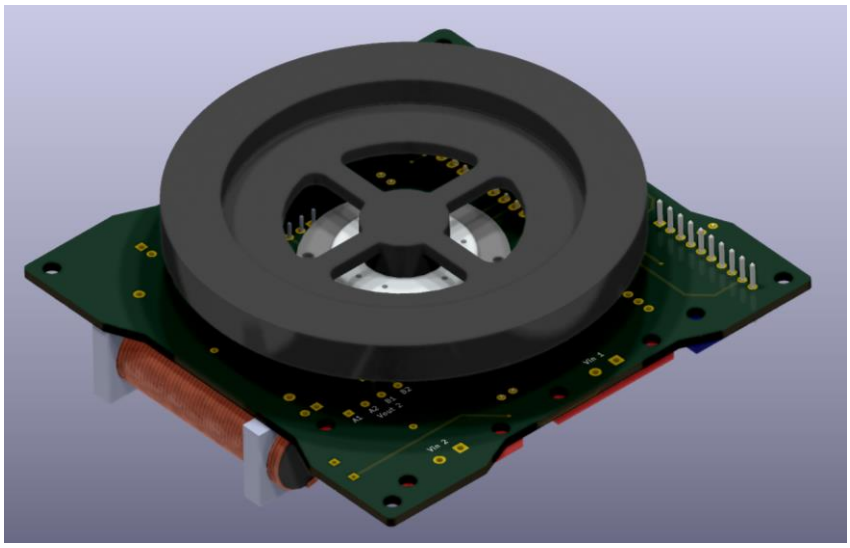


Ilustración 29: Vista inferior de la placa en la que se muestra la rueda de inercia.

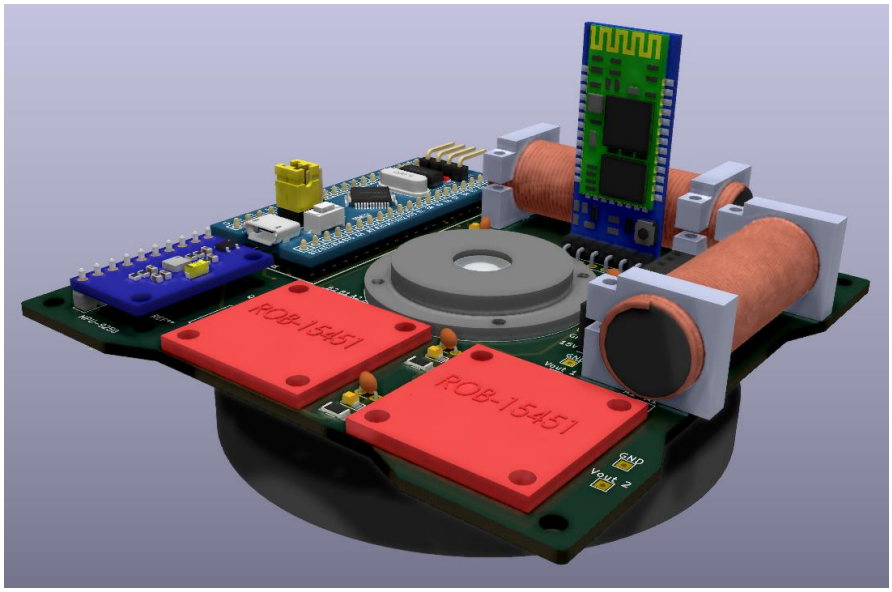


Ilustración 30: Vista general del modelo 3D de la placa ADCS.

Como se observa en las imágenes anteriores, el resultado es correcto ya que ningún componente se sobrepone.

4.4 Obtención de los archivos para la fabricación

El último paso en el diseño de la PCB es generar los archivos necesarios para su fabricación. Estos archivos se denominan archivos Gerber. Estos contienen la información de las capas de la placa, donde se definen las pistas, el tamaño de la PCB, las serigrafías, etc. También se incluyen los archivos Excellon, que contienen la información de los taladros.

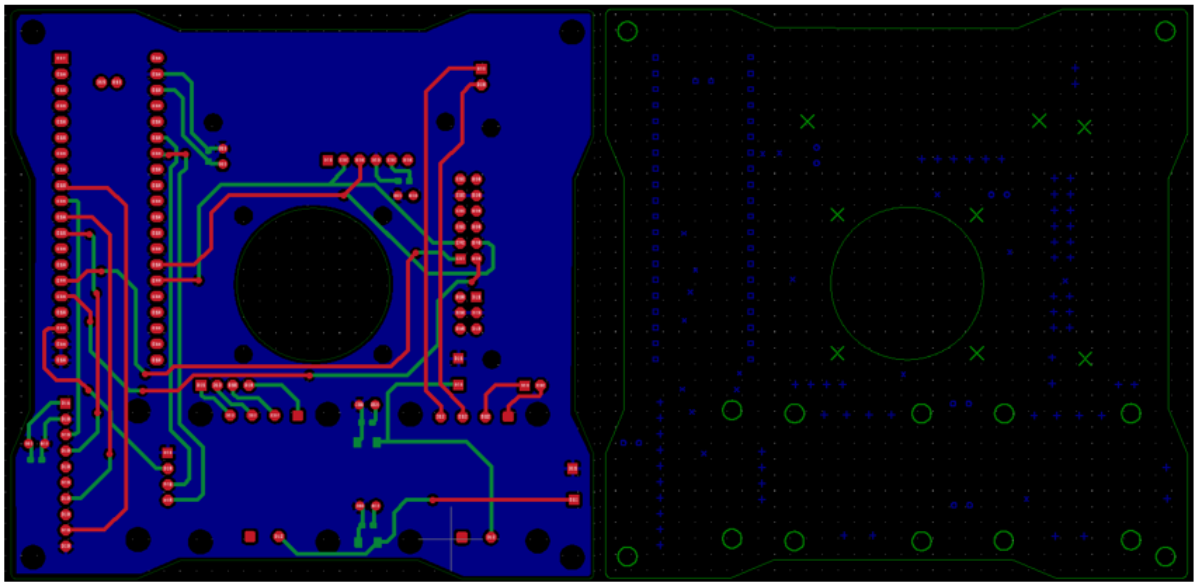


Ilustración 31: Archivos Gerber (izquierda) y Excellon (derecha).

En la imagen de la izquierda observamos el archivo Gerber donde se muestra el enrutado de ambas caras, junto con el plano de masa en azul. A la derecha observamos el archivo Excellon, que muestra los NPTH (*Non-Plated Holes*) en verde y los PTH (*Plated Holes*) en azul.

4.5 PCB fabricada

Una vez generados los archivos Gerber y Excellon, se envían al fabricante JLCPCB. El resultado obtenido se muestra en las siguientes imágenes.

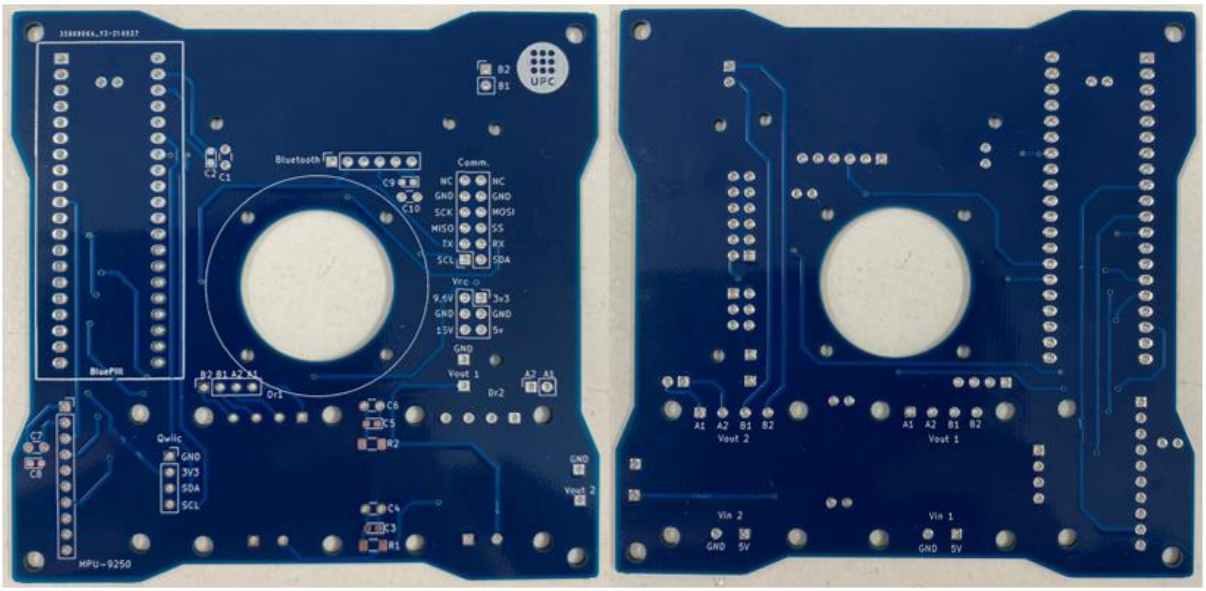


Ilustración 32: Resultado de la fabricación de la PCB. Cara superior en la izquierda e inferior en la derecha.

Una vez tenemos las placas fabricadas, podemos proceder a soldar los componentes para empezar a hacer pruebas de control. En las siguientes imágenes observamos la placa con el montaje completo realizado en el laboratorio TIEG de la ESEIAAT.

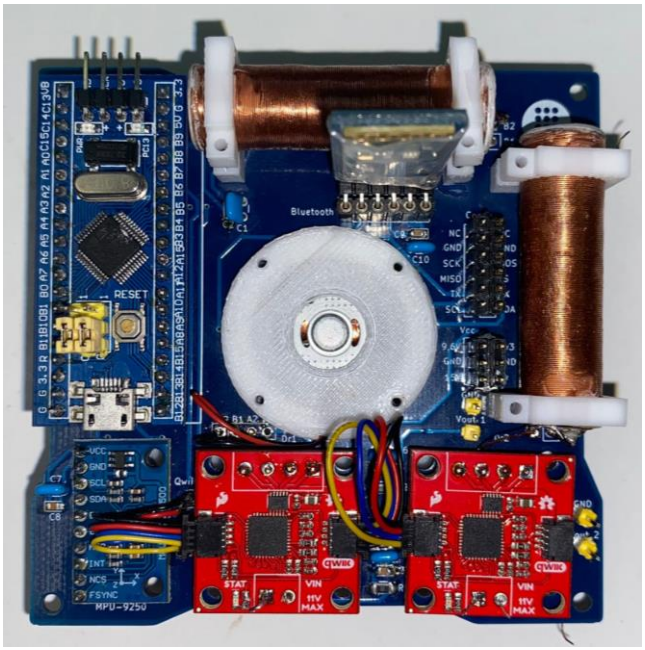


Ilustración 33: Vista superior del resultado final de la placa ADCS.

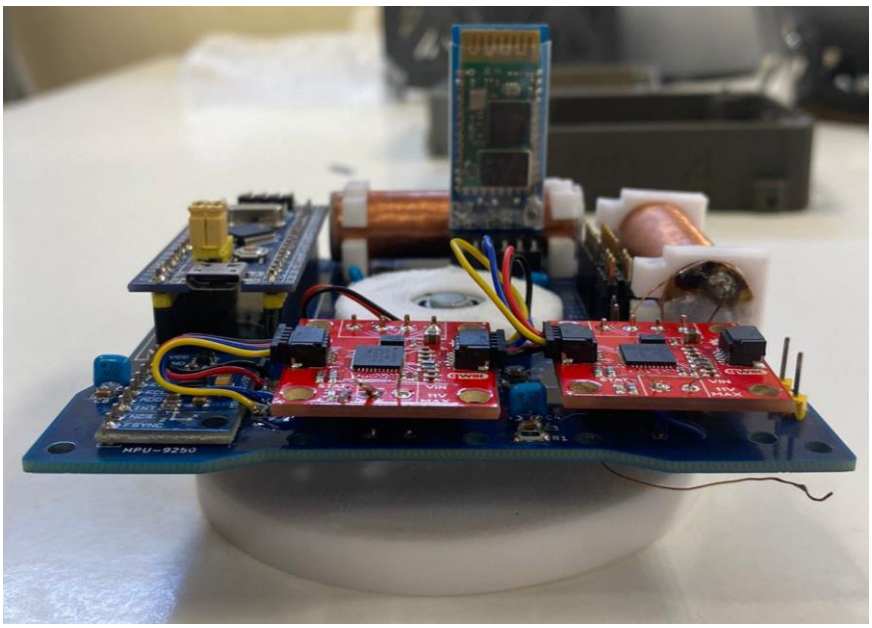


Ilustración 34: Vista frontal del resultado final de la placa ADCS.

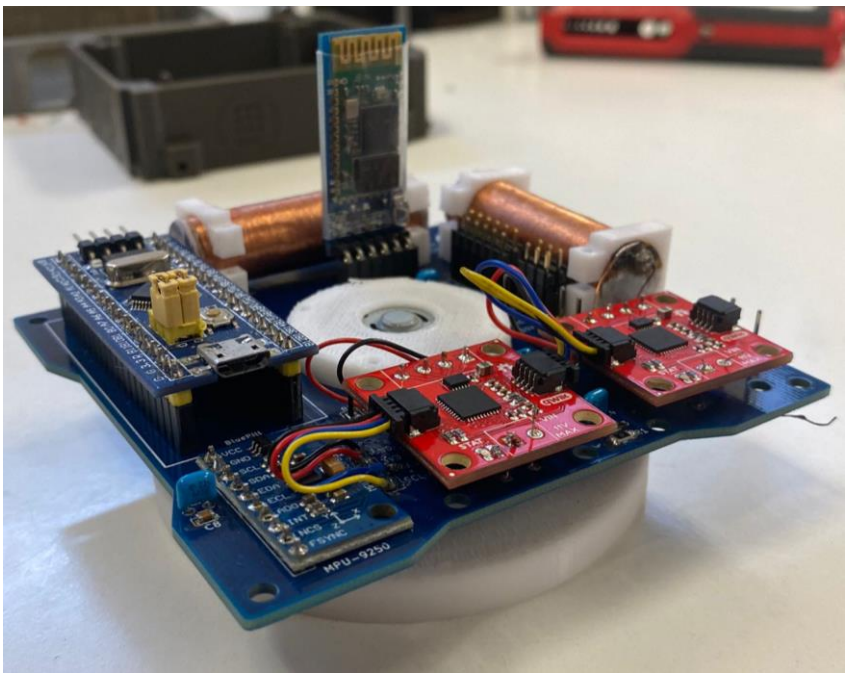


Ilustración 35: Vista general del resultado final de la placa ADCS.

CAPÍTULO 5: CONTROL DE ACTITUD DE LOS PARES MAGNÉTICOS

Existen distintos modos de trabajo por los que pasa el CubeSat durante una misión. En este apartado se explicarán por orden de maniobra tres tipos: la maniobra de detumbling, Nadir Pointing y orientación. Antes de ello, se debe hablar de los pares de perturbación y la saturación de la rueda de inercia, la cual requiere la actuación de los pares magnéticos.

5.1 Pares de perturbación

Las ruedas de reacción, a diferencia de los pares magnéticos, son capaces de ejercer un triple par que abarca todos los grados de libertad de la dinámica de actitud, pero sufren el inconveniente de posiblemente experimentar un aumento gradual de su velocidad de giro, debido a su incapacidad de alterar el impulso total del satélite afectado por perturbaciones externas. En este caso, se activan los pares magnéticos que crean un campo magnético alrededor del CubeSat. El dipolo magnético del satélite se alineará naturalmente con el campo magnético de la Tierra, aunque el trabajo del sistema de control de actitud es mantener una actitud constante, por lo que cambia las velocidades de las ruedas en lugar de permitir que el CubeSat cambie su actitud. Luego, obviamente, los magnetorquers tienen que apagarse tan pronto como las ruedas se hayan desaturado para que no se saturen en la dirección opuesta, momento en el que el sistema comenzaría a moverse para alinearse con el campo magnético de la Tierra.

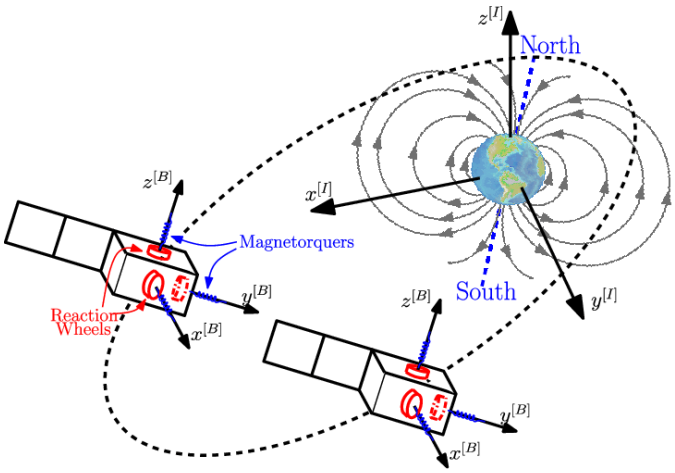


Ilustración 36: Un satélite que apunta orbitando alrededor de la tierra equipado con ruedas de reacción y magnetorquers. Fuente: [26]

Las principales perturbaciones externas que afectan al satélite son las siguientes:

- Pares aerodinámicos: son importantes en órbita baja. En órbitas LEO donde la densidad atmosférica es considerable se producirá una fuerza de arrastre que creará un par de perturbación en el satélite debido a cualquier desplazamiento existente entre el centro aerodinámico de presión y el centro de masa.

- Gradiente gravitatorio: según la ley de Newton:

$$F = G\frac{M1M2}{r^2} \tag{5.1}$$

Donde F es la fuerza de atracción entre dos masas (N) G es la constante de gravitación universal (Nm²/kg²), M_1 y M_2 son las masas de los cuerpos que se atraen (kg) y r es la distancia entre ellos (m) observamos que, de esta manera, la parte más alejada del satélite tiene una atracción gravitatoria menor que la parte más cercana, por lo que el sistema tenderá a alinear su matriz de inercia mínima con la vertical local.

- Pares causados por la presión de radiación solar: Esta actúa sobre toda la superficie del satélite y produce un efecto de acción-reacción dependiendo de cuánto refleje o absorba este tipo de radiación los materiales superficiales del CubeSat. Por lo tanto, se producirá el torque máximo cuando el ángulo de incidencia sea perpendicular al satélite.

- Pares magnéticos: Este par de perturbación surge a través de perturbaciones internas debido a los circuitos eléctricos del satélite que interactúan con el campo magnético terrestre o la magnetización residual de sus componentes.

5.2 Detumbling

Una vez se insiere en órbita el satélite, se detienen los sistemas momentáneamente por razones de seguridad. Cuando el CubeSat ha sido desplegado y los sistemas se vuelven a iniciar, la primera maniobra a llevar a cabo será la de estabilizarse, teniendo en cuenta que en ese instante tendrá una velocidad angular aleatoria. Para ello, se tendrá que disipar dicha velocidad angular aleatoria, teniendo como consigna que:

$$\vec{\omega}_b = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$$

En este proyecto se propone el algoritmo más común para satélites de tamaños reducidos, el B-dot.

