UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Sistema de control de un UAS mediante el controlador RP2040

Trabajo Fin de Grado

Grado en Ingeniería Electrónica Industrial y Automática

AUTOR/A: Valls Ortiz, Alberto

Tutor/a: Balbastre Betoret, Patricia

CURSO ACADÉMICO: 2022/2023

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Sistema de control de un UAS mediante el controlador RP2040

Realizado por:
Alberto Valls Ortiz

Tutorizado por:
Patricia Balbastre Betoret

Curso Académico: 2022 - 2023

Agradecimientos

Este trabajo de fin de grado va dedicado a todas las personas que han creído en mí y las que me han apoyado durante el transcurso de la etapa de este grado. En especial, quería agradecer a mis padres y hermana por haberme dado la oportunidad de poder estudiar y formarme profesionalmente. A Ana Sanchis por haber sido la persona que más me ha apoyado y motivado para continuar. A mis compañeros Ángelo Wang, Iván Silva, Miquel Pons e Iván Esteve por haberlos conocido y haber compartido esta experiencia. Y a mi tutora del trabajo de Fin de grado, Patricia Balbastre, por haberme ayudado en esta tarea y a comprender el trabajo que conlleva hacer un proyecto de esta magnitud.

Resumen

El objetivo de este trabajo es el diseño e implementación de un sistema de control para un UAS. El UAS es controlado por un sistema basado en el microcontrolador RP2040. Este controlador de doble núcleo procesa datos de sensores, ejecuta algoritmos de control y genera señales para los actuadores. Los sensores proporcionan información sobre la posición, altitud y entorno del UAS. Los actuadores convierten las señales de control en acciones físicas, como controlar la velocidad de los motores y los movimientos de los componentes. Algoritmos sofisticados se implementan para garantizar vuelo estable y control preciso. La comunicación inalámbrica permite la interacción con una estación de control y una interfaz de usuario proporciona información en tiempo real.

Abstract

Design and Implementation of an UAS Control System. The UAS is controlled by a system based on the RP2040 microcontroller. This dual-core controller processes sensor data, executes control algorithms, and generates signals for the actuators. Sensors provide information about the UAS's position, altitude, and environment. Actuators convert control signals into physical actions, such as controlling motor speed and component movements. Sophisticated algorithms are implemented to ensure stable flight and precise control. Wireless communication enables interaction with a control station, and a user interface provides real-time information.

Índice

Índice de figuras

Sistema de control de un UAS mediante el controlador RP2040

Documento 1. Memoria

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Trabajo Final de Grado en Ingeniería Electrónica Industrial y Automática

Realizado por: Alberto Valls Ortiz

Tutorizado por: Patricia Balbastre Betoret

Curso Académico: 2022 - 2023

1. Memoria

1.1. Objeto

Este proyecto tiene como objetivo desarrollar un sistema de control altamente eficiente y preciso para un UAS (Remotely Piloted Aircraft System) utilizando el controlador RP2040, un microcontrolador de doble núcleo; procesando datos de sensores, ejecutando algoritmos de control y generando señales para los motores.

El RP2040, los módulos de sensores y los actuadores serán la base de la arquitectura del sistema de control. Los sensores como el IMU (Unidad de medición inercial), el GPS (Sistema de geo-posicionamiento) y otros proporcionarán información crucial sobre la posición, la actitud y el entorno del UAS. Los controladores de motor y servomotores, por ejemplo, convertirán las señales de control generadas por el RP2040 en acciones físicas para el vuelo y las maniobras del UAS.

1.2. Estudio de necesidades, factores a considerar: limitaciones y condicionantes

1.2.1. Historia de los drones

En julio del año 1849, el imperio Austrohúngaro puso en marcha más de 200 globos aerostáticos no tripulados, cargados de bombas hacia la ciudad de Venecia. Desde entonces la idea de una aeronave no tripulada se empezó a plantear en las mentes de la gente de esa época. Veinte años más tarde, con Samuel P. Langley, se desarrollaron naves de vapor que servirían como precedente a los drones actuales. Aunque estos no eran drones como tal, sino aviones sin piloto que se pudieron trasladar a lo largo del río PotoMac en Washington DC, estados unidos.

Posteriormente, cuando Nikola Tesla empezó a patentar sus inventos, entre uno de ellos se hallaba una máquina capaz de sobrevolar ciudades. Aunque el inventor no fue capaz de realizarlo, dejó el legado para que años más tarde sí que se cumpliera.

Tras la idea del gran Nikola Tesla, los hermanos Jaques y Louis Breguet crearon el primer cuadricóptero de la historia. Pero el prototipo no funcionaba como lo esperado, únicamente se alzaba 60 cm del suelo y requería de cuatro personas para que este estuviera estabilizado.

Subsiguientemente, llega la primera guerra mundial, y es entonces cuando, el inglés Archibald Low con la tecnología de control remoto de tesla, crea el verdadero avión sin piloto de la historia, el Ruston Proctor Aerial Target. Pretendiendo que este fuera una bomba volante en el campo de combate.

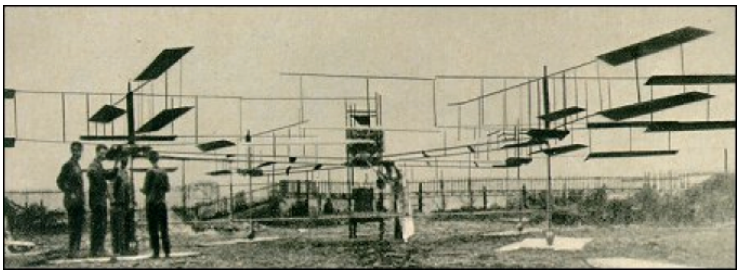


Figura 1: Cuadricóptero de los hermanos Breguet

Ya avanzado el siglo XIX, en 2002 se empieza a comercializar los drones para uso no militar. Utilizándose para una operación de policía en estados unidos. A partir de aquí los drones empiezan a ponerse en el punto de mira y a desarrollarse las clasificaciones entre ellos.[1]

1.2.2. Definición de dron y tipos

Un dron es un vehículo aéreo no tripulado utilizado en el ámbito militar y civil. Podemos diferenciar los drones por varias tipologías[2]:

1.2.2.1 Clasificacion segun la definición

Drone El nombre más popularizado para referirnos a las aeronaves no tripuladas. Su origen viene de la palabra inglesa drone que significa zángano.

RPA o RPAS Son las siglas en inglés de Remotely Piloted Aircraft y Remotely Piloted Aircraft System. Este concepto hace referencia al control remoto de la aeronave.

UAV o UAS Las siglas en inglés de Unmanned Aerial Vehicle y Unmaned Aerial System. Este concepto hace referencia a la falta de un piloto físico a bordo de la aeronave.

1.2.2.2 Clasificación según la sustentación

Drones de ala fija Estas aeronaves necesitan una velocidad de vuelo inicial para poder volar. No pueden hacer un despegue por sí solos; necesitan una persona o un mecanismo que lo haga. Son lo más parecido a un avión normal desde un punto de vista estético. Este tipo de drones son ideales para sobrevolar y mapear grandes superficies porque su aerodinámica les permite volar durante varias horas.

Drones de ala rotatoria o multi-rotores Son las aeronaves no tripuladas más populares y populares. La sustentación de estos drones es posible gracias a las hélices que llevan incorporadas en los extremos de cada brazo. Cada hélice está impulsada por un motor, lo que permite una gran estabilidad al volar. Los multi-rotores tienen la capacidad de permanecer quietos mientras sobrevuelan en el mismo lugar, a diferencia de los drones con ala fija.