5.2.1 Algoritmo B-dot

El principio del algoritmo B-dot se basa en el uso de magnetorquers para generar un par que se opone a la rotación "natural" del satélite (basado en el equivalente de la segunda ley de Newton para los movimientos de rotación).

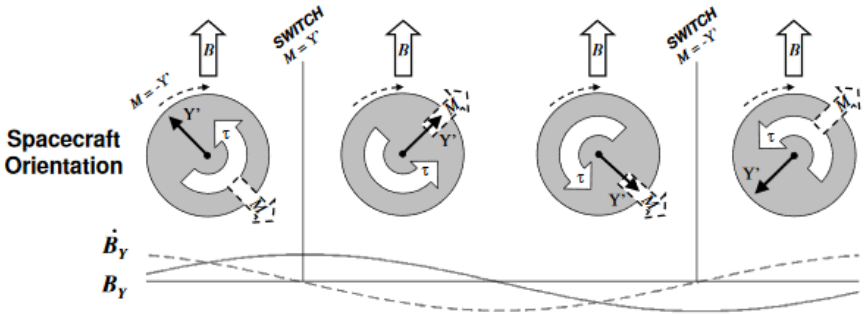


Ilustración 37: B-dot aplicado al satélite de la Weber State University. Fuente: [27]

Según [**28**], el momento dipolar magnético (Am²) sigue la forma:

$$m = -k \times \dot{B} \tag{5.2}$$

Donde k es una constante de ganancia positiva y B es la derivada del vector del campo magnético. Para el cálculo de B-dot seguimos la fórmula:

$$\dot{B} = \frac{B_1 - B_0}{\Delta t} \tag{5.3}$$

Donde B_1 y B_0 denotan dos medidas de campo magnético en un periodo de tiempo Δt.

Así, el algoritmo se puede resumir mediante este diagrama:

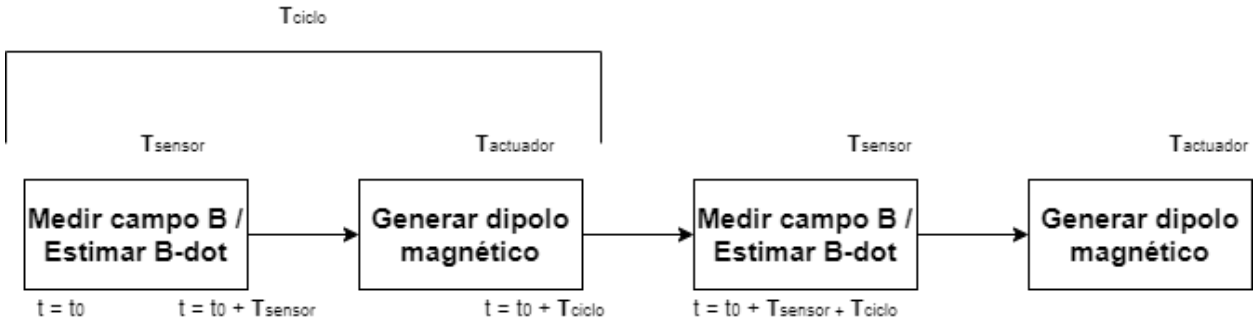


Ilustración 38: Bloques de control del algoritmo B-dot. Fuente: [28]

5.3 Nadir Pointing

Una vez se ha estabilizado el CubeSat, se procede a la fase nominal, en la que el satélite apunta al Nadir, es decir, al centro de la Tierra. De la misma forma, este modo de trabajo puede aplicarse a otros cuerpos celestes, como por ejemplo el Sol.
En este modo también se busca una velocidad angular nula y tener información del cuaternión con respecto al marco orbital.

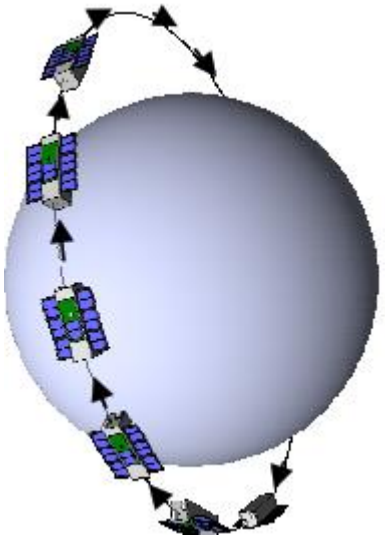


Ilustración 39: Orientación en Nadir-Pointing. Fuente: [29]

En [**30**] se propone la siguiente fórmula para describir el momento dipolar necesario:

$$\vec{m} = \frac{1}{\|\vec{B}\|}\left[\alpha\left(\vec{B}\times\vec{\epsilon}\right) - \beta\left(\vec{B}\times\vec{\omega}_b\right)\right] \tag{5.4}$$

Donde se toman α y β como constantes positivas de ganancia. Como medida de error, se toma el vector de cuaterniones que satisface $q_b = [1\ 0\ 0\ 0]$ y el segundo término con ω_b (rad/s) hace referencia al torque que se opone a la rotación del satélite para frenar.
Finalmente, sustituimos este dipolo magnético en (**2.2**) y obtenemos que el torque producido será:

$$\vec{\tau} = \frac{1}{\|\vec{B}\|}\left[\alpha\left(\vec{B}\times\vec{\epsilon}\right) - \beta\left(\vec{B}\times\vec{\omega}_b\right)\right]\times\vec{B} \tag{5.5}$$

5.4 Orientación

Por último, el modo de orientación surge de la necesidad de comunicación óptica entre dos CubeSats. En este modo de trabajo, se envía un ángulo como consigna para que el satélite gire a partir de su orientación actual. De esta manera, se puede enviar una instrucción al satélite más cercano y transmitir esta información mediante la red de CubeSats que haya en órbita por medio óptico hasta llegar al que deba recibir dicha información.

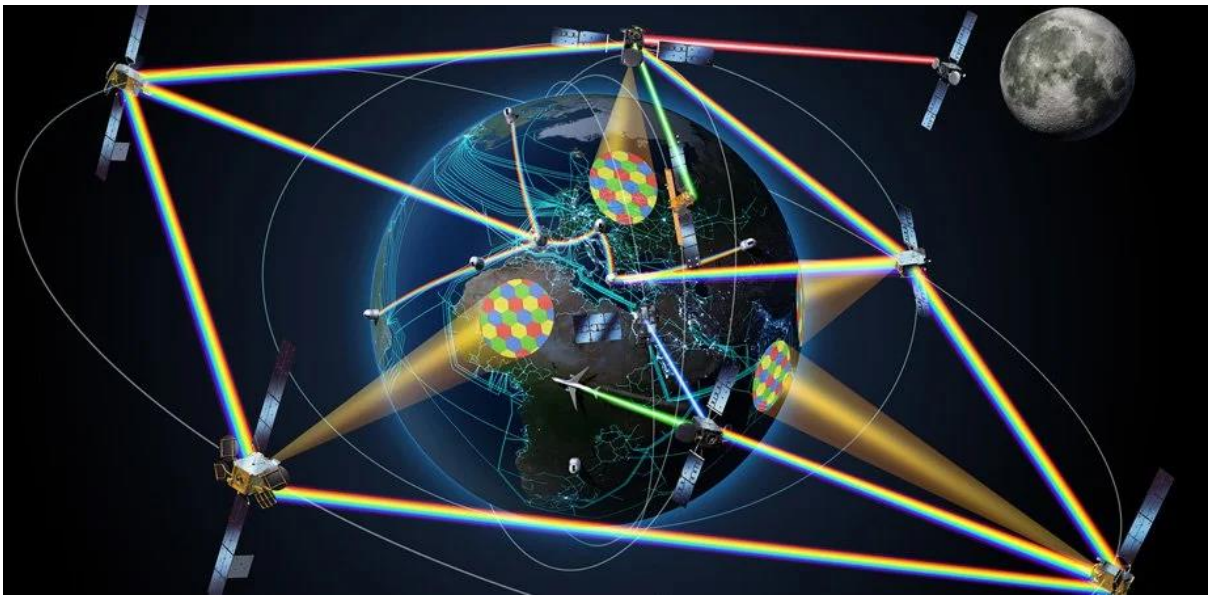


Ilustración 40: Comunicación entre CubeSats por medio óptico. Fuente: [31]

CAPÍTULO 6: VISUALIZACIÓN DE RESULTADOS CON HMI

En este capítulo se explicará el proceso de creación de una interfaz gráfica de tipo HMI (Human Machine Interface), que surge de la necesidad de visualizar los resultados de la simulación en tiempo real, además de poder llevar a cabo un control sobre la placa de una manera más visual. El objetivo final es crear una aplicación que se pueda ejecutar para poder establecer una comunicación con la BluePill y recibir los datos necesarios para graficarlos y poder comparar los resultados obtenidos con los resultados teóricos de proyectos anteriores. Además, en la aplicación se debe poder enviar comandos a la BluePill como por ejemplo el ángulo de giro como consigna y el modo de trabajo del satélite, ya sea Nadir, detumbling o modo de orientación.

Como software de desarrollo de la interfaz se ha escogido Visual Studio 2019 en entorno .NET, un programa muy potente que usa como lenguaje de programación C# (léase C Sharp). Este lenguaje de programación desarrollado por Microsoft, que surge «tomando lo mejor de los lenguajes C y C++, y ha continuado añadiendo funcionalidades, tomando de otros lenguajes, como java, algo de su sintaxis evolucionada.» [**32**]. En este programa se dispone de un panel con tamaño reajustable en el que se añaden todo tipo de objetos (imágenes, botones, gráficos, etiquetas, desplegables…) por lo que lo hace una herramienta muy completa, ya que tiene la capacidad de comunicarse con Arduino mediante el puerto serie para, de esta manera, recibir información y de la misma manera poder enviar comandos.

El programa se encuentra en el anexo **B** explicado por funciones. Sin embargo, es importante resaltar los siguientes puntos respecto a cómo está pensado el programa. La interfaz dispone de un total de cuatro pestañas. En la primera encontramos el menú principal:

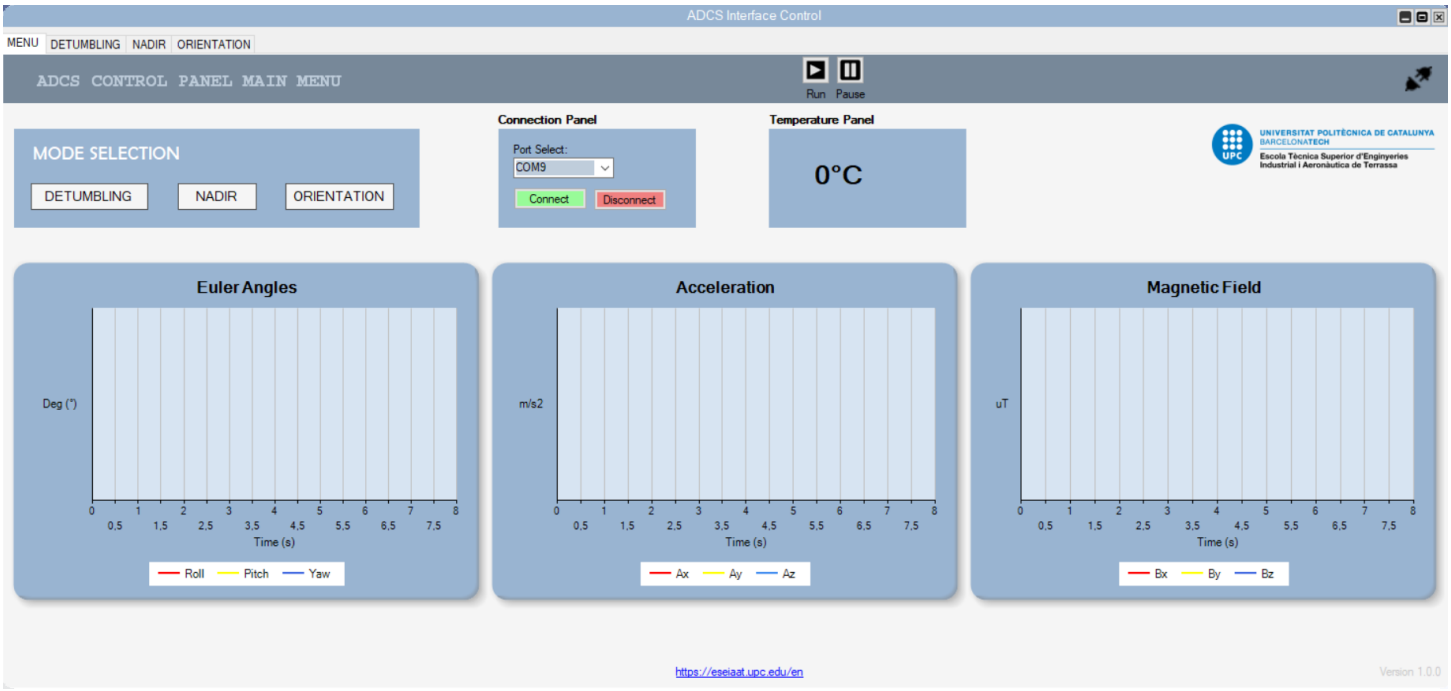


Ilustración 41: Menú principal de la interfaz gráfica

En el menú principal observamos una gran cantidad de información:

- En la parte superior observamos las cuatro pestañas disponibles. La primera pestaña es la actual (menú). La segunda corresponde a la pestaña que realiza el control de variables del primer modo de trabajo que realiza el CubeSat, el detumbling. En la segunda pestaña se muestra la maniobra Nadir. Por último, la pestaña de control del modo de orientación.
- Más abajo observamos el panel de selección de modo. Mediante estos tres botones, se escoge el modo de trabajo del CubeSat para realizar pruebas. Cuando clicamos el botón, se envía una variable por el puerto serie que será recibida por la BluePill, la cual corresponde al modo de trabajo. Para la variable "modo" igual a 1, el modo de trabajo será el detumbling. Para "modo" igual a 2 Nadir y para "modo" igual a 3 orientación.
- En el "Connection Panel" encontramos la configuración de conexión. En ella seleccionamos el puerto serie que usa el Arduino para establecer comunicación.
- Una vez se establece la conexión, se habilitan las gráficas. En los botones superiores "Run" y "Pause" podemos controlar la ejecución del programa y por tanto leer o detener los datos provenientes de la BluePill.
- A la derecha de estos dos botones se encuentra el estado de conexión con el puerto serie.

- En el recuadro de temperatura se muestra la temperatura del hardware, medida por la IMU. Se ha programado para que, cuando las temperaturas alcancen un valor de entre 50ºC y 75ºC el valor se muestre en amarillo y un LED virtual se encienda en la parte superior derecha. Para temperaturas superiores a 75ºC el valor se muestra en rojo y un LED rojo se enciende de la misma manera en la parte superior derecha. Estos valores se justifican ya que la temperatura soportada por el hardware tiene un rango de entre -40ºC y 85ºC [**33**][**34**].
- En la parte central del panel encontramos las tres gráficas principales del sistema, que muestran los valores de los ángulos de Euler en grados (Pitch, Yaw y Roll), la aceleración en m/s² en los tres ejes y el campo magnético medido por la IMU también en los tres ejes.
- En la parte inferior del panel encontramos un enlace a la página web de la universidad y la versión de la interfaz.

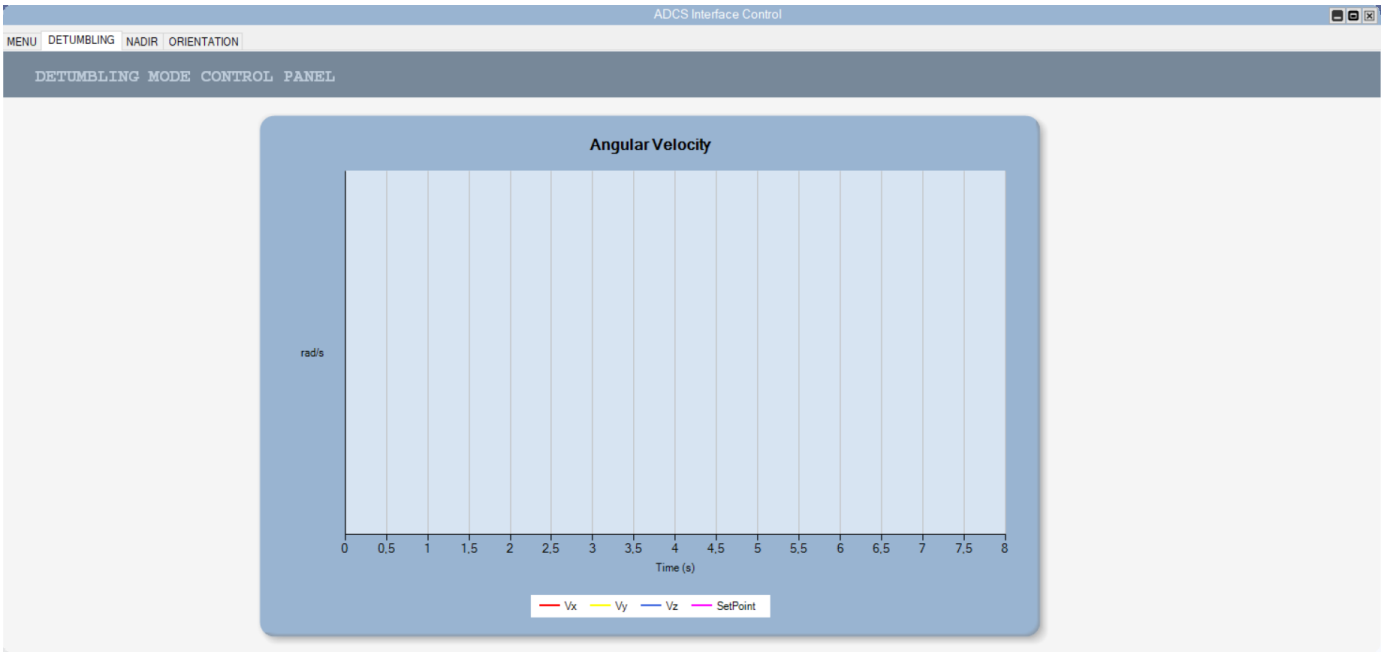


Ilustración 42: Panel de control del modo detumbling.

En la primera pestaña observamos el panel de control del modo de trabajo detumbling. En este se encuentra una gráfica de velocidad angular en los tres ejes. Si nos fijamos en la leyenda en fucsia vemos el “SetPoint”, que en este caso lo que se quiere conseguir es frenar el nanosatélite, por lo que siempre la consigna será velocidad angular cero. De la misma forma se crea otra gráfica con las mismas características en la pestaña Nadir (velocidad angular igual a cero).
En la última pestaña tenemos el panel de control del modo de orientación:

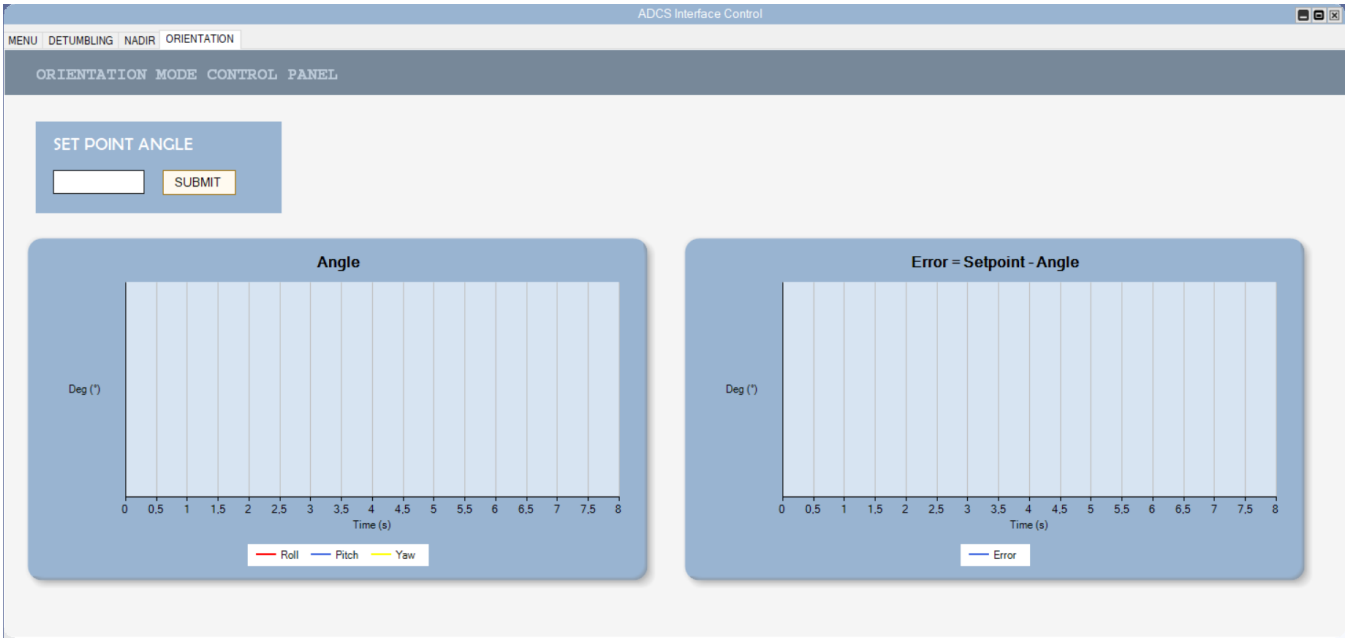


Ilustración 43: Panel de control del modo orientación.

En este panel encontramos, concretamente, tres elementos. El primero, en la parte superior es el ángulo consigna que se envía a la BluePill. Se introduce un ángulo (positivo o negativo) y mediante el botón “Submit” se envía el ángulo introducido en el textbox mediante el puerto serie del microcontrolador. En la parte inferior observamos dos gráficas. En la primera observamos de nuevo los ángulos de Euler, en los que podremos controlar el giro del CubeSat. A la derecha de esta gráfica tenemos el error en grados. El error, en este caso, es la resta del ángulo consigna introducido y la guiñada (Yaw).

6.1 Resultados experimentales

Para la simulación se realiza un montaje simple en el que se incluyen dos módulos montados sobre un air bearing. El air bearing es un simulador que consta de un tubo de aire comprimido que incide en la parte inferior de una esfera que envuelve el CubeSat para reducir al mínimo la fricción y recibir resultados más próximos a lo que sería un entorno de gravedad cero. En el primer módulo se coloca una batería de 4,5V simulando la placa de alimentación del satélite y encima de este el segundo módulo con la placa ADCS. Inicialmente, el prototipo se diseñó para realizar las simulaciones del air bearing dentro de un simulador magnético diseñado por Marcel Colet en el proyecto «*Disseny d’un simulador de camp magnètic terrestre en òrbites LEO per proves de control d’actitud amb prototip de CubeSat* ». En él se simulaba un entorno con el campo magnético distribuido de manera más fiel a la intensidad recibida en órbitas terrestres bajas, aunque también permitiría intensificar el campo magnético

para recibir una respuesta de los magnetorquers más rápida. Sin embargo, por cuestiones de tiempo e incidencias con la empresa encargada de la impresión 3D de los componentes, se tuvo que prescindir de la simulación.

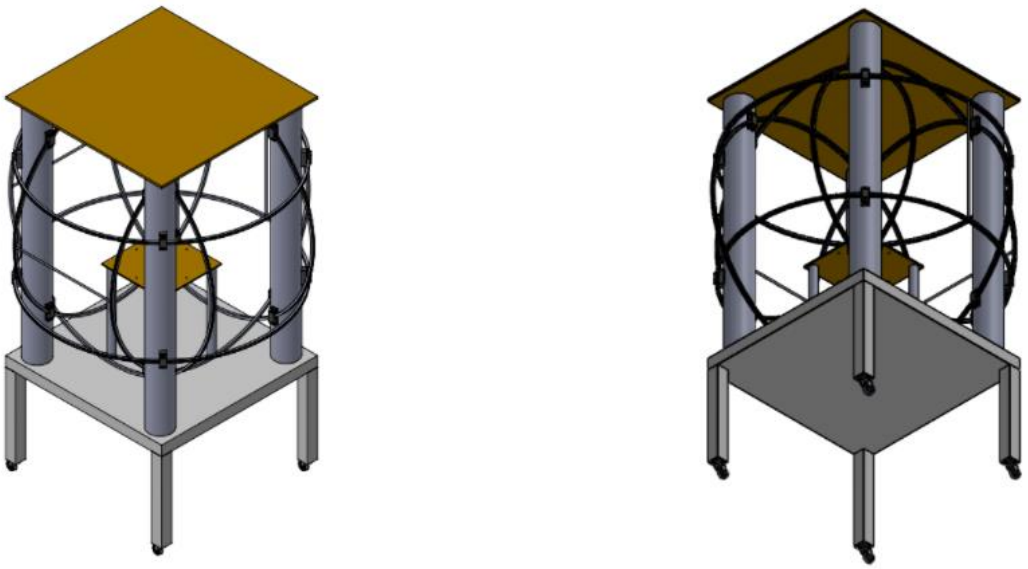


Ilustración 44: Modelo 3D del simulador óptico realizado por Marcel Colet.

Como alternativa, se colocaron dos bobinas para intensificar el campo magnético y conseguir una respuesta para la simulación ya que, además, el compresor disponible en el laboratorio no permite más de 8L de aire por lo que las simulaciones deben ser de periodos cortos, teniendo en cuenta el peso del conjunto.

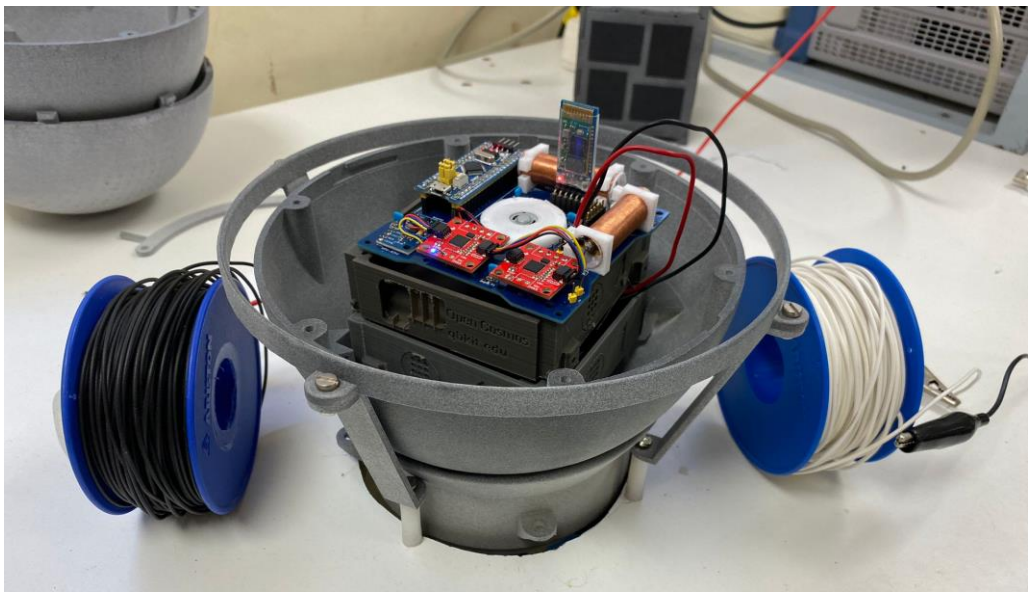


Ilustración 45: Sistema ADC montado sobre Air Bearing.

Antes de empezar la simulación, es importante conocer qué sentido tomará la corriente por cada bobina cuando demos las instrucciones a los drivers de corriente mediante la BluePill.

En el momento del montaje de la PCB en el que se soldaron los extremos de cada bobina, se escogió la instrucción de sentido “0” en el driver hacia afuera y “1” hacia dentro.

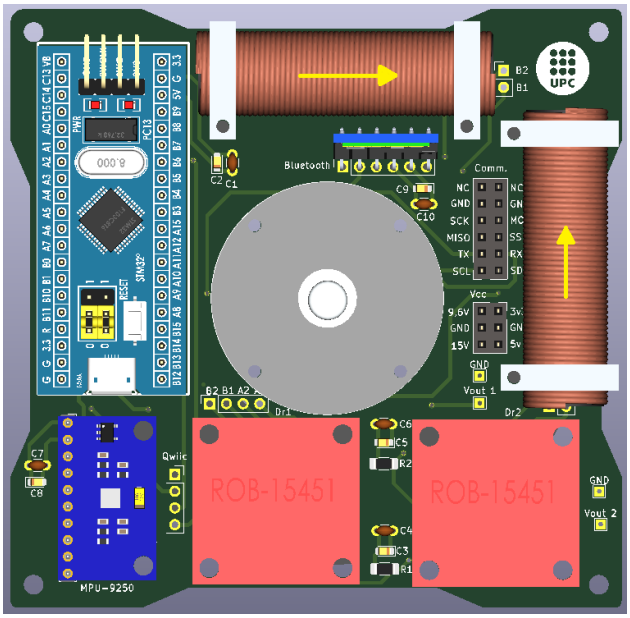


Ilustración 46:Sentido de la corriente de los magnetorquers con el parámetro “sentido” en los drivers igual a 1.

Para empezar la simulación, se conecta la interfaz a la BluePill mediante Bluetooth y se inicializan las gráficas con las bobinas de intensificación de campo encendidas. Una vez iniciadas, se envía una corriente máxima en ambos pares magnéticos, obteniendo un dipolo magnético con un ángulo θ de 45º respecto al eje de la placa. Se observa como el conjunto empieza a moverse hacia la línea de campo que crean las bobinas externas, que es perpendicular al eje transversal de estas. En las gráficas siguientes se muestra el resultado obtenido:

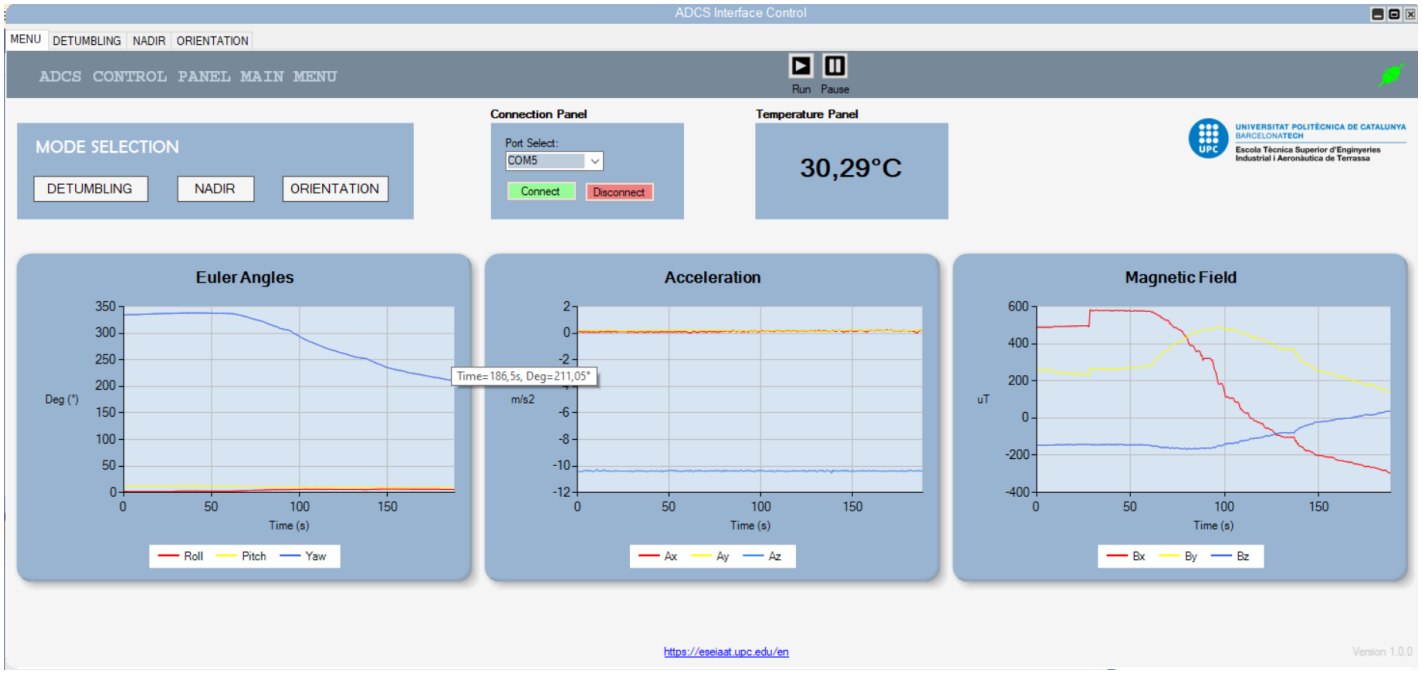


Ilustración 47: Resultado general de la simulación.

El tiempo total de la simulación es de unos tres minutos, ya que en ese instante se empieza a perder presión de aire y empieza a haber fricción de la esfera con el air bearing, por lo que los resultados dejan de ser concluyentes. A continuación, se explicará en cada gráfica si el resultado obtenido es el esperado.

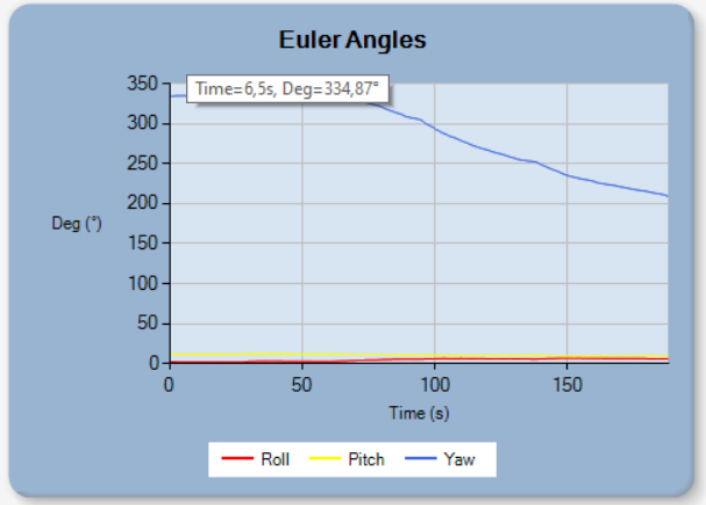


Ilustración 48: Gráfica detallada de ángulos de Euler.

Observamos un ángulo inicial en Yaw de 334,87º cuando el sistema está en reposo. Esta posición se escoge para tener una orientación en la que el campo de las bobinas es perpendicular al dipolo que se creará en los pares magnéticos cuando se inicie la simulación. A los 21 segundos, aproximadamente, se aplica la corriente a los

magnetorquers e instantes después, a los 55 segundos aproximadamente, el conjunto empieza a rotar hasta llegar a un ángulo de 211,05º, realizando así una rotación total de 123,82º en sentido antihorario en unos 2 minutos y 12 segundos. Estos resultados son esperados ya que el dipolo de los magnetorquers tiende a alinearse al dipolo de las bobinas externas, por lo que se mueve en la dirección que las une. Llega un punto en el que el sistema está alineado, pero el conjunto tiene una inercia debida al torque que adquiere y por lo tanto, se pasa. Debido al tiempo reducido de la simulación, no se observa la actitud que adquiere a partir de este punto, pero es de esperar que el conjunto tienda de nuevo a alinearse con el campo de las bobinas, por lo que crearán un torque de sentido contrario hasta alinearse y volver a sobrepasarlo, pero esta vez con menos fuerza. Se repetiría el proceso de nuevo hasta que llega un punto en el que se alinean perfectamente. De esta manera, esperaríamos un resultado gráfico con forma de sistema sobre amortiguado, con la siguiente forma:

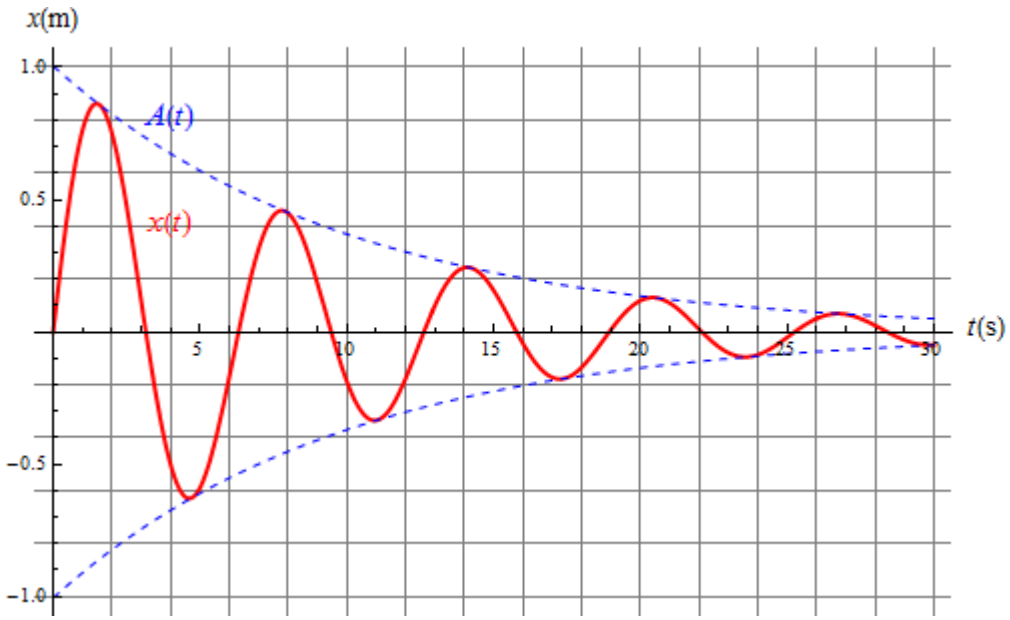


Ilustración 49: Ejemplo de un sistema sobre amortiguado. Fuente: [35]

En cuanto a la gráfica de la aceleración, apenas observamos cambios en los valores en cualquier eje, ya que la velocidad a la que se mueve el sistema es prácticamente constante.