1.2.2.3 Clasificación según el numero de brazos

Partiendo que el dron no es de ala fija, o sea que no necesita sustentación inicial para poder volar. Se pueden clasificar según el número de brazos que posean en tricópteros, cuadricópteros, hexacópteros y octacópteros.

1.2.3. Estudio de necesidades

En el estudio de necesidades para el desarrollo del sistema de control para UAS utilizando el controlador RP2040, es importante considerar diversos factores, así como las limitaciones y condicionantes del proyecto. Algunos de ellos son los siguientes:

Requisitos de rendimiento: Estos requisitos se basan en la capacidad de afrontar cualquier inclemencia del temporal, reducir el consumo energético para aumentar la autonomía de vuelo, además de incorporar sistemas de precisión de control para aumentar la maniobrabilidad del UAS.

Normativas y regulaciones: Según la regulación actual sobre los drones, se debe avisar que la zona de vuelo no es transitable y que existe una altura máxima para evitar colisiones con aeronaves de mayor tamaño como aviones o avionetas. Además de la normativa y tamaño, que se detalla posteriormente.

Seguridad: La seguridad del sistema de control implica garantizar la confiabilidad y la resistencia a fallos del sistema, así como proteger contra amenazas cibernéticas y salvaguardar los datos transmitidos. Esto se logra mediante el uso de protocolos de control incorporados en los terminales y el anonimato de las señales de control de percepción del UAS.

Limitaciones de peso y tamaño: Dado que el UAS tiene restricciones de peso y dimensiones, según el real decreto RD 2019/945, el diseño del sistema de control debe tener en cuenta estas limitaciones para garantizar una integración adecuada en la estructura del UAS y no afectar negativamente su rendimiento.

Este es uno de los condicionantes más elevados, ya que el sistema está pensado para manejar UAS de pequeño tamaño, además de poseer cuatro sistemas de propulsión. Se le da la capacidad al consumidor de mejorar el sistema y de implantar mejoras adicionales como incrementar la cantidad de actuadores o de

mejorar el sistema de control de batería.

Interoperabilidad: Si se requiere la integración del sistema de control con otros sistemas o equipos, es necesario considerar la interoperabilidad y la compatibilidad de protocolos y comunicaciones para lograr una integración sin problemas.

Condiciones ambientales: Dependiendo del entorno operativo previsto para el UAS, se deben considerar factores como la temperatura, humedad, viento y condiciones atmosféricas adversas. El sistema de control debe ser capaz de funcionar de manera confiable en estas condiciones y garantizar un rendimiento óptimo.

Limitaciones de recursos: Es importante tener en cuenta las limitaciones de recursos disponibles, como el presupuesto, el tiempo y las capacidades técnicas. Estas limitaciones pueden influir en el alcance y la capacidad del sistema para poder llegar a su destino.

Objetivos de desarrollo sostenible: También es importante tener en cuenta que existen unos objetivos impuestos y que deberían cumplirse algunos. El objetivo número 9 y el 11 serían los que más cuadran con este proyecto. Siendo estos el requisito de industria, innovación e infraestructura, y el de ciudades y comunidades sostenibles respectivamente.

Teniendo en cuenta todo lo descrito anteriormente, es crucial que el sistema cumpla las características expuestas, como un control de vuelo, un sistema para corregir errores en trazadas y en caso de falta de potencia, que el UAS tenga la capacidad de volver al punto de inicio.

1.3. Planteamiento de soluciones alternativas y justificación de la solución aportada

1.3.1. Soluciones Alternativas

En el proceso de desarrollo de cualquier proyecto, es esencial considerar varias opciones y evaluar soluciones alternativas para abordar los problemas que surgen. Antes de elegir la mejor solución, es necesario examinar una variedad de opciones. En lugar del RP2040 se podría utilizar otro tipo de microcontrolador como:

1.3.1.1 Arduino

En lugar del RP2040, se podría considerar otro microcontrolador disponible en el mercado como el Arduino. Siendo esta la opción que se suele abordar en estos casos. Debido a su amplia comunidad de desarrolladores y su gran gama

de microcontroladores.

En este caso se utilizaría el controlador Atmega328p, ya que es el que mejor se adapta a las necesidades aportadas anteriormente. Pero esta opción carece de salidas de PWM, solo posee 6[3], y no tendríamos la capacidad de ampliación de actuadores.

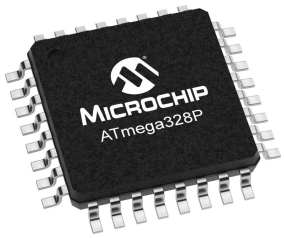


Figura 2: AtMega 328p

1.3.1.2 ESP

En este caso se reemplazaría el microcontrolador RP2040, por el ESP32. Este microcontrolador cumple todas las características que se piden para el sistema. Aunque esta opción no se ha abordado por la alta complejidad del microcontrolador para poder incorporar módulos, además que su comunidad no es tan amplia y si surge algún inconveniente.

Figura 3: Controlador ESP32

1.3.1.3 Sistemas de control externos

En lugar de utilizar un microcontrolador interno como el RP2040, se podría optar por sistemas de control externos, como un controlador de vuelo comercialmente disponible. Un ejemplo sería el controlador PixHawk. Siendo este capaz de controlar el dron sin configuración previa. Siguiendo la filosofía Universal Plug And Play. Esta se basa en que al acabar el montaje, el sistema ya es capaz

de volar por sí mismo.

Figura 4: Controladora PixHawk

El problema de este tipo de sistemas es el coste, ya que es elevado y no posee la capacidad de personalización como la que tiene una controladora diseñada por la comunidad.

1.3.2. Solución final escogida

La solución final elegida tiene varias características destacables que la convierten en la mejor opción para este proyecto.

Potencia y flexibilidad: El RP2040[4] ofrece un alto rendimiento gracias a su arquitectura de doble núcleo y capacidad de multitarea. Esto permite un procesamiento eficiente de los datos de los sensores y una ejecución fluida de los algoritmos de control. Además, el RP2040 es altamente programable, lo que facilita la adaptación y optimización del sistema de control para diferentes aplicaciones de UAS.

Coste-efectividad: El RP2040 es conocido por su relación calidad-precio. Su precio asequible permite la implementación de soluciones de control avanzadas sin incurrir en altos costos. Esto resulta especialmente beneficioso en proyectos con presupuestos limitados o en la implementación de sistemas de control en múltiples UAS.

Amplia comunidad de soporte: El RP2040 cuenta con una comunidad activa de desarrolladores y una amplia gama de recursos disponibles en línea. Esto facilita la obtención de soporte técnico, la resolución de problemas y el acceso a bibliotecas de software y ejemplos de código, lo que agiliza el desarrollo del sistema de control.

Integración y compatibilidad: El tamaño compacto del RP2040 permite una fácil integración en el diseño del UAS, teniendo en cuenta las limitaciones de peso y tamaño. Además, sus interfaces de comunicación estándar facilitan la conexión con una variedad de sensores y actuadores, lo que garantiza una integración sin problemas del sistema de control.

Dicho todo esto, la solución aportada posee dos partes diferenciadas como son el subsistema de hardware y el subsistema de software.