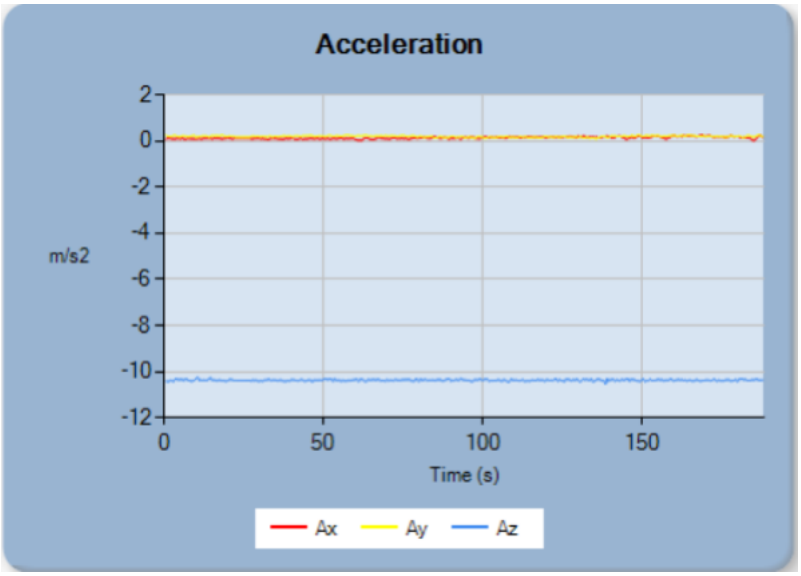


Ilustración 50: Gráfica detallada de la aceleración.

Podemos ver la aceleración en el eje Z, perpendicular al plano de la placa con una aceleración de 10,1 m/s² correspondiente a la aceleración de la gravedad (teniendo una mejor calibración se pueden observar los 9,81 m/s² reales).

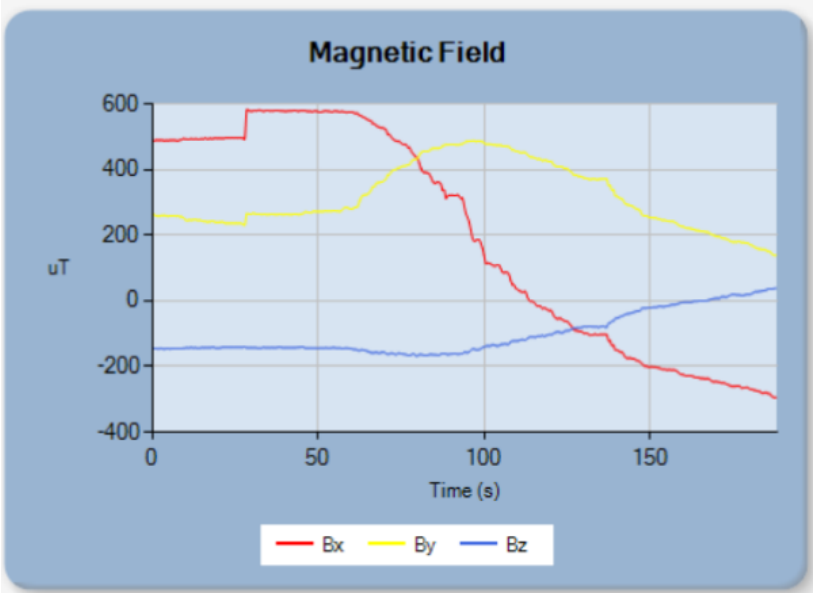


Ilustración 51: Gráfica detallada del campo magnético.

En la gráfica de campo magnético, observamos un pequeño salto del campo magnético en los ejes x e y al aplicar la corriente a los magnetorquers.

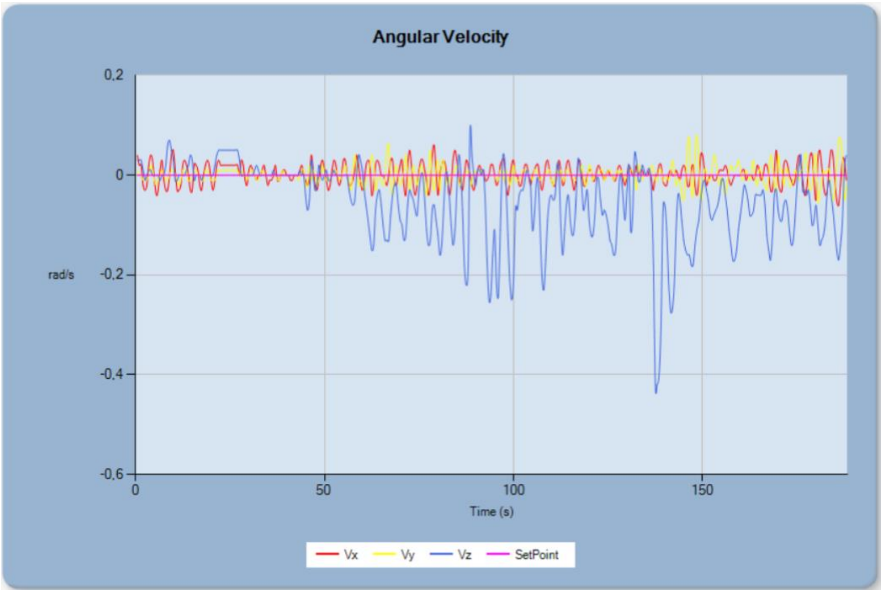


Ilustración 52: Gráfica detallada de la velocidad angular

En cuanto a la gráfica de velocidad angular, extraída de la pestaña del panel de control del modo detumbling, observamos velocidades en el eje X e Y que no sobrepasan los 0,07 radianes por segundo, equivalentes a unos 4 grados por segundo. Si comparamos con la gráfica de los ángulos de Euler, se ha observado que gira unos 124º en 130 segundos, que si dividimos en una media de 1 grado por segundo (0,017 rad/s de media) obtenemos unos 130º girados, lo cual se aproxima bastante al valor obtenido en la gráfica. En cuanto al eje Z, observamos perturbaciones debidas al balanceo del sistema, ya que no estaba completamente estable.

Por último, es importante mencionar lo siguiente:
Como se ha mencionado anteriormente, en la parte superior del panel de control del modo de orientación se da la instrucción del ángulo consigna a la BluePill y esta es la encargada de calcular las corrientes que deben suministrarse para realizar el giro. A pesar de esto, en esta simulación no se ha realizado un control real del satélite, sino que se han realizado pruebas suministrando la intensidad máxima a los pares magnéticos. Se ha aprovechado la consigna dada mediante la interfaz a la BluePill para usar esta variable como magnitud de corriente. Es decir, se ha diseñado el programa de la BluePill de tal manera que el valor recibido en la variable “angulo” (referente al ángulo consigna) se usa para dar la instrucción de la magnitud de la señal PWM (0-255).

```
1 void drive_CubeSat(){
2
3   MT.setDrive( 0,sentido, angulo); //MT Y
4   MT.setDrive( 1,sentido, angulo); //MT X
5
6 }
```

Observamos que cuando damos la instrucción de giro, se suministra la misma corriente (dada por el valor de la variable “angulo”) para los dos magnetopares. El sentido por defecto es 1 para corrientes positivas y 0 para corrientes negativas. Esta lógica se puede observar en el programa “07_Send_Data.ino” adjuntado en el anexo **A**.

Dicho esto, observamos finalmente la última pestaña de la interfaz:

Ilustración 53: Resultado de la simulación en el panel de control del modo de orientación.

La gráfica “Angle” corresponde a los ángulos de Euler, ya explicados anteriormente. En cuanto a la gráfica del error, observamos la resta del ángulo consigna introducido (en este caso un valor de 0 a 255) y la guiñada (Yaw). No tiene sentido analizar esta gráfica ya que no representa un dato relevante, simplemente se observa que, al inicio de la simulación, teniendo un ángulo inicial de 334º, la resta de SetPoint (inicialmente 0) y Yaw es -334. Una vez presionamos el botón “Submit” para enviar el valor a la variable “angulo” del programa del microcontrolador, la resta obtiene un valor de -79. En un programa con control de orientación, deberíamos ver una gráfica que tiende a 0, en la que el ángulo consigna y el ángulo girado fueran el mismo.

CAPÍTULO 7: RESULTADOS NO TÉCNICOS

En el presente capítulo se expondrán las distintas tareas realizadas durante este proyecto junto con su planificación. Además, se presentará un resumen de los costes aproximados y el impacto ambiental del proyecto.

7.1 Planificación

En este apartado se presentarán las tareas realizadas junto con el tiempo invertido en ellas y la duración de las mismas. Finalmente, se adjuntará el diagrama de Gantt que muestra un resumen temporal de las tareas.

7.1.1 Identificación de las tareas

La correcta realización del presente proyecto conlleva la realización de las siguientes tareas ordenadas:

- Planificación inicial del proyecto:

1.1 Establecimiento de tareas
1.2 Redacción del Project Charter

- Búsqueda de información

2.1 Estudio del estado del arte de los CubeSats
2.2 Estudio teórico de las barras de torsión y su principio físico

- Diseño de la placa de control de actitud

3.1 Estudio de los componentes disponibles
3.2 Estudio del prototipo
3.3 Montaje del prototipo
3.4 Programación del microcontrolador
3.5 Diseño y fabricación de la PCB
3.6 Montaje de la PCB
3.7 Comprobación del funcionamiento

- Diseño de la interfaz

4.1 Estudio de las necesidades
4.2 Estudio del software
4.3 Programación y diseño de la interfaz
4.4 Prueba de funcionamiento

- Documentación final

5.1 Redacción del informe

7.1.2 Extensión de las tareas

Código de la tarea	Duración en horas
1.1	5
1.2	5
2.1	15
2.2	15
3.1	20
3.2	20
3.3	30
3.4	40
3.5	50
3.6	20
3.7	15
4.1	5
4.2	20
4.3	40
4.4	20
5.1	280

Tabla 9: Extensión de las tareas.

7.1.3 Diagrama de Gantt

A continuación, se muestra la planificación del proyecto mediante el diagrama de Gantt. Podemos observar cómo hay tareas que dependen entre ellas y hay otras que se han realizado paralelamente.

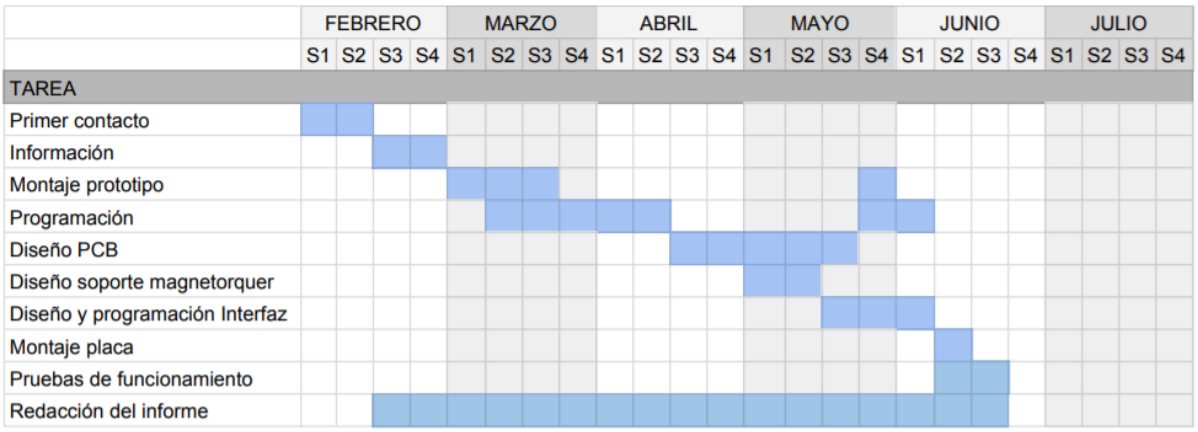


Tabla 10: Diagrama de Gantt

7.2 Estudio económico

A continuación, se realizará un estudio de los costes de realización del proyecto, teniendo en cuenta tanto el material usado como el tiempo invertido.

7.2.1 Presupuesto de la investigación

Para el presupuesto de investigación, se tiene en cuenta el tiempo invertido en la redacción del informe, el ensamblaje del hardware, el estudio teórico del sistema y la programación del software. A continuación, se muestra una tabla con el coste total, teniendo en cuenta un sueldo de ingeniero electrónico de 30€/hora y las horas invertidas, especificadas en la tabla **9**.

Presupuesto de investigación y desarrollo		
Tiempo empleado (h)	Coste unitario (€/h)	Total (€)
600	30	18.000,00

Tabla 11: Presupuesto de investigación

Seguidamente, se muestra una tabla con el consumo eléctrico que supone el desarrollo del proyecto, teniendo en cuenta una tarifa extraída de [**36**]:

Coste consumo eléctrico				
Concepto	Potencia (kW)	€/kWh	Horas	Total (€)
Ordenador	0,19	0,14906	600	17,00

Tabla 12: Presupuesto sobre el consumo eléctrico

7.2.2 Presupuesto de los materiales

En la siguiente tabla se muestran los costes de los materiales necesarios para la fabricación de la placa ADCS:

Coste material			
Concepto	Cantidad	Precio Unitario (€)	Total (€)
Barras de ferrita	4	1,78	7,12
Alambre de cobre	1	2,29	2,29
Cinta aislante	1	0,89	0,89
BluePill	1	5,79	5,79
Arduino Nano	1	19	19
ROB-15451	1	12,43	12,43
Módulo GY-9250	1	3,60	3,6
USB ST-Link V2	1	4,99	4,99
Módulo Bluetooth HC-05	1	5,00	5,00
Cables Qwiic	1	1,26	1,26
Cables Protoboard	1x60	4,36	4,36
Protoboard	2	4,99	9,98
PCB	5	6,56	33,69
TOTAL (€)			110,40

Tabla 13: Presupuesto del material

7.2.3 Presupuesto total

Por último, se muestra en la siguiente tabla el presupuesto total del proyecto, incluyendo el coste de investigación, coste eléctrico y el coste del consumo.

Presupuesto total	
Concepto	Coste (€)
Investigación y desarrollo	18.000,00
Coste energético	17,00
Coste material	110,40
Total (€)	18.127,40

Tabla 14: Presupuesto total

Así pues, el coste total del proyecto asciende a la cantidad de dieciocho mil ciento veintisiete euros con cuarenta céntimos.

CAPÍTULO 8: CONCLUSIONES Y RECOMENDACIONES

A fin de dar por concluido el proyecto, se destina el último capítulo a la redacción de las conclusiones. Posteriormente, se incluirá también un apartado de recomendaciones para próximos proyectos relacionados con el presente.

8.1 Conclusiones del proyecto

En este proyecto se buscaba realizar el diseño y manufacturación de una placa de control de actitud para un prototipo de CubeSat. La principal limitación que se ha presentado ha sido las dimensiones del satélite.
Al ser una primera versión, lejana de cualquier sistema final que se considere listo para su uso, se han usado componentes prefabricados que limitan, si cabe más aún, el espacio de trabajo a la hora de diseñar la PCB. El principal elemento afectado por ello han sido los magnetorquers. Su diseño se ha visto perjudicado por las restricciones dimensionales y han causado que las prestaciones del satélite se hayan visto reducidas en cuanto a intensidad del dipolo creado. Es por ello que, en las simulaciones de prueba, realizadas al final, se ha tenido que hacer uso de un elemento externo para intensificar la interacción de los campos magnéticos.

Otra limitación presentada ha sido la distribución del laboratorio a principios del proyecto. Por razones de la pandemia, el laboratorio disponía de un aforo limitado, por lo que acotaba las horas disponibles de laboratorio.

Aun así, se ha llevado a cabo un gran trabajo entre todos los proyectistas y docentes del proyecto PLATHON, con los que ha habido reuniones semana tras semana para obtener el mejor resultado posible. El proyecto se ha considerado un éxito ya que se ha comprobado su funcionamiento. Aunque sea un prototipo al que le falta mucho por madurar, esto es, de hecho, un gran comienzo.

8.2 Recomendaciones futuras

Esta tesis ha abarcado una gran variedad de conceptos que se pueden tener en cuenta en proyectos futuros. Desde principios teóricos de los elementos, uso de hardware, control del satélite, etc. De todas formas, aún hay muchos factores a tener en cuenta para llegar al objetivo final.

Las recomendaciones se pueden clasificar en recomendaciones a corto y largo plazo.

Las de corto plazo son las recomendaciones que daría a la persona que sigue inmediatamente este proyecto y serían las siguientes:

- La realización del control de actitud en los modos de trabajo de detumbling, Nadir y orientación. Para ello se puede usar el algoritmo B-dot junto con un controlador PD.

- La optimización de la placa ADCS. Para ello, se deberían integrar únicamente los componentes necesarios. Por ejemplo, en lugar de usar una BluePill como microcontrolador, se puede integrar directamente en la placa el microcontrolador que lleva integrado, para reducir los pines inutilizados.

- Para el uso de la IMU, es recomendable hacer uso del protocolo I2C ya que hay mucha más información relativa a la extracción de cuaterniones del DMP y de esta manera se reduce el tiempo de cálculo y la carga del microcontrolador.

A largo plazo, se podrían realizar las siguientes tareas:

- Incluir una tercera bobina en el eje Z para obtener un control total de los pares magnéticos. La PCB diseñada tiene destinadas dos salidas de uno de los drivers para ello.

- La integración y comunicación de los sistemas ya diseñados. Estos son EPS, placa de comunicaciones y placa ADCS.

Bibliografía

[1] ISIS. *ISIPOD CubeSat Deployer*, https://www.isispace.nl/product/isipod-cubesat-deployer/.

[2] SEGWiki. “1.” *Momento Dipolar Magnético*, https://wiki.seg.org/wiki/Dictionary:Magnetic_dipole_moment/es.

[3] Bellini, Niccolò. “Magnetic Actuators for Nanosatellite Attitude Control.” 2014, https://core.ac.uk/download/pdf/31157054.pdf.

[4] Garcia, Oriol Laserna. “Design of a cubesat attitude 2D control based on the Earth's magnetic field.” *UPCCommons*, 2020, http://hdl.handle.net/2117/328987.

[5] Santiago, Mario Castro. *Cubesat Attitude Control System based on embedded magnetorquers in photovoltaic panels*, 2018, https://digibug.ugr.es/bitstream/handle/10481/53817/tfg_mario_v03_Definitiva.pdf?sequence=1&isAllowed=y.