1.3.2.1 Subsistema de hardware

Controlador RP2040 Es un microcontrolador desarrollado por la empresa RaspBerry. Este es uno de los microcontroladores más pequeños y versátiles que existen a día de hoy. En este podemos implantar código de MicroPyhton y de C, además de otros lenguajes que se están desarrollando como Rust-for-Embedded.

El microcontrolador posee dos núcleos ARM Cortex-M0+, que pueden funcionar hasta una frecuencia de 133 MHz. También tiene 256k de memoria RAM y 30 pines de GPIO para una mejor comunicación con el entorno.

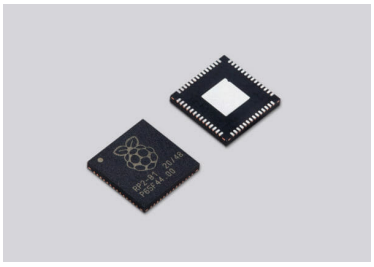


Figura 5: RP2040

MPU-6050 Es un módulo integrado IMU, Unidad de medición inercial. En este, mediante comunicación I2C, donde el módulo actúa como esclavo y el controlador actúa como maestro, se pueden obtener los valores de giroscopio. Con estos se procesan los datos y mediante el PID, se realizará una acción de control para enviar la señal a los actuadores y corregir trayectorias de vuelo.

Figura 6: Sensor MPU-6050

Winbond-250128 El módulo que se ha optado en esta opción para ser utilizado de memoria flash externa al microcontrolador, para poder almacenar el programa y las funciones programadas.

Figura 7: Winbond-250128

Convertidor 3.3 V Mediante este tipo de convertidor se establece la señal del sistema a 3.3 voltios, realizando así las conversiones necesarias para que el chip RP2040 funcione correctamente. Si el sistema carece de alimentación por parte de una fuente USB, como es el caso de la carga o de la programación de puertos, se podrá conectar una batería externa al terminal de 3.3 para poder conectarse remotamente al sistema.

Figura 8: Convertidor a 3.3 V

Cristal oscilador de 12 MHz El sistema debe funcionar mediante una señal oscilante y periódica para poder controlar la frecuencia de computación. Adaptamos la señal a 12 MHz porque es suficiente tiempo de refresco para los sensores captar variaciones y además procesar los cálculos de estabilización de sistema.

1.3.2.2 Subsistema de software

Lenguaje de programación C Debido a la versatilidad y la alta capacidad de edición de registros, además de la velocidad que puede aportar a los cómputos, se ha elegido como lenguaje C.

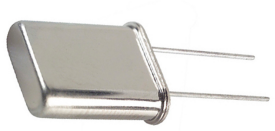


Figura 9: Cristal oscilador de 12 MHz

Librerías de control del RP2040 Gracias a un proyecto creado por desarrolladores de la comunidad, se ha creado una librería para poder programar el controlador RP2040[6]. Habilitando así los buses GPIO además de los canales I2C.

Librerías de control IMU Para el control del IMU se han utilizado las librerías que el fabricante del circuito integrado creo para C, y mediante una pequeña API se pueden interpretar los datos y procesarlos.

Control PID Para el control PID se ha desarrollado una pequeña librería personalizada, incluida en el proyecto, para poder implantar las funciones de viraje del sistema.[5]

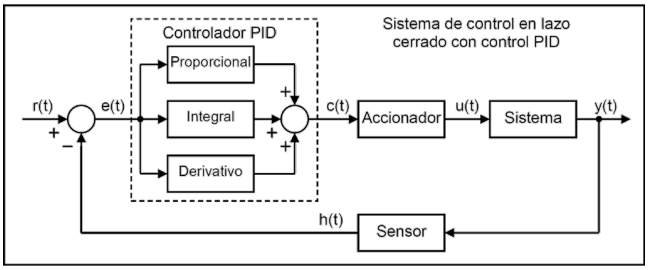


Figura 10: PID en diagrama de bloques

Se ha implantado un PID para cada eje de rotación y así tener controlada la posición del sistema en todo momento. En el posterior documento se le llamará *roll*, *pitch* y *yaw* a cada una de las direcciones. 11

1.4. Descripción detallada de la solución aportada

El sistema que se va a diseñar y configurar se detallará en el trabajo actual, como se mencionó anteriormente. Para lograrlo, se establecieron los requisitos

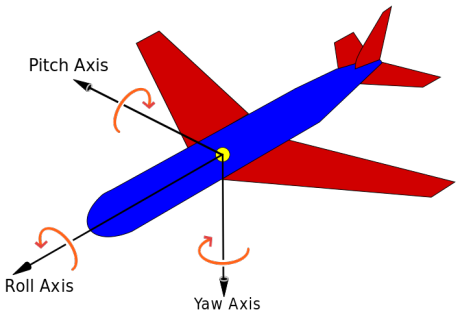


Figura 11: Representación de Roll, Pitch, Yaw

que debe cumplir el sistema.

Cuando se discuten los requisitos, consideraremos el propósito del sistema, cómo abordar los problemas propuestos, así como las respuestas a las preguntas de qué función se llevará a cabo, cómo y cuánto tiempo estará en funcionamiento, y de manera más general, la razón detrás de este desarrollo.

El sistema podrá ser controlado mediante radiocontrol. Para poder proteger a la nave de las inclemencias del tiempo y de cambios bruscos en la trayectoria, el sistema tiene implantado un PID que aportara una señal estabilizadora y completamente a medida para cada caso de construcción.

Aunque el sistema está pensado para ser liviano y no tener ningún tipo de sistema acoplado, este puede ser modificado para que cumpla con los requisitos del consumidor final. Añadiendo actuadores adicionales, sensores y algún tipo de videocámara. Todo el sistema será de código abierto para que el consumidor pueda cambiar lo que necesite a su merced.

Como último requerimiento, también existe la capacidad de ampliar el sistema de procesamiento, añadiéndole algún módulo ya creado e importar bibliotecas o parches al código para mejorar el sistema y la solución aportada.

Así que con todo expuesto, el sistema estará basado en un bucle simple de retroalimentación donde los sensores captan las distintas variables del entorno y calculan una modificación de la señal original para poder estabilizar el sistema.12

1.4.1. Funcionamiento del sistema

Al inicializar el sistema, se tomará como referencia la posición actual. Se establecerán todos los valores a los actuales de giroscopio. Y se procederá a iniciar la marcha.

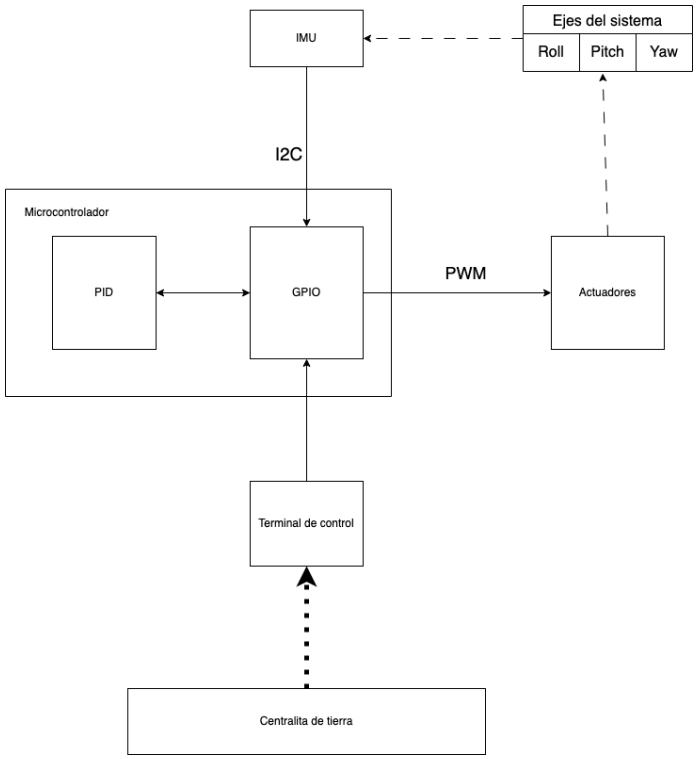


Figura 12: Diagrama de funcionamiento del sistema

En primera instancia, si el sistema carece de señal de control, este no funcionará. En caso contrario se podrá controlar mediante las palancas de un terminal controlador externo a este. Ya que la señal del controlador se recibe mediante un receptor externo a la placa, se debe aplicar una función para establecer los límites y como se va a interpretar la señal.

Debido a que las palancas del controlador son dispositivos lineales resistivos, la señal será lineal y según el tipo de palanca, el UAS deberá de comportarse de una manera o de otra. Existen dos tipos de palancas en los terminales controladores.

1.4.1.1 Captación e interpretación de la señal del terminal

En primer lugar, la **Palanca de no retorno**. Esta es la palanca que se utiliza para la propulsión vertical. Esta posee un mínimo en 0 y un máximo en el tope de maniobra. Así que podemos establecer una función de transferencia para este comportamiento como:

$$f(x) = \begin{cases} 0 \text{ si } x = 0 \\ 1 \text{ si } x = \text{Rango Máximo} \end{cases} \tag{1}$$

Entonces aplicando los conceptos de función de transferencia lineal con estos parámetros se debe cumplir esta ecuación y calcular los valores de m y n:

$$f(x) = mx + n \tag{2}$$

$$m = \frac{f(x_{max}) - f(x_{min})}{x_{max} - x_{min}} \tag{3}$$

Sustituyendo el valor de x mínimo por 0 y el de x máximo por Rango Máximo, además de su valor respectivo en el eje de las ordenadas, el resultado de la m es:

$$m = \frac{\text{Rango Máximo} - 0}{1 - 0} \tag{4}$$

Gracias a esta deducción encontramos que el valor de la pendiente es del valor del Rango Máximo.

$$m = \text{Rango Máximo} \tag{5}$$

E interpolando el resultado a la ecuación de la recta:

$$f(x) = (\text{Rango Máximo})x + n \tag{6}$$

Solo faltaría obtener el valor de la constante n, que sustituyendo con los valores iniciales en la ecuación (**??**)

$$f(0) = (\text{Rango Máximo}) \times 0 + n \tag{7}$$

$$n = 0 \tag{8}$$

Y con todos estos valores la función de transferencia será:

$$f(x) = (\text{Rango Máximo}) \times x \in [0, 1] \tag{9}$$

En segundo lugar, las palancas con retorno, estas al aplicar un movimiento a la palanca y soltarla volverán al estado inicial. Así que con estos datos podemos establecer los tres valores que definirán la función de transferencia:

$$f(x) = \begin{cases} 0 \text{ si } x = 0 \\ 1 \text{ si } x = \text{Rango Máximo} \\ -1 \text{ si } x = \text{Rango Mínimo} \end{cases} \tag{10}$$

Aunque para esta función de transferencia se podría utilizar la función sigmoide, es más apropiado establecerla como una función lineal. Así que su forma será:

$$f(x) = mx + n \tag{11}$$

Como se ha explicado anteriormente, primero se debe calcular el valor de la pendiente con nombre m:

$$m = \frac{f(x_{max}) - f(x_{min})}{x_{max} - x_{min}} \tag{12}$$

Y sustituyendo los valores máximos y mínimos, y estableciéndolos en el periodo de [-1, 1] el valor de m se quedaría en:

$$m = \frac{\text{Rango Máximo} - \text{Rango Mínimo}}{1 - (-1)} = \frac{\text{Rango Máximo} - \text{Rango Mínimo}}{2} \tag{13}$$

Después de esto se debe calcular el valor de n en la ecuación:

$$f(x) = (\frac{\text{Rango Máximo} - \text{Rango Mínimo}}{2}) \times x + n \tag{14}$$

Y aplicando los valores para f(0) = 0 obtenemos el valor de n:

$$f(0) = 0 = (\frac{\text{Rango Máximo} - \text{Rango Mínimo}}{2}) \times 0 + n \tag{15}$$

$$n = 0 \tag{16}$$

Y con esto la función de transferencia para las palancas con retorno es:

$$f(x) = (\frac{\text{Rango Máximo} - \text{Rango Mínimo}}{2}) \times x \tag{17}$$

Gracias a estos valores, se pueden trasladar los valores a los que el sistema pueda procesar correctamente.

1.4.1.2 Procesamiento de los datos de entrada del terminal y de los sensores

A continuación de captar la señal que el terminal de control manda, se debe procesar la señal mediante un control PID, (Proporcional, Integrador y Derivador).

Se realizan las comparaciones necesarias con la señal para determinar el valor de la salida a los actuadores. Para ello se han tuneado los valores de las constantes de control para que este sea más preciso y no entre en pérdida.

1.4.1.3 Salida de la señal a los actuadores

Para que la señal pueda ser interpretada por los controladores ESC (Control Electrónico de Velocidad), la señal debe estar expresada en las frecuencias entre 6 kHz y 48 kHz, además la señal debe ser una PWM[7].

Señal PWM Una señal PWM es una función que se repite con el tiempo y posee la forma de: En esta la amplitud no varía, siempre está en los límites que

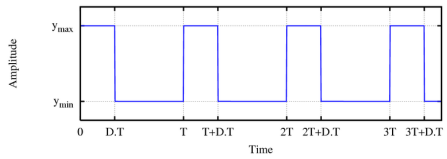


Figura 13: Señal de ejemplo

se designan. Donde varía es en la frecuencia de ON/OFF. Durante un valor DT inferior a T la onda está en ON y en la parte siguiente en OFF. Así que con esto se puede controlar la velocidad, ya que si se realiza el valor medio de la señal en un periodo, el valor es proporcional a D. A continuación se muestra el cálculo de como se realiza el valor medio:

$$f(x) = \begin{cases} Y_{max} \text{ si } kT \leq t \leq k \cdot DT, k \in N \\ Y_{min} \text{ si } k \cdot DT \leq t \leq k \times (T+1), k \in N \\ D \in [0,1] \end{cases} \tag{18}$$

Entonces si realizamos el valor medio de la función en cada periodo:

$$V = \frac{1}{T} \int_{x=0}^{x=DT} Y_{max} \cdot dt = \frac{Y_{max} \cdot t}{T}\Big|_0^{DT} = \frac{Y_{max} \cdot DT}{T} - \frac{Y_{max} \cdot 0}{T} = Y_{max} \cdot D, D \in [0,1] \tag{19}$$

Con esto se puede deducir que el valor de la integral es de Ymax por D siendo esta un valor entre 0 y 1.