[6] “Varilla de ferrita Fair-Rite, diámetro 8mm, altura 45mm.” *RS Components*, https://es.rs-online.com/web/c/componentes-pasivos/nucleos-de-ferrita/varillas-de-ferrita/?applied-dimensions=4294863120.

[7] Alibaba. *Enameled copper wire coil*. https://www.alibaba.com/product-detail/enameled-copper-wire-copper-wire-coil_1578886454.html.

[8] Hard Zone. *Todo lo que necesitas saber sobre los procesadores ARM*, https://hardzone.es/tutoriales/componentes/procesador-arm/.

[9] Laboratorio Gluon. *STM32 CDC con CubeMX en Bluepill*, https://www.laboratoriogluon.com/stm32-cdc-con-cubemx-en-bluepill/.

[10] STM32duino Wiki. *Blue Pill*, https://stm32duinoforum.com/forum/wiki_subdomain/index_title_Blue_Pill.html

[11] STMicroelectronics. “STM32 Datasheet.” https://www.st.com/resource/en/datasheet/stm32f103c8.pdf.

[12] InvenSense. *MPU-9250 Datasheet*. https://invensense.tdk.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf.

[13] ElectroPeak. *MPU9250 SPI/I2C 9-Axis Gyro Accelerator Magnetometer Module*, https://electropeak.com/imu-ahrs-i2c-mpu9250.

[14] Wikipedia. *Ejes del avión*, https://es.wikipedia.org/wiki/Ejes_del_avi%C3%B3n.

[15] LaptrinhX. *Arduino Nano Pinout, Board Layout, Specifications, Pin Description*, https://laptrinhx.com/arduino-nano-pinout-board-layout-specifications-pin-description-3402138136/.

[16] Arduino. “Arduino Nano Datasheet.” https://www.arduino.cc/en/uploads/Main/ArduinoNanoManual23.pdf.

[17] ITead Studio. “HC-05 Datasheet.” https://components101.com/asset/sites/default/files/component_datasheet/HC-05%20Datasheet.pdf.

[18] Naylamp Mechatronics. *MÓDULO BLUETOOTH HC05*, https://naylampmechatronics.com/inalambrico/43-modulo-bluetooth-hc05.html.

[19] Naylamp Mechatronics. *CONFIGURACIÓN DEL MÓDULO BLUETOOTH HC-05 USANDO COMANDOS AT*,

https://naylampmechatronics.com/blog/24_configuracion-del-modulo-bluetooth-hc-05-usando-comandos-at.html.

[20] 3CU Electrónica. *Comandos AT*, https://sites.google.com/site/3cuelectronica/home/comandos-at-1.

[21] bitwiseAR. “Curso Arduino desde Cero.” *Github*, 2020, https://github.com/bitwiseAr/Curso-Arduino-desde-cero/blob/master/Capitulo24/Capitulo24-Programa1.txt.

[22] Ndsybkiot. *Split String Arduino*, https://ndsybkiot.wordpress.com/2018/01/17/split-string-arduino/.

[23] SparkFun. *Hookup Guide for the Qwiic Motor Driver*, https://learn.sparkfun.com/tutorials/hookup-guide-for-the-qwiic-motor-driver?_ga=2.213344276.1886170875.1624298498-877969878.1615032729.

[24] yet-another-average-joe. *KiCad STM-32*, 2020, https://github.com/yet-another-average-joe/Kicad-STM32.

[25] JLCPCB. *PCB Capabilities*, https://jlcpcb.com/capabilities/Capabilities.

[26] Trégouët, Jean-François, et al. *Reaction Wheels Desaturation Using Magnetorquers and Static Input Allocation*. 2015, https://www.researchgate.net/publication/273395432_Reaction_Wheels_Desaturation_Using_Magnetorquers_and_Static_Input_Allocation.

[27] EC3SAT. *How to slow down satellite rotation ? - The B-dot algorithm*, http://www.ece3sat.com/blog/2018-01-25-how-to-slow-down-rotation-the-bdot-algorithm/.

[28] M Böttcher, et al. *Testing and validation of a B-dot algorithm for cubesat satellites*, 2016,

https://www.researchgate.net/publication/309593616_Testing_and_validation_of_a_B_algorithm_for_cubesat_satellites.

[29] Kenyon, Shaun. *STRaND-1: Use of a $500 Smartphone as the Central Avionics of a Nanosatellite*, 2011, https://www.researchgate.net/publication/277819389_STRaND-1_Use_of_a_500_Smartphone_as_the_Central_Avionics_of_a_Nanosatellite/figures?lo=1&utm_source=google&utm_medium=organic.

[30] Miller, Duncan. "Design optimization of the CADRE Magnetorquers." 2013, https://www.aerospades.com/uploads/3/7/3/2/37325123/cadre_torquers.pdf.

[31] SPIE. *Free-space Optics Beginning to Achieve Real-World Value*, https://spie.org/news/free-space-optics-beginning-to-achieve-real-world-value?SSO=1.

[32] BC Software. *¿Qué es C# y para qué sirve?*, https://bsw.es/que-es-c/.

[33] ERC Handbook. *Blue Pill (STM32F103C8T6)*, https://erc-bpgc.github.io/handbook/electronics/Development_Boards/STM32/.

[34] ProtoSupplies. *MPU-9250 3-Axis Accelerometer, Gyroscope & Magnetometer Sensor Module*, https://protosupplies.com/product/mpu-9250-3-axis-accel-gryo-mag-sensor-module/.

[35] Universidad de Sevilla. *Oscilaciones amortiguadas (GIE)*, http://laplace.us.es/wiki/index.php/Oscilaciones_amortiguadas_(GIE

[36] Tarifasgasyluz by Selectra. *Precio del kWh en el mercado regulado*, https://tarifasgasluz.com/comparador/precio-kwh.

APÉNDICE A: CÓDIGOS DE ARDUINO

Sensores.ino

```
/*JUAN PALOMARES MOYANO - 2019*/

/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/*~~~~~~~~~~LIBRERÍAS~~~~~~~~~~*/
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/

//Conexionados UART e I2C
#include "Wire.h"
#include "SoftwareSerial.h"

//IMU
#include "MPU6050_6Axis_MotionApps20.h" //Mediante esta biblioteca se pueden obtener datos del DMP
(Digital Motion Processor) de la IMU
#include "I2Cdev.h" //Mediante esta biblioteca se pueden obtener datos del DMP de la IMU
#include "helper_3dmath.h" //Mediante esta biblioteca se pueden realizar operaciones de cuaterniones

//GPS
#include "NMEAGPS.h"

/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/*~~~DEFINICIÓN DE VARIABLES~~~*/
/*~~~~~~~~Y CONSTANTES~~~~~~~~~~*/
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/

const float pi = 3.141592653;

//IMU
const int mpuAddress = 0x68;  // Se define la dirección de la IMU, puede ser 0x68 o 0x69
MPU6050 mpu(mpuAddress); //Se crea un objeto MPU6050 para poder extraer datos de la DMP
int fifoCount = 0, packetSize; //Se crea un contador de valores en el FIFO de la MPU y el tamaño del paquete
que obtendremos
byte fifoBuffer[42]; //Se crea el buffer que se utilizará para obtener los datos roll, pitch y yaw.
float roll, pitch, yaw, sqx, sqy, sqz, sqw, test; //Se crea tanto las variables de roll, pitch y yaw como las variables
"float"
                                        //intermedias necesarias para obtenerlas
Quaternion q; //Se crea el cuaternión necesario para obtener los valores de pitch, yaw y roll

//GPS
NMEAGPS GPSobject; //Se define un objeto NMEAGPS para poder acceder a los datos del GPS
gps_fix GPSdata; //Se define el "struct" donde se guardarán los datos del GPS
#define GPSerial Serial //Como los datos del GPS se obtendrán mediante el serial, se renombra el serial.
float LatLon[2],Vel,Alt,Sat; //Se definen las variables float donde se guardarán los datos obtenidos de GPSdata.

//Bluetooth
const int tx = 4; //Se define el pin digital 4 como el TX del Arduino en la conexión UART con el módulo
Bluetooth
const int rx = 3; //Se define el pin digital 3 como el RX del Arduino en la conexión UART con el módulo
Bluetooth
SoftwareSerial BT(rx, tx); //Se crea el puerto sere mediante software que utilizará el módulo Bluetooth para
conectarse con el Arduino
```

```
//RADIO
//Se crea una unión para pasar los datos "float" a "array de bytes" y así poderlos enviar por transmisión de radio
union {
  byte array[4];
  float FloatNumber;
} FloatByteTX;
//Se crea una unión para pasar los datos provenientes del buffer de recepción ("array de bytes") de la radio a float
union {
  byte array[4];
  float FloatNumber;
} FloatByteRX;
byte bufferRX[60], bufferTX[60];
float DatosOtroSat[8]; //Se define un array con todos los datos que se obtendrán del otro satélite.


/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/*~~~~~~~~~~~~SETUP~~~~~~~~~~~~*/
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/


void setup()
{

  //PUERTOS SERIE
  BT.begin(115200); //Se inicializa el puerto serie del bluetooth
  while (!BT); //Se espera hasta que el puerto del Bluetooth está inicializado
  GPSerial.begin(9600); //Se inicializa el puerto serie del GPS
  while (!GPSerial); //Se espera hasta que el puerto del GPS está inicializado
  Wire.begin(); //Se inicializa la librería Wire.h

  //IMU
  mpu.initialize(); //Inicialización de la imu
  mpu.dmpInitialize(); //Inicialización del DMP
  mpu.setDMPEnabled(true); //Habilitación del DMP
  packetSize = mpu.dmpGetFIFOPacketSize(); //Se obtiene el tamaño del paquete que obtendremos del DMP
(42)

  //Se introducen los offsets calculados mediante MPU6050_calibration
  mpu.setXAccelOffset(-2538);
  mpu.setYAccelOffset(-1997);
  mpu.setZAccelOffset(1888);
  mpu.setXGyroOffset(95);
  mpu.setYGyroOffset(34);
  mpu.setZGyroOffset(-45);

  //Se define el cuaternión inicial de actitud
  q.w = 1;
  q.x = 0;
  q.y = 0;
  q.z = 0;

  delay(100); //Se da un margen de tiempo para que todos los procesos se completen
}

/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/*~~~~~~~~~~~~LOOP~~~~~~~~~~~~~*/
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
```

```
void loop()
{
  IMU();
  GPS();
  RadioTX();
  RadioRX();
  Printer();
  delay(10); //Tiempo para relajar las conexiones entre componentes
}
```

04_RADIO_TX

```
/*JUAN PALOMARES MOYANO - 2019*/

/* ESTA FUNCIÓN CODIFICA Y ENVÍA LOS DATOS QUE SE QUIEREN ENVIAR POR ANTENA AL ARDUINO ESCLAVO*/

void RadioTX() {
  int j = 0; //Se inicializa un contador para que el buffer no se vaya sobreescribiendo conforme se codifican nuevos valores
  //Codificación del "roll"
  FloatByteTX.FloatNumber = roll; //El valor float de la unión de transmisión ahora será el "roll"
  for (int i = 0; i < sizeof(float); i++) { //Se guardan todos los bytes que codifican el "roll" en el buffer de transmisión
    bufferTX[j] = (FloatByteTX.array[i]); //Se escriben los bytes que codifican el "roll" en el buffer de transmisión
    j += 1; //El siguiente byte se guardará en la siguiente posición del "array" del "bufferTX"
  }

  //Codificación del "pitch"
  FloatByteTX.FloatNumber = pitch;
  for (int i = 0; i < sizeof(float); i++) {
    bufferTX[j] = (FloatByteTX.array[i]);
    j += 1;
  }

  //Codificación del "yaw"
  FloatByteTX.FloatNumber = yaw;
  for (int i = 0; i < sizeof(float); i++) {
    bufferTX[j] = (FloatByteTX.array[i]);
    j += 1;
  }

  //Codificación del número de satélites que aportan datos al módulo GPS
  FloatByteTX.FloatNumber = float(Sat);
  for (int i = 0; i < sizeof(float); i++) {
    bufferTX[j] = (FloatByteTX.array[i]);
    j += 1;
  }

  //Codificación de la latitud
  FloatByteTX.FloatNumber = LatLon[0];
  for (int i = 0; i < sizeof(float); i++) {
    bufferTX[j] = (FloatByteTX.array[i]);
    j += 1;
  }

  //Codificación de la longitud
  FloatByteTX.FloatNumber = LatLon[1];
  for (int i = 0; i < sizeof(float); i++) {
    bufferTX[j] = (FloatByteTX.array[i]);
    j += 1;
  }

  //Codificación de la velocidad
  FloatByteTX.FloatNumber = Vel;
  for (int i = 0; i < sizeof(float); i++) {
    bufferTX[j] = (FloatByteTX.array[i]);
    j += 1;
```

```
  }

  //Codificación de la altura
  FloatByteTX.FloatNumber = Alt;
  for (int i = 0; i < sizeof(float); i++) {
    bufferTX[j] = (FloatByteTX.array[i]);
    j += 1;
  }

  //Transmisión de los datos al Arduino esclavo
  Wire.beginTransmission (8); //Inicio de la transmisión con el Arduino esclavo con dirección 0x08
  for (int i = 0; i < j; i++) { //Este for recorre todo el buffer de transmisión desde la primera posición
hasta la última con valor
    Wire.write(bufferTX[i]); //Se envía el byte en cuestión
  }
  Wire.endTransmission (); //Finalización de la transmisión
}
```

Ard_nano.ino

```
/*JORDAN MORALES KRUEGER - 2021*/

/* FUNCIÓN PRINCIPAL DEL ARDUINO NANO COMO ESCLAVO PARA RECIBIR ÁNGULO CONSIGNA POR PUERTO SERIE.*/


#include <Wire.h>
//#include SoftwareSerial

float angulo = 0;  //valor angulo bueno
float angulo_float = 0;
String angulo_string = "";
int cont = 0;

//Bluetooth
//const int tx = 4; //Se define el pin digital 4 como el TX del Arduino en la conexión UART con el módulo Bluetooth
//const int rx = 3; //Se define el pin digital 3 como el RX del Arduino en la conexión UART con el módulo Bluetooth
//SoftwareSerial BT(rx, tx); //Se crea el puerto serie mediante software que utilizará el módulo Bluetooth para conectarse con el Arduino

union {
  byte array[4];
  float FloatNumber;
} FloatByteTX;

union {
  byte array[4];
  float FloatNumber;
} FloatByteRX;

byte bufferRX[60], bufferTX[60];
float DatosOtroSat[12]; //Se define un array con todos los datos que se obtendrán del maestro.

//Datos recibidos
float Ix = 0;
float Iy = 0;
float HTx = 0;


void setup() {

  //Puertos Serie
  Serial.begin(9600); // Abrimos puerto serie a 9600 bps
//  BT.begin(115200); //Se inicializa el puerto serie del bluetooth
//  while (!BT); //Se espera hasta que el puerto del Bluetooth está inicializado

  Wire.begin(8); //direccion 0x08 de i2c
  Wire.onRequest(requestEvent); //registrar evento de salida
  Wire.onReceive(receiveEvent); //registrar evento de entrada
}

void loop() {

  if (cont == 0) { //Solo vemos mensaje 1 vez
    delay(5000); //añadimos delay para poder visualizar
```

```
    Serial.println("Introduce angle: ");
    cont = 1;
  }
  angulo = consigna();//mientras haya datos que leer en el monitor serial
  Serial.print("HTx: ");
  Serial.println(HTx);
}

float consigna() { //recibe el angulo por el serial monitor

  while (Serial.available()) { //mientras haya datos que leer en el monitor serial
    char c = Serial.read();  //recibe datos caracter a caracter
    angulo_string += c; //crea un string a partir de los caracteres
    delay(2); //damos tiempo a leer
  }

  if (angulo_string.length() > 0) {
    angulo_float = angulo_string.toFloat(); //pasamos el string creado a partir de caracteres a float y guardamos la
variable
    angulo_string = ""; //vaciamos el string para poder recibir datos de nuevo
  }
  return (angulo_float);
}

void receiveEvent() { //el maestro envía datos
  Receive();
}
void requestEvent() { //el maestro pide datos
  encoder();
}
```

01_Encoder.ino

```
/*JORDAN MORALES KRUEGER - 2021*/

/* ESTA FUNCIÓN CODIFICA LOS DATOS A ENVIAR A LA BLUEPILL.*/


void encoder() { //Codifica el angulo consigna en un formato apto para envío por i2c
  int j = 0; //Se inicializa un contador para que el buffer no se vaya sobreescribiendo conforme se codifican
nuevos valores
  //Codificación del "angulo"
  FloatByteTX.FloatNumber = angulo; //El valor float de la unión de transmisión ahora será el "angulo"
  for (int i = 0; i < sizeof(float); i++) { //Se guardan todos los bytes que codifican el "angulo" en el buffer de
transmisión
    bufferTX[j] = (FloatByteTX.array[i]); //Se escriben los bytes que codifican el "angulo" en el buffer de
transmisión
    j += 1; //El siguiente byte se guardará en la siguiente posición del "array" del "bufferTX"
  }

  for (int i = 0; i < j; i++) { //Este for recorre todo el buffer de transmisión desde la primera posición hasta la
última con valor
    Wire.write(bufferTX[i]); //Se envía el byte en cuestión
  }
}
```

02_Receive.ino

```
/*JORDAN MORALES KRUEGER - 2021*/

/* ESTA FUNCIÓN DECODIFICA LOS DATOS RECIBIDOS DE LA BLUEPILL.*/


void Receive() {

 int i=0;
  while (Wire.available()) { // mientras haya datos que leer por i2c
    bufferRX[i] = Wire.read(); //Los bytes que se obtienen se guardan en el buffer de recepción
    i = i + 1;
  }
  //Decodificación de los datos recibidos
  for (int j = 0; j < 3; j++) { //Mediante este "for" se reciben 3 "float" que se esperan obtener
    for (int i = 0; i < sizeof(float); i++) {  //sizeof(float) devuelve el tamaño de un float (4 bytes)
      FloatByteRX.array[i] = bufferRX[i + j * 4]; //Los 4 bytes que representan 1 float se guardan en la unión de
recepción
    }
    DatosOtroSat[j] = FloatByteRX.FloatNumber; //Se convierten los 4 bytes a "float" y se guarda el valor
  }

  Ix = DatosOtroSat[0]; //Intensidad en X
  Iy = DatosOtroSat[1]; //Intensidad en y
  HTx = DatosOtroSat[2]; //Campo magnético en X

}
```