1.5. Conclusión

Mediante la creación del UAS, se puede concluir que se han cumplido los objetivos propuestos al inicio de este trabajo, siendo estos:

- Se ha diseñado el sistema para que cumpla los criterios de estabilidad y precisión dentro de los límites establecidos por las leyes y por los requerimientos previos.
- Se han seleccionado los componentes necesarios mediante el conocimiento adquirido en el grado.
- El sistema se ha diseñado gracias a varios PID que han resultado de gran utilidad en los sistemas de estabilidad.
- Se han generado funciones de transferencia para la comprensión de cualquier terminal terrestre.
- Se ha establecido la salida del sistema en una función PWM para mejor integración del sistema con los actuadores externos.

En conclusión, con los conocimientos adquiridos y el apoyo de los profesores durante el grado universitario en la Universidad Politécnica de Valencia se ha logrado diseñar un sistema desde cero con una idea base, además este cuenta con la mayoría de funciones deseadas.

1.6. Propuesta de mejoras

Para mejorar este sistema hay ciertos puntos que se podrían mejorar en el código como:

- Crear un sistema de guiado para que el UAS fuera autónomo y pudiera volver al punto de partida al detectar bajadas de tensión.
- Crear un sistema de auto-creación de rutas para que no se necesite un terminal externo de control.

Además, para mejorar el sistema también se podrían agregar módulos como:

- Un módulo GPS para localizar el sistema en caso de fallo o para poder trazar las rutas correctamente.
- Un módulo ESC para no requerir la presencia de los ESC externos y reducir el peso del UAS, además de su coste.

1.7. Bibliografía

Referencias

[1] Historia de los drones por la página eldrone.es http://eldrone.es/historia-de-los-drones/

[2] Tipos de drones por la página novodrone.com https://novodrone.com/tipos-de-drones/

[3] Hoja de datos de ATmega328p https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

[4] Web del fabricante del RP2040 https://www.raspberrypi.com/products/rp2040/

[5] Brogan, William L. Modern Control Theory. 3rd. ed., Prentice Hall, 1991.

[6] Librería de Raspberry pi Pico https://github.com/raspberrypi/pico-sdk

[7] Modulación por ancho de pulsos https://es.wikipedia.org/wiki/Modulaci%C3%B3n_por_ancho_de_pulsos

Sistema de control de un UAS mediante el controlador RP2040

Documento 2. Planos

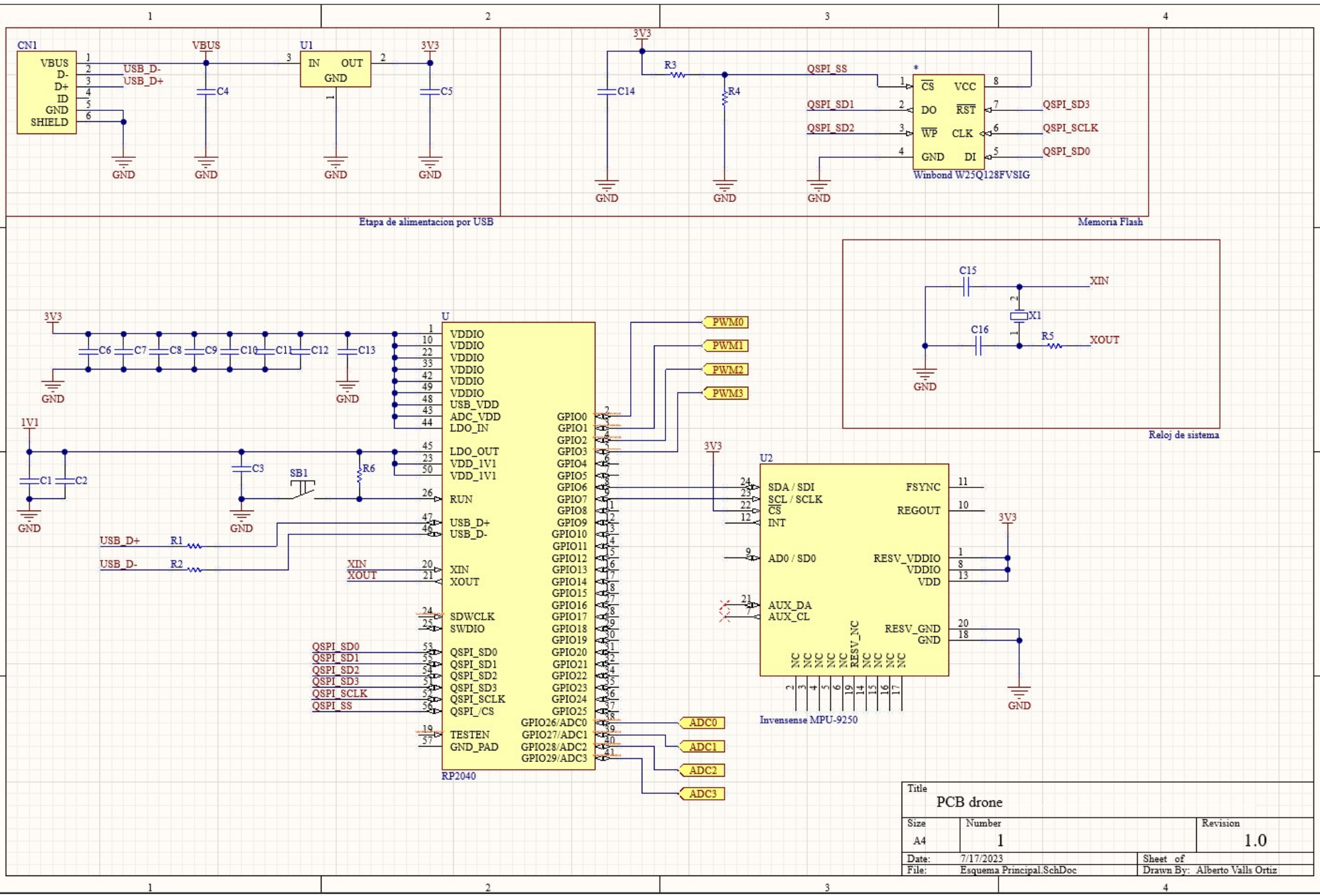
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Trabajo Final de Grado en Ingeniería Electrónica Industrial y Automática

Realizado por: Alberto Valls Ortiz

Tutorizado por: Patricia Balbastre Betoret

Curso Académico: 2022 - 2023


CN1
VBUS
D-
D+
ID
GND
SHIELD
USB_D-
USB_D+
VBUS
C4
U1
IN
OUT
GND
3V3
C5
Etapa de alimentacion por USB
3V3
R3
C14
R4
QSPI_SS
QSPI_SD1
QSPI_SD2
CS
DO
WP
GND
VCC
RST
CLK
DI
QSPI_SD3
QSPI_SCLK
QSPI_SD0
Winbond W25Q128FVSIG
Memoria Flash
C15
XIN
X1
C16
R5
XOUT
Reloj de sistema
3V3
C6
C7
C8
C9
C10
C11
C12
C13
U
VDDIO
USB_VDD
ADC_VDD
LDO_IN
LDO_OUT
VDD_1V1
RUN
USB_D+
USB_D-
XIN
XOUT
SDWCLK
SWDIO
TESTEN
GND_PAD
GPIO0
GPIO1
GPIO2
GPIO3
GPIO4
GPIO5
GPIO6
GPIO7
GPIO8
GPIO9
GPIO10
GPIO11
GPIO12
GPIO13
GPIO14
GPIO15
GPIO16
GPIO17
GPIO18
GPIO19
GPIO20
GPIO21
GPIO22
GPIO23
GPIO24
GPIO25
GPIO26/ADC0
GPIO27/ADC1
GPIO28/ADC2
GPIO29/ADC3
QSPI_SD0
QSPI_SD1
QSPI_SD2
QSPI_SD3
QSPI_SCLK
QSPI_SS
QSPI_/CS
RP2040
PWM0
PWM1
PWM2
PWM3
ADC0
ADC1
ADC2
ADC3
1V1
C1
C2
C3
SB1
R6
R1
R2
U2
SDA / SDI
SCL / SCLK
CS
INT
AD0 / SD0
AUX_DA
AUX_CL
FSYNC
REGOUT
RESV_VDDIO
VDDIO
VDD
RESV_GND
GND
RESV_NC
NC
Invensense MPU-9250
Title
PCB drone
Size
A4
Number
1
Revision
1.0
Date:
7/17/2023
File:
Esquema Principal.SchDoc
Sheet of
Drawn By: Alberto Valls Ortiz

Sistema de control de un UAS mediante el controlador RP2040

Documento 3. Pliego de condiciones

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Trabajo Final de Grado en Ingeniería Electrónica Industrial y Automática

Realizado por: Alberto Valls Ortiz
Tutorizado por: Patricia Balbastre Betoret
Curso Académico: 2022 - 2023

3. Pliego de condiciones

3.1. Objeto

El objeto del pliego de condiciones de este proyecto es establecer los requisitos técnicos, funcionales y operativos necesarios para el diseño, desarrollo e implementación del sistema de control de un UAS utilizando el controlador RP2040. El pliego tiene como objetivo principal definir los parámetros y criterios que deben cumplirse, incluyendo aspectos como la arquitectura del sistema, los componentes de hardware y software requeridos, las capacidades de comunicación y navegación, los protocolos de seguridad, la integración con sensores y actuadores, así como las pruebas y validaciones necesarias. El pliego de condiciones busca garantizar un sistema de control eficiente, confiable y seguro, que cumpla con las necesidades específicas del UAS y permita su operación adecuada en diferentes escenarios y aplicaciones.

3.2. Condiciones de los materiales

3.2.1. Descripción

Todos los materiales a utilizar deberán cumplir los valores establecidos en los planos, así como los estándares propuestos por el documento.

3.2.2. Control de calidad

Para poder asegurar un correcto funcionamiento, los materiales deberán cumplir los siguientes requerimientos:

3.2.2.1 Resistencias

Los valores de las resistencias poseerán una tolerancia del 10 % y estas deberán ser SMD, para su correcta incorporación en la placa.

3.2.2.2 Condensadores

Los valores de los condensadores deberán estar dentro de los valores definidos y su estructura de SMD.

3.2.2.3 Cristal Oscilador

El cristal deberá ser de 12 MHz para su correcto funcionamiento, aunque si se aumenta su frecuencia se puede variar en el código del sistema.

3.2.2.4 RP2040

El controlador RP2040 deberá ser original de la marca de Raspberry, no ser una imitación debido a que puede que no posea las funciones que se necesitan en el sistema.

3.2.2.5 Memoria Flash

La memoria flash utilizada puede ser mayor, ya que el sistema soporta más capacidad de memoria. Se recomienda el uso de Winbond-250128 debido a que es la recomendada por el fabricante y la que mejor se adapta al controlador.

3.2.2.6 Convertidor 3.3 V

El convertidor deberá cumplir el valor prometido de voltaje, la mayoría del tiempo. Además de poder aceptar el valor de la batería externa que se le implante. Para este proyecto el convertidor que se ha implementado puede llegar hasta los 12 V, pero si se requiere aumentar el valor de la fuente de alimentación externa, el convertidor deberá ser consonante a su valor máximo.

3.2.2.7 PCB

La PCB donde se implante el circuito deberá cumplir todos los estándares IPC de las PCB para su correcto funcionamiento.

3.3. Condiciones de la ejecución

3.3.1. Definición

Para la correcta ejecución del sistema, después de haber creado la PCB con los componentes correctos, se deberá implantar el código proporcionado en el sistema mediante la interfaz USB al controlador RP2040.

3.3.2. Control de calidad

3.3.2.1 Version de proyecto

Ya que el código está realizado sobre un prototipo y puede contener errores, el sistema deberá estar actualizado a la última versión proporcionada.

3.3.2.2 Version de firmware

El sistema deberá estar actualizado a la última versión de firmware, ya que esta poseerá corrección de errores referentes a fallos externos al proyecto.

3.4. Pruebas y ajustes finales o de servicio

Para el correcto funcionamiento del sistema, el PID de este deberá ser afinado teniendo en cuenta los tamaños del UAS. Para ello se deberá hacer varias pruebas de vuelo y establecer los valores de los parámetros para que se autoestabilice correctamente. Además, el sistema deberá estar conectado a una terminal desde tierra donde se controle remotamente.

Sistema de control de un UAS mediante el controlador RP2040

Documento 4. Presupuesto

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Trabajo Final de Grado en Ingeniería Electrónica Industrial y Automática

Realizado por: Alberto Valls Ortiz

Tutorizado por: Patricia Balbastre Betoret

Curso Académico: 2022 - 2023

4. Presupuesto

A continuación se muestra una tabla con los componentes utilizados en la construcción del controlador del UAS y sus costes.

Componente	Precio	IVA	Total
RP2040	3,95 €	1,05 €	5 €
MPU9250	10,70 €	2,85 €	13,55 €
5 Resistencias SMD	3,12 €	0,83 €	3,95 €
16 Condensadores SMD	1,9 €	0,5 €	2,4 €
W250128FVSIG	3,2 €	0,85 €	4,05 €
Conector USB	0,87 €	0,23 €	1,10 €
Cristal Oscilador	0,30 €	0,8 €	0,38 €
Convertidor de voltaje	0,62 €	0,17 €	0,79 €
Total	-	-	31,22 €

Cuadro 1: Presupuesto de los componentes

Además, hay que añadir el ensamblaje y el coste de la placa de deposición de materiales, con lo que el coste del controlador es de:

Ítem	Precio	IVA	Total
Componentes	24,66 €	6,56 €	31,22 €
Montaje	16,39 €	4,35 €	20,75 €
Placa de cobre	3,32 €	0,88 €	4,20 €
Total	-	-	56,17 €

Cuadro 2: Presupuesto de la placa ensamblada

En cuanto al coste de los recursos humanos, se estima que se ha dedicado unas 360 horas. De las cuales un porcentaje elevado ha sido de aprendizaje, desarrollo y corrección de errores, además de la experimentación e instalación de software.

De todo ese periodo, se puede abstraer que no todo el tiempo ha sido de trabajo y que solo se ha estado una semana en desarrollo profundo. Entonces, considerando que han sido 37 días de jornada completa de 8 horas y medio día adicional, se puede deducir que han sido 300 horas de trabajo. Así que a 30 euros la hora, explicado en el artículo de Jobted: ."El coste del ingeniero desarrollador se ha extraído del salario medio de los ingenieros industriales, según las estadísticas de Jobted", serían, 12000 euros.

Tras haber evaluado los honorarios del ingeniero, se procede a realizar una estimación del precio del UAS, teniendo en cuenta el montaje y la configuración

del mismo.