Bluepill_BT.ino

```
/*JORDAN MORALES KRUEGER - 2021*/

/*FUNCIÓN PRINCIPAL DE LA BLUEPILL.*/


/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/*~~~~~~~~~~~~LIBRERÍAS~~~~~~~~~~~~*/
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/

#include <math.h>

//Driver
#include <Arduino.h>
#include <stdint.h>
#include <SCMD.h>
#include <SCMD_config.h> //Contains #defines for common SCMD register names and values

//I2C
#include <Wire.h>

//IMU
#include <MPU9250.h>


/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/*~~~DEFINICIÓN DE VARIABLES~~~*/
/*~~~~~~~~Y CONSTANTES~~~~~~~~~~~~*/
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/

//Drivers
int sentido = 1; //sentido por defecto
SCMD MT; //creamos un objeto Magnetorquer que controla los 2 MT
SCMD RW; //Driver rueda inercia


//IMU
MPU9250 IMU(SPI,PA4);
int status;

unsigned long now = 0;              //Variables de integracion
unsigned long last_sum = 0;
unsigned long dT = 0;

float BTx=0, BTy=0, BTz=0, Ax=0, Ay=0, Az=0, Vx=0, Vy=0, Vz=0;
float Temp;
float Gpitch = 0, Groll = 0, Gyaw = 0;  //Variables geometricas
float Atotal = 0, Apitch = 0, Aroll = 0;
float Mpitch = 0, Mroll = 0;
float M_x_eq = 0, M_y_eq = 0;
float Mag_x_damp = 0, Mag_y_damp = 0;
float Heading = 0, Heading_damp = 0;
float G_pitch_output = 0, G_roll_output = 0;
bool Gyro_sync = false;


//Conversion de radianes a grados y viceversa
float rad_to_deg = 57.29577951;
```

```
float deg_to_rad = 0.01745329;


//Variables generales
float error = 0; //angle_setpoint-angle_init
float angulo = 0; //angulo introducido
float angulo_sp; //angulo setpoint
float x = 0; //variable de prueba

/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/*~~~~~~~~~~~~~~SETUP~~~~~~~~~~~~~~*/
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/

void setup()
{

  //Puertos Serie
  Serial.begin(9600);  // inicializamos puerto serie
  Wire.begin();        // inicializamos el bus i2c
  Serial1.begin(115200); //inicializamos el puerto serie virtual

//****************************************IMU*****************************************//


  while(!Serial) {}
  // start communication with IMU

  status = IMU.begin(); //Si no responde la IMU se queda en el bucle
  if (status < 0) {
    while(1) {}
  }

  IMU.setGyroRange(IMU.GYRO_RANGE_500DPS); //Se cambia el rango de medicion del giroscopio, lo que
hace mas preciso el sensor


  //*********************************************************************************//


  //**********************************Magnetoquer**************************************//

  //  Configuracion magnetorquers
  RW.settings.commInterface = I2C_MODE;
  MT.settings.commInterface = I2C_MODE;

  RW.settings.I2CAddress = 0x5D; //configuramos patron "1000" en el driver para direccion 0x5D
  MT.settings.I2CAddress = 0x61; //configuramos patron "1100" en el driver para direccion 0x61

  delay(500);


  //  Inicializamos el driver y habilitamos los outputs

  while ( MT.begin() != 0xA9 )
  {
    delay(5000);
  }

  while (MT.ready() == false );
```

```
  while (MT.busy());
  MT.enable();


  while ( RW.begin() != 0xA9 )
  {
    delay(5000);
  }

  while (RW.ready() == false );
  while ( RW.busy() );
  RW.enable();

  //calibrate_mag();       //Esta funcion calibra el magnetometro. Tarda bastante, unos 2 minutos, durante los cuales es necesario
                           //estar "dibujando" un 8 con la placa en la mano. Los valores que se obtienen de una vez para otra son
                           //considerablemente dispares, aunque con los intentos se ve que confluyen. Se recomienda realizar el proceso
                           //mas de una vez para observar dicha confluencia y calcular unos valores medios.
                           //Cuando ya se tengan los valores deseados, se comenta y guardan los valores obtenidos en las siguientes lineas(siempre descomentadas).

    IMU.setMagCalX(-0.94,1.24); //La primera columna corresponede al bias magnetico y la segunda al factor de escala, mencionados en la funcion calibrate_mag();
    IMU.setMagCalY(-7.18,0.90);
    IMU.setMagCalZ(6.90,0.93);
}
//*****************************************************************************************//


/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/*~~~~~~~~~~~~~~LOOP~~~~~~~~~~~~~~~~~~*/
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/

void loop()
{
  bluetooth();
  read_IMU();
  drive_CubeSat();
  delay(50);
}
```

02_IMU.ino

```
void read_IMU() { //esta funcion lee los valores "raw" y llama a la funcion euler_heading

  IMU.readSensor();

  Ax = IMU.getAccelX_mss(); //Aceleración en eje X, Y, Z
  Ay = IMU.getAccelY_mss();
  Az = IMU.getAccelZ_mss();

  Vx = IMU.getGyroX_rads(); //Velocidad angular en X, Y, Z
  Vy = IMU.getGyroY_rads();
  Vz = IMU.getGyroZ_rads();
```

```
  BTx = IMU.getMagX_uT(); //Campo magnético en X, Y, Z
  BTy = IMU.getMagY_uT();
  BTz = IMU.getMagZ_uT();

  Temp = IMU.getTemperature_C(); //Temperatura

  now = millis();           //Milisegundos transcurridos desde el principio de la compilacion.
  dT = now - last_sum;      //Variable de integracion numerica. Calcula el tiempo transcurrido entre este loop y el
anterior.
  last_sum = now;
  euler_heading(dT);
}
```

03_drive_CubeSat.ino

```
void drive_CubeSat(){

  MT.setDrive( 0,sentido, angulo); //MT Y
  MT.setDrive( 1,sentido, angulo); //MT X

}
```

04_getValue.ino

```
String getValue(String data, char separator, int index) //Función para decodificar datos de la interfaz
  {
    int found = 0;
    int strIndex[] = {0, -1};
    int maxIndex = data.length() - 1; //el índice máximo se encuentra en el último valor sin contar el símbolo "$"
de inicio
    for (int i = 0; i <= maxIndex && found <= index; i++)
    {
      if (data.charAt(i) == separator || i == maxIndex)
      {
        found++;
        strIndex[0] = strIndex[1] + 1;
        strIndex[1] = (i == maxIndex) ? i + 1 : i;
      }
    }
return found > index ? data.substring(strIndex[0], strIndex[1]) : "";
  }
```

05_euler_heading.ino

```
/*Andres Gómez, Miguel Reurer - 2021 */

/*FUNCIÓN PARA EXTRAER PITCH, ROLL Y YAW.*/

void euler_heading(unsigned long dT) {

  IMU.readSensor();

  IMU.readSensor(); //Lectura del sensor (Valores RAW).
```

```
  Gpitch += dT*(IMU.getGyroY_rads())*rad_to_deg*0.001;    //0,001 es la conversion del resultado de la funcion milis()
  Groll += dT*(IMU.getGyroX_rads())*rad_to_deg*0.001;     //de milisegundos a segundos.
  Gyaw += dT*(IMU.getGyroZ_rads())*rad_to_deg*0.001;

  Gpitch = Gpitch + Groll*sin(dT*(IMU.getGyroZ_rads())*0.001);   //Compensa el problema del gimbal lock. Explicado en la memoria.
  Groll = Groll - Gpitch*sin(dT*(IMU.getGyroZ_rads())*0.001);

  Atotal = sqrt((IMU.getAccelX_mss() * IMU.getAccelX_mss()) + (IMU.getAccelY_mss() * IMU.getAccelY_mss()) + (IMU.getAccelZ_mss() * IMU.getAccelZ_mss()));     //Calculo de la aceleracion total.
  Apitch = asin((float)IMU.getAccelX_mss() / Atotal) * rad_to_deg;
//Cabeceo mediante los valores del acelerometro.
  Aroll = asin((float)IMU.getAccelY_mss() / Atotal) * rad_to_deg;
//Alabeo mediante los valores del acelerometro.

  //VALORES DE CALIBRACION DEL ACELEROMETRO

  Apitch -= -11.18;  //Valores de calibracion del acelerometro. No hay ScaleFactor, pero funcionan considerablemente bien. Para obtenerlos es necesario ponerlos a 0,0.
  Aroll -= -0.63;  //Tambien es necesario descomentar los tres prints que suceden a estas lineas y si se quiere, para limpiar la lectura des del monitor serie, comentar
                   //el resto de prints de esta funcion. Una vez obtenidos, guardar los valores y sustituirlos por el 0,0. Al repetir el programa con los nuevos valores
                   //deberiamos ver como los valores se aproximan ahora al cero. Si no convencen, repetir el proceso (poner a 0,0 y leer). Una vez finalizado el proceso,
                   //volver a dejar como estaba: (Los prints comentados y el resto descomentados y las variables, cambiadas, sin comentar).

//    Serial.print(Apitch,6);
//    Serial.print("\t ");
//    Serial.println(Aroll,6);


  if (Gyro_sync)  //Proceso que solo se realiza al principio, para sincronizar el filtro en un comienzo
  {
    Gpitch = Gpitch * 0.98 + Apitch * 0.02;       //Filtro complementario. Se ajusta la lectura del giroscopio con la del acelerometro.
    Groll = Groll * 0.98 + Aroll * 0.02;          //Algunas fuentes utilizan 0,9996 y 0,0004. Tal como esta se obtienen mejores resultados.
  }
  else
  {
    Gpitch = Apitch;          //Solo se entra en este else{} en el primer loop() (ya que la variable Gyro_sync se incia como "false".
    Groll = Aroll;            //Sincroniza las lecturas del acelerometro y el giroscopio.
    Gyro_sync = true;
  }

  G_pitch_output = G_pitch_output * 0.9 + Gpitch * 0.1;   //Amortigua los picos en la lectura final del girocopio.
  G_roll_output = G_roll_output * 0.9 + Groll * 0.1;

  Mpitch = -G_roll_output * deg_to_rad;     //Conversion al sistema de referencia del magnetometro y conversion a radianes (unidades Arduino).
  Mroll = G_pitch_output * deg_to_rad;

  M_x_eq = IMU.getMagX_uT() * cos(Mpitch) + IMU.getMagY_uT() * sin(Mroll) * sin(Mpitch) - IMU.getMagZ_uT() * cos(Mroll) * sin(Mpitch); //Funciones mencionadas, proyectan el rumbo en el plano horizontal.
```

```
  M_y_eq = IMU.getMagY_uT() * cos(Mroll) + IMU.getMagZ_uT() * sin(Mroll);

  Mag_x_damp = Mag_x_damp * 0.9 + M_x_eq * 0.1;  //Amortigua los valores (Igual que G_pitch_output) para reducir los picos de lectura.
  Mag_y_damp = Mag_y_damp * 0.9 + M_y_eq * 0.1;

  Heading = -atan2(Mag_y_damp, Mag_x_damp) * rad_to_deg;    //Como se comenta en la memoria, se ha cambiado alguna cosa (x por y) y el signo del principio.
                                                            //Originalmente, cuando habia x/y, la lectura estaba desplazada 90 grados en sentido horario y sin el negativo del principio,
                                                            //lo que hacia que la lectura sumase los angulos en sentido antihorario en vez de horario.
  Heading_damp = Heading_damp * 0.9 + Heading * 0.1;

  //Heading += Declination; //Esta linea se puede descomentar si se quiere ser preciso del todo. Anade la declinacion del lugar geografico en el que nos encontramos.
                            //Se puede obtener en https://www.ign.es/web/gmt-declinacion-magnetica.

  if (Heading < 0) Heading += 360;
  if (Heading >= 360) Heading -= 360; //Estas dos lineas contienen el valor del heading entre 0 y 360 grados.

  Temp = IMU.getTemperature_C();

  now = millis();           //Milisegundos transcurridos desde el principio de la compilacion.
  dT = now - last_sum;      //Variable de integracion numerica. Calcula el tiempo transcurrido entre este loop y el anterior.
  last_sum = now;
}
```

06_calibrate_mag.ino

```
/*Andres Gómez, Miguel Reurer - 2021 */

/*FUNCIÓN PARA CALIBRAR MAGNETÓMETRO.*/

void calibrate_mag() {

    IMU.calibrateMag();

    Serial.println("Done");

    Serial.print(IMU.getMagBiasX_uT());  //Valores del bias magnetico
    Serial.print(", ");
    Serial.print(IMU.getMagBiasY_uT());
    Serial.print(", ");
    Serial.println(IMU.getMagBiasZ_uT());

    Serial.print(IMU.getMagScaleFactorX()); //Valores del factor de escala
    Serial.print(", ");
    Serial.print(IMU.getMagScaleFactorY());
    Serial.print(", ");
    Serial.println(IMU.getMagScaleFactorZ());

}
```

07_Send_Data.ino

```
/*JORDAN MORALES KRUEGER - 2021 */

/*FUNCIÓN PARA ENVIAR DATOS POR BLUETOOTH A LA INTERFAZ. SI SE QUIERE ENVIAR
DATOS POR PUERTO SERIE CAMBIAR "Serial1" por "Serial" */

void bluetooth() { //Este subprograma intercambia información a la interfaz

  if (Serial.available() > 0) //Mientras haya datos disponibles en el puerto serie
  {
    String datos_serie = Serial.readString(); // Se lee un string donde los datos vienen con el formato
dato1;dato2;...;datoN
    String modo_st = getValue(datos_serie, ';', 0); //Se guardan los datos en las variables de tipo String
    String angulo_st = getValue (datos_serie, ';', 1); //ídem
    modo = modo_st.toFloat(); //Se pasa el valor de String a Float
    angulo = angulo_st.toFloat();
    angulo_sp = angulo + Heading; // El setpoint es la suma del ángulo medido en el momento de la instrucción
(heading) más el ángulo introducido
  }

  if (angulo < 0)
  {
    x = -angulo;
    sentido = 0;
  }
  else
  {
    sentido = 1;
    x = angulo;
  }

  error = angulo_sp - Heading;
  String s = '$' + String(Vx) + ';' + String(Vy) + ';' + String(Vz) + ';' + String(Ax) + ';' + String(Ay) + ';' +
String(Az) + ';' + String(BTx) + ';' + String(BTy) + ';' + String(BTz) + ';' + String(G_roll_output) + ';' +
String(G_pitch_output)+ ';'+ String(Heading) + ';' + String(Temp) + ';' + String(error);
  Serial.println(s);
  delay(50);

}
```

Driver_Hbridge.ino

```
#include <Wire.h>
#include <SCMD.h>
#include <SCMD_config.h>
//Librería SCMD: https://github.com/sparkfun/SparkFun_Serial_Controlled_Motor_Driver_Arduino_Library


SCMD MT; //creamos un objeto Magnetorquer que controla los 2 MT
void setup() {
  Serial.begin(9600);  // inicializamos puerto serie
  Wire.begin();        // inicializamos el bus i2c

  //  Configuracion magnetorquers
  MT.settings.commInterface = I2C_MODE;
  MT.settings.I2CAddress = 0x5D; // Configuramos patrón "1000" en el driver para dirección 0x5D
```

```
  delay(500);

  // Inicializamos driver y habilitamos los outputs
  Serial.print("Magnetorquer driver reading ID = 0x");
  Serial.println(MT.begin(), HEX);

  delay(500);

  while ( MT.begin() != 0xA9 ) // Si no devuelve la dirección 0xA9 la comunicación no ha sido correcta
  {
    Serial.println( "ID Mismatch in magnetorquer driver. Trying again..." );
    delay(5000);
  }

  Serial.println( "ID matches 0xA9" );

  Serial.println("Waiting for magnetorquer configuration to complete...");
  while (MT.ready() == false ); //
  Serial.println("Magnetorquer configuration done.");

  while ( MT.busy() );
  MT.enable(); //Habilitamos el objeto MT

}

void loop() {

 MT.setDrive( 0,0, 255); //(Objeto, Sentido, Velocidad)

}
```