	Precio	Horas Invertidas	Total (€)
Honorarios de ingeniero	40 €/h	300	12.000
Materiales	56,17 €	-	56,17
Total	-	-	12.056,17

Cuadro 3: Presupuesto total

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Sistema de control de un UAS mediante el controlador RP2040

Anexos

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Trabajo Final de Grado en Ingeniería Electrónica Industrial y Automática

Realizado por: Alberto Valls Ortiz

Tutorizado por: Patricia Balbastre Betoret

Curso Académico: 2022 - 2023

Source.c

En este documento se incorpora el archivo de source o principal del sistema:

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "pico/stdlib.h"
#include "hardware/gpio.h"
#include "pico/binary_info.h"
#include "hardware/adc.h"
#include "hardware/pwm.h"
#include "icm20948.h"
#include "lps22hb.h"
#include "pid.h"

float transferFunctionReturn(float x, float maxValue, float
    minValue) {
  return (maxValue-minValue)/2 * x;
}

float transferFunctionNoReturn(float x, float maxValue) {
  return maxValue * x;
}


int main(void)
{
  adc_init();
  stdio_init_all();

  IMU_EN_SENSOR_TYPE enMotionSensorType;
  IMU_ST_ANGLES_DATA stAngles;
  IMU_ST_SENSOR_DATA stGyroRawData;
  IMU_ST_SENSOR_DATA stAccelRawData;
  IMU_ST_SENSOR_DATA stMagnRawData;
  float PRESS_DATA = 0;
  float TEMP_DATA = 0;
  uint8_t u8Buf[3];

  PID_PARAMS pitchParams = {0.01, 0.01, 0.01};
  PID_PARAMS rollParams = {0.01, 0.01, 0.01};
  PID_PARAMS yawParams = {0.01, 0.01, 0.01};

  PID_AUX pitchAuxParams = {0.0, 0.0};
  PID_AUX rollAuxParams = {0.0, 0.0};
  PID_AUX yawAuxParams = {0.0, 0.0};

  double rollData = 0.0;
  double pitchData = 0.0;
  double yawData = 0.0;

  double frontRightThruster = 0.5;
  double frontLeftThruster = 0.5;
  double backRightThruster = 0.5;
  double backLeftThruster = 0.5;

```

```
  double rollPoint = 0.0;
  double pitchPoint = 0.0;
  double yawPoint = 0.0;

  float thurst = 0.0;

  imuInit(&enMotionSensorType);
  if (IMU_EN_SENSOR_TYPE_ICM20948 == enMotionSensorType)
  {
    printf("Motion sersor is ICM-20948\n");
  }
  else
  {
    printf("Motion sersor NULL\n");
  }
  if (!LPS22HB_INIT())
  {
    printf("LPS22HB Init Error\n");
    return 0;
  }

  adc_gpio_init(26);
  adc_gpio_init(27);
  adc_gpio_init(28);
  adc_gpio_init(29);

  gpio_set_function(2, GPIO_FUNC_PWM); // Pin 2 en Raspberry Pi
    Pico corresponde a PWM0
    uint slice_num_2 = pwm_gpio_to_slice_num(2);
    pwm_set_wrap(slice_num, 10000); // Periodo de PWM: 10000 ciclos
    pwm_set_chan_level(slice_num_2, PWM_CHAN_A, 0); // Inicializar
    valor de PWM

  gpio_set_function(3, GPIO_FUNC_PWM); // Pin 3 en Raspberry Pi
    Pico corresponde a PWM0
    uint slice_num_3 = pwm_gpio_to_slice_num(3);
    pwm_set_wrap(slice_num, 10000); // Periodo de PWM: 10000 ciclos
    pwm_set_chan_level(slice_num_3, PWM_CHAN_B, 0); // Inicializar
    valor de PWM

  gpio_set_function(4, GPIO_FUNC_PWM); // Pin 4 en Raspberry Pi
    Pico corresponde a PWM0
    uint slice_num_4 = pwm_gpio_to_slice_num(4);
    pwm_set_wrap(slice_num, 10000); // Periodo de PWM: 10000 ciclos
    pwm_set_chan_level(slice_num_4, PWM_CHAN_C, 0); // Inicializar
    valor de PWM

  gpio_set_function(5, GPIO_FUNC_PWM); // Pin 5 en Raspberry Pi
    Pico corresponde a PWM0
    uint slice_num_5 = pwm_gpio_to_slice_num(5);
    pwm_set_wrap(slice_num, 10000); // Periodo de PWM: 10000 ciclos
    pwm_set_chan_level(slice_num_5, PWM_CHAN_D, 0); // Inicializar
    valor de PWM


  while (1)
  {
```

```
    LPS22HB_START_ONESHOT();
    if ((I2C_readByte(LPS_STATUS) & 0x01) == 0x01) // a new
    pressure data is generated
    {
      u8Buf[0] = I2C_readByte(LPS_PRESS_OUT_XL);
      u8Buf[1] = I2C_readByte(LPS_PRESS_OUT_L);
      u8Buf[2] = I2C_readByte(LPS_PRESS_OUT_H);
      PRESS_DATA = (float)((u8Buf[2] << 16) + (u8Buf[1] << 8) +
    u8Buf[0]) / 4096.0f;
    }
    if ((I2C_readByte(LPS_STATUS) & 0x02) == 0x02) // a new
    pressure data is generated
    {
      u8Buf[0] = I2C_readByte(LPS_TEMP_OUT_L);
      u8Buf[1] = I2C_readByte(LPS_TEMP_OUT_H);
      TEMP_DATA = (float)((u8Buf[1] << 8) + u8Buf[0]) / 100.0f;
    }
    imuDataGet(&stAngles, &stGyroRawData, &stAccelRawData, &
    stMagnRawData);

    for (int i = 0; i<4; i++){
      adc_select_input(i);
      adc_value[i] = adc_read()/ 4095.0;
    }

    thrust = transferFunctionNoReturn(adc_value[0], 4095.0);
    rollPoint = transferFunctionReturn(adc_value[1], 2047.5,
    -2047.5);
    pitchPoint = transferFunctionReturn(adc_value[2], 2047.5,
    -2047.5);
    yawPoint = transferFunctionReturn(adc_value[3], 2047.5,
    -2047.5);


    rollData = executePID(rollPoint, stAngles.fRoll, rollParams,
    rollAuxParams);
    pitchData = executePID(pitchPoint, stAngles.fPitch, pitchParams
    , pitchAuxParams);
    yawData = executePID(yawPoint, stAngles.fYaw, yawParams,
    yawAuxParams);

    frontLeftThruster = thurst;
    frontRightThruster = thurst;
    backRightThruster = thurst;
    backLeftThruster = thurst;

    if (pitchData >= 0 ) {
      frontRightThruster *= pitchData;
      frontLeftThruster *= pitchData;
      backRightThruster /= pitchData;
      backLeftThruster /= pitchData;
    } else {
      frontRightThruster /= pitchData;
      frontLeftThruster /= pitchData;
      backRightThruster *= pitchData;
      backLeftThruster *= pitchData;
    }
```