APÉNDICE B: CÓDIGO VISUAL STUDIO 2019

```
// Jordan Morales Krueger (2021)

using System;
using System.Drawing;
using System.Threading;
using System.Windows.Forms;
using System.Runtime.InteropServices;
using System.Windows.Forms.DataVisualization.Charting;

namespace ADCS_Control
{
   public partial class ADCS_control : Form
   {
            bool puerto_isopen = false;
            string angulo_string; //variable angulo en formato String
            double angulo;
       int modo = 0;
            System.IO.Ports.SerialPort puerto; //variable puerto
            String[] listado_puerto = System.IO.Ports.SerialPort.GetPortNames(); //obtenemos listado de puertos
            string datos_puerto; //variable donde se reciben los datos del arduino
            double pitch = 0.0, yaw = 0.0, roll = 0.0, ax = 0.0, ay = 0.0, az = 0.0, bx = 0.0, by = 0.0, bz = 0.0, vx =
0.0, vy = 0.0, vz = 0.0, Temp = 0.0, error = 0.0; //variables de entrada
            double tiempo = 0.0; //variable tiempo
            string Xg;
            string Yg;
            int x = 0;
            int y = 0;
            int sp_detumbling = 0; //SetPoint es 0 rad/s para grafica
            System.Drawing.Rectangle r1 = new System.Drawing.Rectangle();



            private void buttonSubmit_Click(object sender, EventArgs e) //boton para enviar angulo a BluePill
            {
            angulo_string = textBox1.Text;
            angulo = Convert.ToDouble(angulo_string);

            if (puerto_isopen == true)
            {

                  puerto.Write(modo + ";" + angulo_string);

            }
            else MessageBox.Show("No connection found.", "COM Port Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }


            private void button4_Click(object sender, EventArgs e) //boton detumbling
            {
            modo = 1;

          labelDetumblingInactive.BringToFront();
```

```
        labelRotationInactive.BringToFront();
          buttonDetumbling.BackColor = Color.WhiteSmoke;

          labelDetumblingActive.SendToBack();
          labelRotationActive.SendToBack();
          buttonRotation.BackColor = Color.WhiteSmoke;

          if (puerto_isopen == true)
          {

           labelNadirActive.BringToFront();
           labelNadirInactive.SendToBack();
          buttonNadir.BackColor = Color.Silver;

          puerto.Write(modo + ";" + angulo_string);

          }
          else MessageBox.Show("No connection found.", "COM Port Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);

          }

          private void button5_Click(object sender, EventArgs e) //boton Nadir
          {
          modo = 2;

          labelNadirInactive.BringToFront();
        labelRotationInactive.BringToFront();
          buttonNadir.BackColor = Color.WhiteSmoke;

          labelNadirActive.SendToBack();
          labelRotationActive.SendToBack();
          buttonRotation.BackColor = Color.WhiteSmoke;

          if (puerto_isopen == true)
          {

           labelDetumblingActive.BringToFront();
           labelDetumblingInactive.SendToBack();
          buttonDetumbling.BackColor = Color.Silver;

          puerto.Write(modo + ";" + angulo_string);

          }
          else MessageBox.Show("No connection found.", "COM Port Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
      }

          private void button6_Click(object sender, EventArgs e) //boton orientacion
          {
          modo = 3;

        labelDetumblingInactive.BringToFront();
          labelNadirInactive.BringToFront();
          buttonNadir.BackColor = Color.WhiteSmoke;
          labelDetumblingActive.SendToBack();
          labelNadirActive.SendToBack();
          buttonDetumbling.BackColor = Color.WhiteSmoke;

          if (puerto_isopen == true)
```

```
            {

            labelRotationActive.BringToFront();
              labelRotationInactive.SendToBack();
            buttonRotation.BackColor = Color.Silver;

            puerto.Write(modo + ";" + angulo_string);

            }
            else MessageBox.Show("No connection found.", "COM Port Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }

            private void buttonConnect_Click(object sender, EventArgs e) //boton de conexion
            {
            try
            {
            serial();
              puerto.Open(); //abrimos puerto
            puerto_isopen = true;
            pictureBoxDisconnected.Visible = false; //Imagen de desconectado desaparece
            pictureBoxConnected.Visible = true; //Imagen de conectado aparece
            MessageBox.Show("Connection to port succesful.", "Success!", MessageBoxButtons.OK);
            }
            catch
            {
            MessageBox.Show("Connection unsuccesful." + System.Environment.NewLine + "- Check selected
port" + System.Environment.NewLine + "- Connection already established", "COM Port Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            }

            private void buttonDisconnect_Click(object sender, EventArgs e) //boton desconectar
            {
            try
            {
            puerto.Close(); //cerramos el puerto
            timer1.Stop(); //paramos el timer
            puerto_isopen = false;
            pictureBoxDisconnected.Visible = true; //Imagen de desconectado aparece
            pictureBoxConnected.Visible = false; //Imagen de conectado desaparece
            MessageBox.Show("Port disconnected.", "COM Port Info", MessageBoxButtons.OK);
            }

            catch
            {
            MessageBox.Show("No connection found.", "COM Port Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
            }

            //Tooltips Gráficas. Los tooltips sirven para visualizar un valor exacto en la gráfica cuando ponemos el
cursor encima de la gráfica.
            //Código extraído de: https://github.com/Picaio/visual2017_arduino

            //**************TOOLTIP VELOCIDAD ANGULAR*****************//
            System.Drawing.Point? prevPosition = null;
            ToolTip tooltip = new ToolTip();
            private void chartVelAng_MouseDown(object sender, MouseEventArgs e)
            {
```

```
            chartVelAng.Invalidate();
            r1.X = x;
            r1.Y = y;
            r1.Width = 3;
            r1.Height = 3;
            }

            private void chartVelAng_MouseMove(object sender, MouseEventArgs e)
            {
            var pos1 = e.Location;
            if (prevPosition.HasValue && pos1 == prevPosition.Value)
            return;
            tooltip.RemoveAll();
            prevPosition = pos1;
            var results1 = chartVelAng.HitTest(pos1.X, pos1.Y, false,
                                    ChartElementType.DataPoint);
            foreach (var result1 in results1)
            {
            if (result1.ChartElementType == ChartElementType.DataPoint)
            {
                    var prop1 = result1.Object as DataPoint;
                  if (prop1 != null)
                    {
                    var pointXPixel = result1.ChartArea.AxisX.ValueToPixelPosition(prop1.XValue);
                    var pointYPixel = result1.ChartArea.AxisY.ValueToPixelPosition(prop1.YValues[0]);

                    // check if the cursor is really close to the point (2 pixels around)

                    tooltip.Show("Time=" + prop1.XValue + "s"+ ", Deg=" + prop1.YValues[0] + "°",
this.chartVelAng,
                                    pos1.X, pos1.Y - 15);

                    }
            }
            }
            }

            private void chartVelAng_PostPaint(object sender,
System.Windows.Forms.DataVisualization.Charting.ChartPaintEventArgs e)
            {
            System.Drawing.Font drawFont = new System.Drawing.Font("Verdana", 8);
            System.Drawing.SolidBrush drawBrush = new
System.Drawing.SolidBrush(System.Drawing.Color.Red);

            System.Drawing.StringFormat drawFormat = new System.Drawing.StringFormat();
            if (!(string.IsNullOrEmpty(Xg) && string.IsNullOrEmpty(Yg)))
            {
             e.ChartGraphics.Graphics.DrawString("Time=" + Xg + "\n" + "°=" + Yg, drawFont, drawBrush, x + 5,
y - 2);
            e.ChartGraphics.Graphics.DrawRectangle(new Pen(Color.Red, 3), r1);
            }
            }

            //**************TOOLTIP ACELERACION*****************//
            private void chartAcceleration_PostPaint(object sender, ChartPaintEventArgs e)
            {
            System.Drawing.Font drawFont = new System.Drawing.Font("Verdana", 8);
            System.Drawing.SolidBrush drawBrush = new
System.Drawing.SolidBrush(System.Drawing.Color.Red);
```

```
            System.Drawing.StringFormat drawFormat = new System.Drawing.StringFormat();
            if (!(string.IsNullOrEmpty(Xg) && string.IsNullOrEmpty(Yg)))
            {
             e.ChartGraphics.Graphics.DrawString("Time=" + Xg + "\n" + "Accel=" + Yg, drawFont, drawBrush,
x + 5, y - 2);
            e.ChartGraphics.Graphics.DrawRectangle(new Pen(Color.Red, 3), r1);
            }
            }

            private void chartAcceleration_MouseDown(object sender, MouseEventArgs e)
            {
            chartVelAng.Invalidate();
            r1.X = x;
            r1.Y = y;
            r1.Width = 3;
            r1.Height = 3;
            }

            private void chartAcceleration_MouseMove(object sender, MouseEventArgs e)
            {
            var pos1 = e.Location;
            if (prevPosition.HasValue && pos1 == prevPosition.Value)
            return;
            tooltip.RemoveAll();
            prevPosition = pos1;
            var results1 = chartAcceleration.HitTest(pos1.X, pos1.Y, false,
                                ChartElementType.DataPoint);
            foreach (var result1 in results1)
            {
            if (result1.ChartElementType == ChartElementType.DataPoint)
            {

 var prop1 = result1.Object as DataPoint;
                        if (prop1 != null)
                        {
                        var pointXPixel = result1.ChartArea.AxisX.ValueToPixelPosition(prop1.XValue);
                        var pointYPixel = result1.ChartArea.AxisY.ValueToPixelPosition(prop1.YValues[0]);

                        // check if the cursor is really close to the point (2 pixels around)

                        tooltip.Show("Time=" + prop1.XValue + "s" + ", Accel=" + prop1.YValues[0] + "m/ss",
this.chartAcceleration,
                                    pos1.X, pos1.Y - 15);

                        }
            }
            }
            }

            //**************TOOLTIP CAMPO MAGNETICO*****************//
            private void chartMagneticField_MouseDown(object sender, MouseEventArgs e)
            {
            chartVelAng.Invalidate();
            r1.X = x;
            r1.Y = y;
            r1.Width = 3;
            r1.Height = 3;
```

```
        }

        private void chartMagneticField_MouseMove(object sender, MouseEventArgs e)
        {
        var pos1 = e.Location;
        if (prevPosition.HasValue && pos1 == prevPosition.Value)
        return;
        tooltip.RemoveAll();
        prevPosition = pos1;
        var results1 = chartMagneticField.HitTest(pos1.X, pos1.Y, false,
                                    ChartElementType.DataPoint);
        foreach (var result1 in results1)
        {
        if (result1.ChartElementType == ChartElementType.DataPoint)
        {
                    var prop1 = result1.Object as DataPoint;
            if (prop1 != null)
                    {
                    var pointXPixel = result1.ChartArea.AxisX.ValueToPixelPosition(prop1.XValue);
                    var pointYPixel = result1.ChartArea.AxisY.ValueToPixelPosition(prop1.YValues[0]);

                    // check if the cursor is really close to the point (2 pixels around)

                    tooltip.Show("Time=" + prop1.XValue + "s" + ", B=" + prop1.YValues[0] + "µT",
this.chartMagneticField,
                                pos1.X, pos1.Y - 15);

                    }
        }
        }
        }

        private void chartMagneticField_PostPaint(object sender, ChartPaintEventArgs e)
        {
        System.Drawing.Font drawFont = new System.Drawing.Font("Verdana", 8);
        System.Drawing.SolidBrush drawBrush = new
System.Drawing.SolidBrush(System.Drawing.Color.Red);

        System.Drawing.StringFormat drawFormat = new System.Drawing.StringFormat();
        if (!(string.IsNullOrEmpty(Xg) && string.IsNullOrEmpty(Yg)))
        {
         e.ChartGraphics.Graphics.DrawString("Time=" + Xg + "\n" + "B=" + Yg, drawFont, drawBrush, x +
5, y - 2);
         e.ChartGraphics.Graphics.DrawRectangle(new Pen(Color.Red, 3), r1);
        }
        }

        //**************TOOLTIP ANGULO*****************//
        private void chartAngle_MouseDown(object sender, MouseEventArgs e)
        {
        chartVelAng.Invalidate();
        r1.X = x;
```

```
            r1.Y = y;
            r1.Width = 3;
            r1.Height = 3;
            }

            private void chartAngle_MouseMove(object sender, MouseEventArgs e)
            {
            var pos1 = e.Location;
            if (prevPosition.HasValue && pos1 == prevPosition.Value)
            return;
            tooltip.RemoveAll();
            prevPosition = pos1;
            var results1 = chartAngle.HitTest(pos1.X, pos1.Y, false,
                                    ChartElementType.DataPoint);
            foreach (var result1 in results1)
            {
            if (result1.ChartElementType == ChartElementType.DataPoint)
            {
                        var prop1 = result1.Object as DataPoint;
                        if (prop1 != null)
                        {
                        var pointXPixel = result1.ChartArea.AxisX.ValueToPixelPosition(prop1.XValue);
                        var pointYPixel = result1.ChartArea.AxisY.ValueToPixelPosition(prop1.YValues[0]);

                        // check if the cursor is really close to the point (2 pixels around)

                        tooltip.Show("Time=" + prop1.XValue + "s" + ", Angle=" + prop1.YValues[0] + "°",
this.chartAngle,
                                    pos1.X, pos1.Y - 15);

                        }
            }
            }
            }

            private void chartAngle_PostPaint(object sender, ChartPaintEventArgs e)
            {
            System.Drawing.Font drawFont = new System.Drawing.Font("Verdana", 8);
            System.Drawing.SolidBrush drawBrush = new
System.Drawing.SolidBrush(System.Drawing.Color.Red);

            System.Drawing.StringFormat drawFormat = new System.Drawing.StringFormat();
            if (!(string.IsNullOrEmpty(Xg) && string.IsNullOrEmpty(Yg)))
            {
              e.ChartGraphics.Graphics.DrawString("Time=" + Xg + "\n" + "Angle=" + Yg, drawFont, drawBrush,
x + 5, y - 2);
                e.ChartGraphics.Graphics.DrawRectangle(new Pen(Color.Red, 3), r1);
            }
            }

            //**************TOOLTIP NADIR*****************//
```

```
        private void chartNadir_MouseDown(object sender, MouseEventArgs e)
        {
        chartVelAng.Invalidate();
        r1.X = x;
        r1.Y = y;
        r1.Width = 3;
        r1.Height = 3;
        }

        private void chartNadir_MouseMove(object sender, MouseEventArgs e)
        {
        var pos1 = e.Location;
        if (prevPosition.HasValue && pos1 == prevPosition.Value)
        return;
        tooltip.RemoveAll();
        prevPosition = pos1;
        var results1 = chartAngle.HitTest(pos1.X, pos1.Y, false,
                                    ChartElementType.DataPoint);
        foreach (var result1 in results1)
        {
        if (result1.ChartElementType == ChartElementType.DataPoint)
        {
                var prop1 = result1.Object as DataPoint;
                if (prop1 != null)
                {
                var pointXPixel = result1.ChartArea.AxisX.ValueToPixelPosition(prop1.XValue);
                var pointYPixel = result1.ChartArea.AxisY.ValueToPixelPosition(prop1.YValues[0]);
              // check if the cursor is really close to the point (2 pixels around)

                tooltip.Show("Time=" + prop1.XValue + "s" + ", Angle=" + prop1.YValues[0] + "°",
this.chartAngle,
                        pos1.X, pos1.Y - 15);

                }
        }
        }
        }

        private void chartNadir_PostPaint(object sender, ChartPaintEventArgs e)
        {
        System.Drawing.Font drawFont = new System.Drawing.Font("Verdana", 8);
        System.Drawing.SolidBrush drawBrush = new
System.Drawing.SolidBrush(System.Drawing.Color.Red);

        System.Drawing.StringFormat drawFormat = new System.Drawing.StringFormat();
        if (!(string.IsNullOrEmpty(Xg) && string.IsNullOrEmpty(Yg)))
        {
         e.ChartGraphics.Graphics.DrawString("Time=" + Xg + "\n" + "Angle=" + Yg, drawFont, drawBrush,
x + 5, y - 2);
         e.ChartGraphics.Graphics.DrawRectangle(new Pen(Color.Red, 3), r1);
        }
        }

        //**************TOOLTIP ERROR*****************//

        private void chartError_MouseDown(object sender, MouseEventArgs e)
```

```
        {
        chartError.Invalidate();
        r1.X = x;
        r1.Y = y;
        r1.Width = 3;
        r1.Height = 3;
        }


        private void chartError_MouseMove(object sender, MouseEventArgs e)
        {
        var pos1 = e.Location;
        if (prevPosition.HasValue && pos1 == prevPosition.Value)
        return;
        tooltip.RemoveAll();
        prevPosition = pos1;
        var results1 = chartError.HitTest(pos1.X, pos1.Y, false,
                                ChartElementType.DataPoint);
        foreach (var result1 in results1)
        {
        if (result1.ChartElementType == ChartElementType.DataPoint)
           {
                var prop1 = result1.Object as DataPoint;
                if (prop1 != null)
                {
                var pointXPixel = result1.ChartArea.AxisX.ValueToPixelPosition(prop1.XValue);
                var pointYPixel = result1.ChartArea.AxisY.ValueToPixelPosition(prop1.YValues[0]);

                // check if the cursor is really close to the point (2 pixels around)

                tooltip.Show("Time=" + prop1.XValue + "s" + ", Error=" + prop1.YValues[0] + "°",
this.chartError,
                                pos1.X, pos1.Y - 15);

                }
        }
        }
        }

        private void chartError_PostPaint(object sender, ChartPaintEventArgs e)
        {
        System.Drawing.Font drawFont = new System.Drawing.Font("Verdana", 8);
        System.Drawing.SolidBrush drawBrush = new
System.Drawing.SolidBrush(System.Drawing.Color.Red);

        System.Drawing.StringFormat drawFormat = new System.Drawing.StringFormat();
        if (!(string.IsNullOrEmpty(Xg) && string.IsNullOrEmpty(Yg)))
        {
            e.ChartGraphics.Graphics.DrawString("Time=" + Xg + "\n" + "=" + Yg, drawFont, drawBrush, x +
5, y - 2);
            e.ChartGraphics.Graphics.DrawRectangle(new Pen(Color.Red, 3), r1);
        }
        }


        //*****************TOOLTIP DETUMBLING**********************//

        private void chartDetumbling_MouseDown(object sender, MouseEventArgs e)
        {
        chartError.Invalidate();
```

```
            r1.X = x;
            r1.Y = y;
            r1.Width = 3;
            r1.Height = 3;
            }

            private void chartDetumbling_MouseMove(object sender, MouseEventArgs e)
            {
            var pos1 = e.Location;
            if (prevPosition.HasValue && pos1 == prevPosition.Value)
            return;
            tooltip.RemoveAll();
            prevPosition = pos1;
            var results1 = chartDetumbling.HitTest(pos1.X, pos1.Y, false,
                                    ChartElementType.DataPoint);
            foreach (var result1 in results1)
            {
            if (result1.ChartElementType == ChartElementType.DataPoint)
            {
                        var prop1 = result1.Object as DataPoint;
                    if (prop1 != null)
                        {
                        var pointXPixel = result1.ChartArea.AxisX.ValueToPixelPosition(prop1.XValue);
                        var pointYPixel = result1.ChartArea.AxisY.ValueToPixelPosition(prop1.YValues[0]);

                        // check if the cursor is really close to the point (2 pixels around)

                        tooltip.Show("Time=" + prop1.XValue + "s" + ", ω=" + prop1.YValues[0] + "rad/s",
this.chartDetumbling,
                                    pos1.X, pos1.Y - 15);

                        }
            }
            }
            }

            private void chartDetumbling_PostPaint(object sender, ChartPaintEventArgs e)
            {
            System.Drawing.Font drawFont = new System.Drawing.Font("Verdana", 8);
            System.Drawing.SolidBrush drawBrush = new
System.Drawing.SolidBrush(System.Drawing.Color.Red);

            System.Drawing.StringFormat drawFormat = new System.Drawing.StringFormat();
            if (!(string.IsNullOrEmpty(Xg) && string.IsNullOrEmpty(Yg)))
            {
             e.ChartGraphics.Graphics.DrawString("Time=" + Xg + "\n" + "ω=" + Yg, drawFont, drawBrush, x +
5, y - 2);
             e.ChartGraphics.Graphics.DrawRectangle(new Pen(Color.Red, 3), r1);
            }
            }

            //*****************T*******************************//

            //Asignamos variables a cada gráfica
            private void timer1_Tick(object sender, EventArgs e)
            {
            tiempo++;
           chartVelAng.Series[0].Points.AddXY(tiempo/2, roll);
           chartVelAng.Series[1].Points.AddXY(tiempo/2, pitch);
            chartVelAng.Series[2].Points.AddXY(tiempo/2, yaw);
```

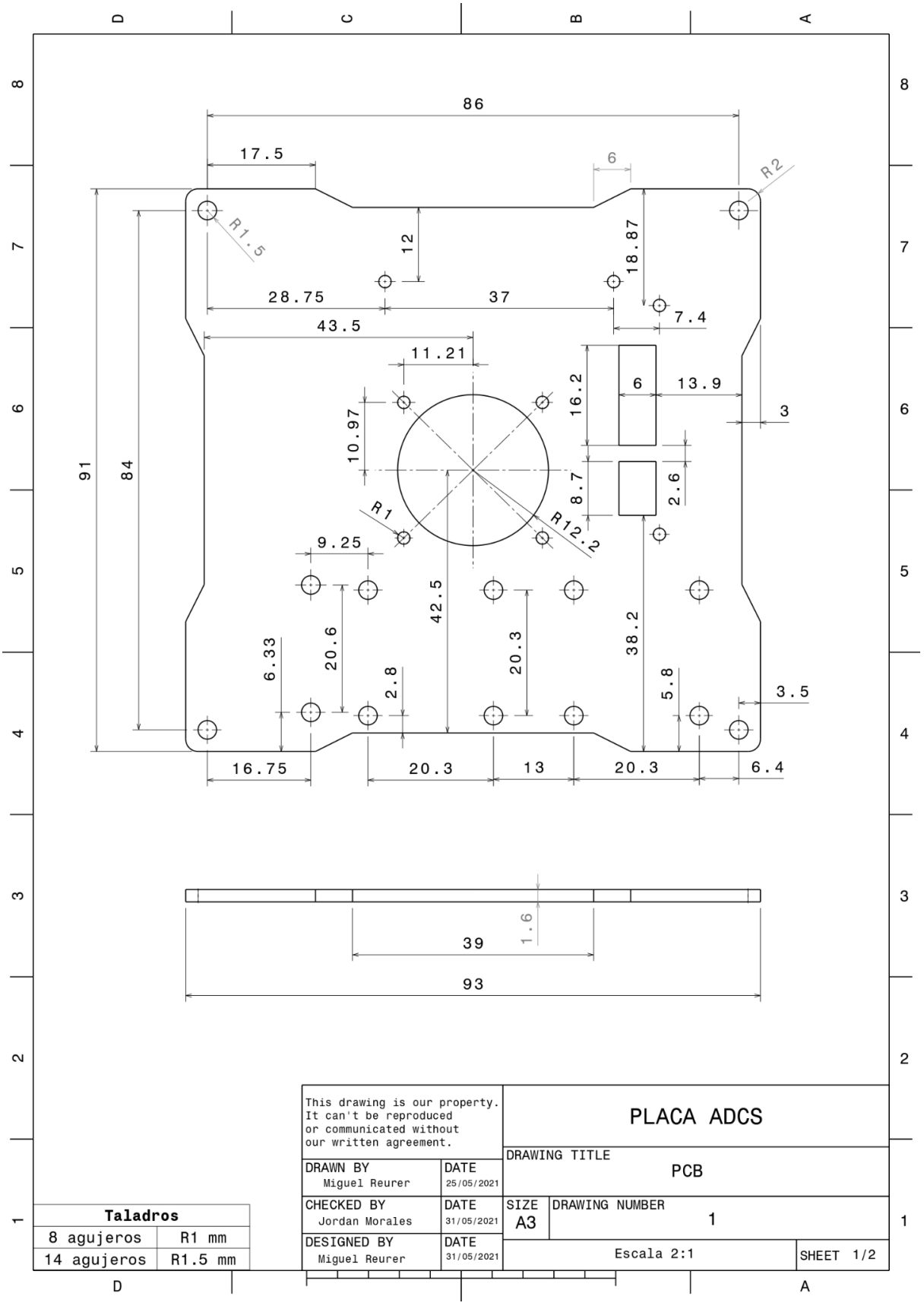
```
chartNadir.Series[0].Points.AddXY(tiempo / 2, vx);
chartNadir.Series[1].Points.AddXY(tiempo / 2, vy);
chartNadir.Series[2].Points.AddXY(tiempo / 2, vz);
  chartDetumbling.Series[3].Points.AddXY(tiempo / 2, 0);

chartAcceleration.Series[0].Points.AddXY(tiempo / 2, ax);
chartAcceleration.Series[1].Points.AddXY(tiempo / 2, ay);
chartAcceleration.Series[2].Points.AddXY(tiempo / 2, az);
chartMagneticField.Series[0].Points.AddXY(tiempo / 2, bx);
chartMagneticField.Series[1].Points.AddXY(tiempo / 2, by);
  chartMagneticField.Series[2].Points.AddXY(tiempo / 2, bz);

chartAngle.Series[0].Points.AddXY(tiempo / 2, roll);
chartAngle.Series[1].Points.AddXY(tiempo / 2, pitch);
chartAngle.Series[2].Points.AddXY(tiempo / 2, yaw);

  chartError.Series[0].Points.AddXY(tiempo / 2, error);

chartDetumbling.Series[0].Points.AddXY(tiempo / 2, vx);
chartDetumbling.Series[1].Points.AddXY(tiempo / 2, vy);
chartDetumbling.Series[2].Points.AddXY(tiempo / 2, vz);
  chartDetumbling.Series[3].Points.AddXY(tiempo / 2, sp_detumbling);


  //Mostrar temperatura

  if (puerto_isopen == true)
  {
  labelTemp.Text = Temp.ToString()+ "°C";
  labelTemp.Location = new Point (40, 30);
  if (Temp >= 75)
  {
            labelTemp.ForeColor = Color.Red;

            labelTMain.BackColor = Color.Red;
            labelTNadir.BackColor = Color.Red;
            labelTDetumbling.BackColor = Color.Red;
            labelTRotation.BackColor = Color.Red;

            labelTMain.Text = "T";
            labelTNadir.Text = "T"; ;
            labelTDetumbling.Text = "T"; ;
            labelTRotation.Text = "T"; ;
  }

  else if (Temp >= 50 & Temp < 75)
  {
            labelTemp.ForeColor = Color.Yellow;
            labelTMain.BackColor = Color.Yellow;
            labelTNadir.BackColor = Color.Yellow;
            labelTDetumbling.BackColor = Color.Yellow;
            labelTRotation.BackColor = Color.Yellow;

     labelTMain.Text = "T";
            labelTNadir.Text = "T"; ;
            labelTDetumbling.Text = "T"; ;
            labelTRotation.Text = "T"; ;
  }
  else
```

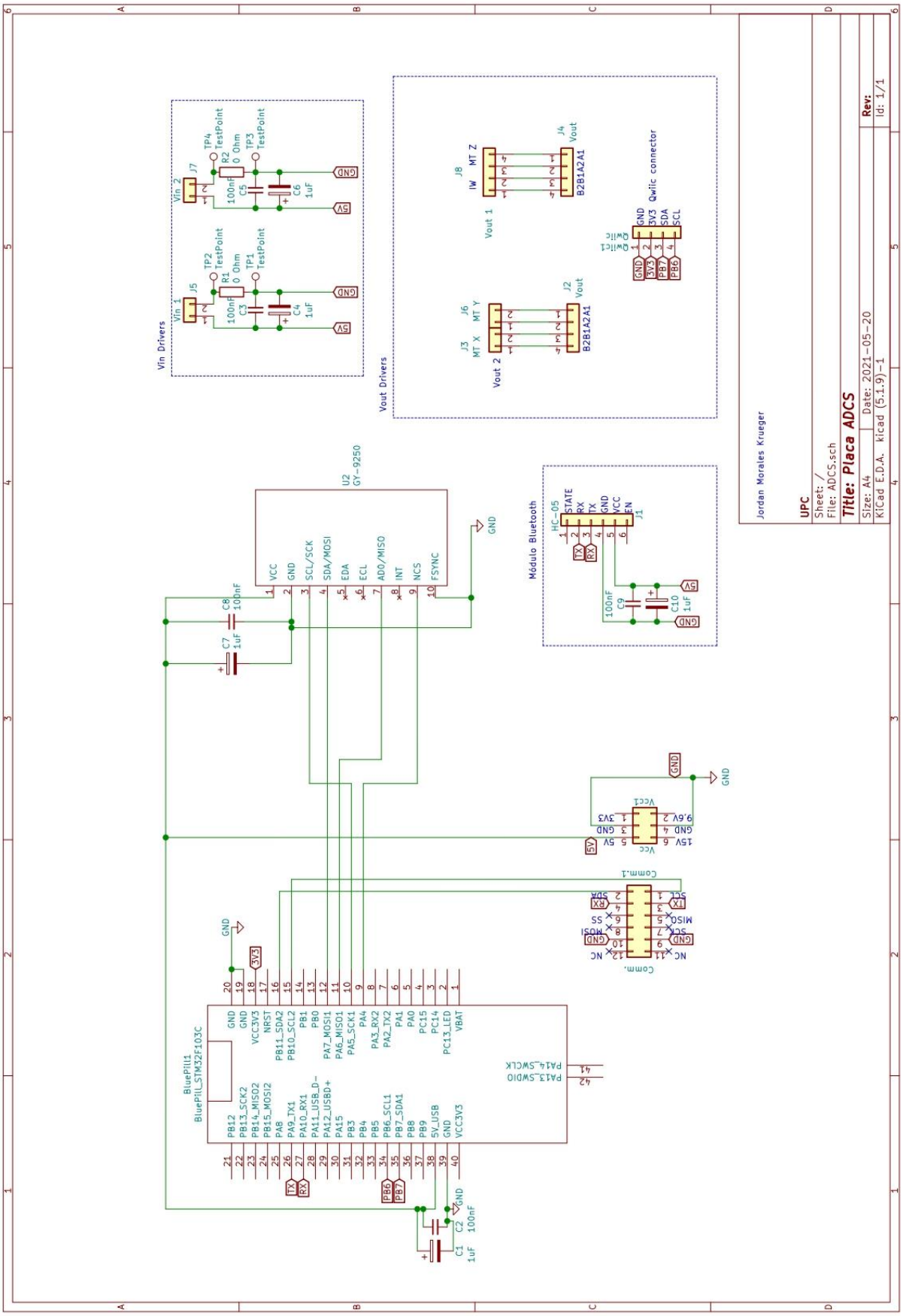
```
        {
                labelTemp.ForeColor = Color.Black;
                labelTMain.Text = "";
                labelTNadir.Text = ""; ;
                labelTDetumbling.Text = ""; ;
                labelTRotation.Text = ""; ;
        }
        }

        }

        //Inicializamos gráficas mediante el botón
        private void buttonStartCharts_Click(object sender, EventArgs e)
        {
        try
        {
        if (puerto_isopen == true)
        {
                timer1.Start();
        }

        }

        catch
        {
        MessageBox.Show("Check connections", "Timer Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        }
        }


        //Boton para detener gráficas
        private void buttonStopCharts_Click(object sender, EventArgs e)
        {
        try
        {
        if (puerto_isopen == true)
        {
                timer1.Stop();
        }
        }

        catch
        {
        MessageBox.Show("No timer set.", "Timer Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        }


        //Link de la universidad
        private void linkLabel1_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
        {
        this.linkLabel1.LinkVisited = true;
        System.Diagnostics.Process.Start("https://eseiaat.upc.edu/en");
        }


    //Minimizar pestaña
        private void button3_Click(object sender, EventArgs e)
        {
```

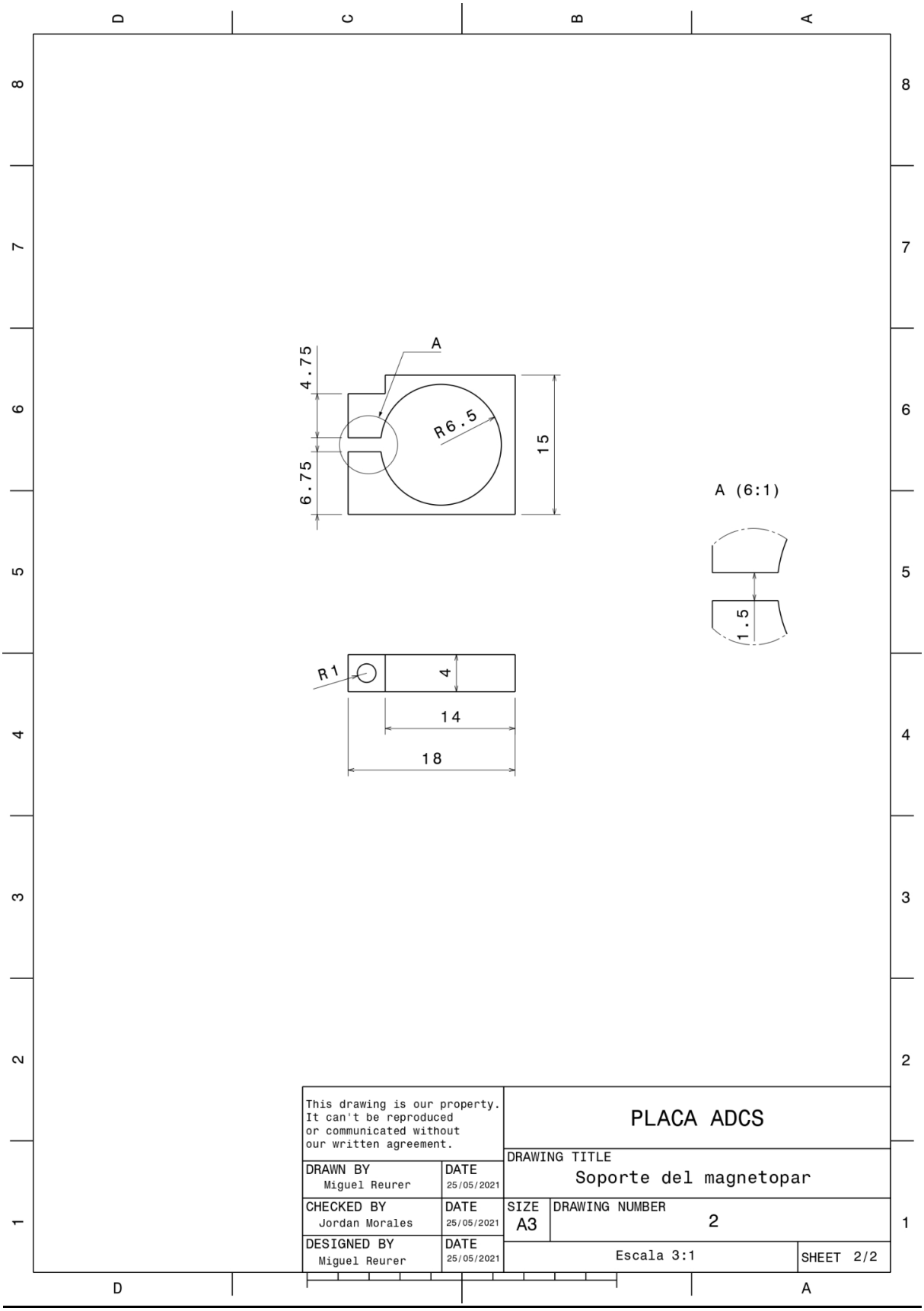
```
this.WindowState = FormWindowState.Minimized;
}

//Maximizar pestaña
private void button2_Click(object sender, EventArgs e)
{
this.WindowState = FormWindowState.Maximized;
}

//Cerrar pestaña
private void button1_Click(object sender, EventArgs e)
{
this.Close();
}


//Redondear borde de la aplicación. Extraído de: https://github.com/codeflow-yt/rounded-corner-form/blob/main/rounded_corner_form/Form1.cs
[DllImport("Gdi32.dll", EntryPoint = "CreateRoundRectRgn")]
private static extern IntPtr CreateRoundRectRgn(
int left,
int top,
int right,
int bottom,
int width,
int height
);

//Mostrar listado de puertos
public ADCS_control()
{
InitializeComponent();
this.comboBox1.DataSource = listado_puerto; // listado de puertos
}


//Inicializamos puerto serie
public void serial()
{

try
{
  this.puerto = new System.IO.Ports.SerialPort("" + comboBox1.SelectedItem, 115200, System.IO.Ports.Parity.None, 8, System.IO.Ports.StopBits.One); //configuramos puerto serie
this.puerto.DataReceived += new System.IO.Ports.SerialDataReceivedEventHandler(recepcion); //llamamos al método de recepción

}
catch (Exception) //mensaje de error
{
MessageBox.Show("Please verify:" + System.Environment.NewLine + "- Voltage" + System.Environment.NewLine + "- Port Connexion", "COM Port Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}
}


public void recepcion(object sender, System.IO.Ports.SerialDataReceivedEventArgs e)
{
```

```
            try
            {
            Thread.Sleep(10); //damos tiempo a que el buffer del arduino tenga tiempo a llenarse en la parte física
del PC
            datos_puerto = this.puerto.ReadLine(); //leemos datos y los guardamos
            Console.WriteLine(datos_puerto);
            if (datos_puerto.StartsWith("$")) //si los datos recibidos empiezan con $ llamamos a la funcion
"actualizar"
            { this.Invoke(new EventHandler(actualizar)); }

            }

            catch (Exception) { }

            }

            public void actualizar(object s, EventArgs e)
            {
            datos_puerto = datos_puerto.Remove(0, 1);   //borramos primer carácter $
            string[] arreglo = datos_puerto.Split(';'); //separamos los datos separados con ";"
                                        //aseguramos que el formato decimal sea el correcto p.ej 9,8 en lugar de 9.8

            //Guardamos las variables separadas por comas mediante el arreglo
            vx = Convert.ToDouble(arreglo[0].Replace(".", ","));
            vy = Convert.ToDouble(arreglo[1].Replace(".", ","));
            vz = Convert.ToDouble(arreglo[2].Replace(".", ","));
            ax = Convert.ToDouble(arreglo[3].Replace(".", ","));
            ay = Convert.ToDouble(arreglo[4].Replace(".", ","));
            az = Convert.ToDouble(arreglo[5].Replace(".", ","));
            bx = Convert.ToDouble(arreglo[6].Replace(".", ","));
            by = Convert.ToDouble(arreglo[7].Replace(".", ","));
            bz = Convert.ToDouble(arreglo[8].Replace(".", ","));
            roll = Convert.ToDouble(arreglo[9].Replace(".", ","));
            pitch = Convert.ToDouble(arreglo[10].Replace(".", ","));
            yaw = Convert.ToDouble(arreglo[11].Replace(".", ","));
            Temp = Convert.ToDouble(arreglo[12].Replace(".", ","));
            error = Convert.ToDouble(arreglo[13].Replace(".", ","));
            }


            private void Form1_Load(object sender, EventArgs e)
            {
            Region = System.Drawing.Region.FromHrgn(CreateRoundRectRgn(0, 0, Width, Height, 25, 25));

            }
    }
}
```

APÉNDICE C: PLANOS DE LA PCB


86
17.5
6
R2
R1.5
12
18.87
28.75
37
7.4
43.5
11.21
16.2
6
13.9
3
10.97
91
84
8.7
2.6
R1
R12.2
9.25
42.5
20.6
20.3
38.2
6.33
2.8
5.8
3.5
16.75
20.3
13
20.3
6.4
1.6
39
93
Taladros
8 agujeros R1 mm
14 agujeros R1.5 mm
This drawing is our property. It can't be reproduced or communicated without our written agreement.
PLACA ADCS
DRAWN BY Miguel Reurer
DATE 25/05/2021
DRAWING TITLE PCB
CHECKED BY Jordan Morales
DATE 31/05/2021
SIZE A3
DRAWING NUMBER 1
DESIGNED BY Miguel Reurer
DATE 31/05/2021
Escala 2:1
SHEET 1/2


Vin Drivers
Vout Drivers
Módulo Bluetooth
HC-05
U2
GY-9250
BluePill1
BluePill_STM32F103C
Qwiic connector
Jordan Morales Krueger
UPC
Sheet: /
File: ADCS.sch
Title: Placa ADCS
Size: A4
Date: 2021-05-20
KiCad E.D.A. kicad (5.1.9)-1
Rev:
Id: 1/1


A
4.75
6.75
R6.5
15
A (6:1)
1.5
R1
4
14
18
This drawing is our property.
It can't be reproduced
or communicated without
our written agreement.
PLACA ADCS
DRAWING TITLE
Soporte del magnetopar
DRAWN BY
Miguel Reurer
DATE
25/05/2021
CHECKED BY
Jordan Morales
DATE
25/05/2021
SIZE
A3
DRAWING NUMBER
2
DESIGNED BY
Miguel Reurer
DATE
25/05/2021
Escala 3:1
SHEET 2/2