```
149
150
151     if (rollData >= 0 ) {
152       frontRightThruster *= rollData;
153       frontLeftThruster /= rollData;
154       backRightThruster *= rollData;
155       backLeftThruster /= rollData;
156     } else {
157       frontRightThruster /= rollData;
158       frontLeftThruster *= rollData;
159       backRightThruster /= rollData;
160       backLeftThruster *= rollData;
161     }
162
163     if (yawData >= 0 ) {
164       frontRightThruster *= yawData;
165       frontLeftThruster /= yawData;
166       backRightThruster /= yawData;
167       backLeftThruster *= yawData;
168     } else {
169       frontRightThruster /= yawData;
170       frontLeftThruster *= yawData;
171       backRightThruster *= yawData;
172       backLeftThruster /= yawData;
173     }
174
175
176     pwm_set_chan_level(slice_num_2, PWM_CHAN_A, frontLeftThruster);
177     pwm_set_chan_level(slice_num_3, PWM_CHAN_B, frontRightThruster)
        ;
178     pwm_set_chan_level(slice_num_4, PWM_CHAN_C, backLeftThruster);
179     pwm_set_chan_level(slice_num_5, PWM_CHAN_D, backRightThruster);
180
181     sleep_ms(10);
182   }
183   return 0;
184 }
```

Clase del PID

Esta es la clase del PID, donde se crea un encabezado y un cuerpo:

Encabezado

En este documento se inicializan los componentes, las funciones y las macros de la clase:

```
#ifndef PID_H
#define PID_H


#include <math.h>
#define UPPERBOUND 2.0
#define LOWERBOUND 0.0




typedef struct pid_calculation_params
{
    double kProportional;
    double kIntegral;
    double kDerivative;

} PID_PARAMS;

typedef struct pid_last_values
{
    double auxD;
    double auxI;
} PID_AUX;



float executePID(double setPoint, double measuredData, PID_PARAMS
    params, PID_AUX auxiliarParams);
float clamp(double value, double upperBound, double lowerBound);



#endif
```

Código

En el siguiente documento se implementan las funciones y los métodos de la clase PID:

```
#include "pid.h"

float sigmoid(float x, float val)
{
    return val/(1+exp(-x));
}
```

```
7
8
9  float clamp(double value, double upperBound, double lowerBound)
10 {
11     if(value <= lowerBound)
12     {
13         return lowerBound;
14     }
15     else if (value >= upperBound)
16     {
17         return upperBound;
18     }
19     else
20     {
21         return value;
22     }
23 }
24
25 float executePID(double setPoint, double measuredData, PID_PARAMS
       params, PID_AUX auxiliarParams)
26 {
27     float error = setPoint - measuredData;
28
29     float proportional = params.kProportional * error;
30
31     auxiliarParams.auxI += params.kIntegral * error;
32
33     auxiliarParams.auxD = params.kDerivative * (error -
       auxiliarParams.auxD);
34
35     auxiliarParams.auxD = error;
36
37     float output = proportional + auxiliarParams.auxI +
       auxiliarParams.auxD;
38
39     output = sigmoid(output);
40
41     return output;
42
43 }
```

Librerías IMU

Estas son las librerías utilizadas para controlar el IMU:

Clase del ICM20948

En estos documentos se incorporan las macros, clases, métodos y funciones a utilizar por el ICM20948, que es el medidor de inercia del sistema

Encabezado

```
#ifndef __ICM20948__H
#define __ICM20948__H
#include <stdio.h>
#include <math.h>
#include "hardware/i2c.h"
#include "pico/stdlib.h"

//typedef uint8_t bool;
#define true 1
#define false 0
/* define ICM-20948 Device I2C address*/
#define I2C_ADD_ICM20948 0x68
#define I2C_ADD_ICM20948_AK09916 0x0C
#define I2C_ADD_ICM20948_AK09916_READ 0x80
#define I2C_ADD_ICM20948_AK09916_WRITE 0x00
/* define ICM-20948 Register */
/* user bank 0 register */
#define REG_ADD_WIA 0x00
#define REG_VAL_WIA 0xEA
#define REG_ADD_USER_CTRL 0x03
#define REG_VAL_BIT_DMP_EN 0x80
#define REG_VAL_BIT_FIFO_EN 0x40
#define REG_VAL_BIT_I2C_MST_EN 0x20
#define REG_VAL_BIT_I2C_IF_DIS 0x10
#define REG_VAL_BIT_DMP_RST 0x08
#define REG_VAL_BIT_DIAMOND_DMP_RST 0x04
#define REG_ADD_PWR_MIGMT_1 0x06
#define REG_VAL_ALL_RGE_RESET 0x80
#define REG_VAL_RUN_MODE 0x01 //Non low-power mode
#define REG_ADD_LP_CONFIG 0x05
#define REG_ADD_PWR_MGMT_1 0x06
#define REG_ADD_PWR_MGMT_2 0x07
#define REG_ADD_ACCEL_XOUT_H 0x2D
#define REG_ADD_ACCEL_XOUT_L 0x2E
#define REG_ADD_ACCEL_YOUT_H 0x2F
#define REG_ADD_ACCEL_YOUT_L 0x30
#define REG_ADD_ACCEL_ZOUT_H 0x31
#define REG_ADD_ACCEL_ZOUT_L 0x32
#define REG_ADD_GYRO_XOUT_H 0x33
#define REG_ADD_GYRO_XOUT_L 0x34
#define REG_ADD_GYRO_YOUT_H 0x35
#define REG_ADD_GYRO_YOUT_L 0x36
#define REG_ADD_GYRO_ZOUT_H 0x37
#define REG_ADD_GYRO_ZOUT_L 0x38
#define REG_ADD_EXT_SENS_DATA_00 0x3B
#define REG_ADD_REG_BANK_SEL 0x7F
#define REG_VAL_REG_BANK_0 0x00
#define REG_VAL_REG_BANK_1 0x10
#define REG_VAL_REG_BANK_2 0x20
#define REG_VAL_REG_BANK_3 0x30

/* user bank 1 register */
/* user bank 2 register */
#define REG_ADD_GYRO_SMPLRT_DIV 0x00
#define REG_ADD_GYRO_CONFIG_1 0x01
#define REG_VAL_BIT_GYRO_DLPCFG_2 0x10   /* bit[5:3] */
#define REG_VAL_BIT_GYRO_DLPCFG_4 0x20   /* bit[5:3] */
```

```
#define REG_VAL_BIT_GYRO_DLPCFG_6 0x30   /* bit[5:3] */
#define REG_VAL_BIT_GYRO_FS_250DPS 0x00  /* bit[2:1] */
#define REG_VAL_BIT_GYRO_FS_500DPS 0x02  /* bit[2:1] */
#define REG_VAL_BIT_GYRO_FS_1000DPS 0x04 /* bit[2:1] */
#define REG_VAL_BIT_GYRO_FS_2000DPS 0x06 /* bit[2:1] */
#define REG_VAL_BIT_GYRO_DLPF 0x01       /* bit[0]   */
#define REG_ADD_ACCEL_SMPLRT_DIV_2 0x11
#define REG_ADD_ACCEL_CONFIG 0x14
#define REG_VAL_BIT_ACCEL_DLPCFG_2 0x10 /* bit[5:3] */
#define REG_VAL_BIT_ACCEL_DLPCFG_4 0x20 /* bit[5:3] */
#define REG_VAL_BIT_ACCEL_DLPCFG_6 0x30 /* bit[5:3] */
#define REG_VAL_BIT_ACCEL_FS_2g 0x00    /* bit[2:1] */
#define REG_VAL_BIT_ACCEL_FS_4g 0x02    /* bit[2:1] */
#define REG_VAL_BIT_ACCEL_FS_8g 0x04    /* bit[2:1] */
#define REG_VAL_BIT_ACCEL_FS_16g 0x06   /* bit[2:1] */
#define REG_VAL_BIT_ACCEL_DLPF 0x01     /* bit[0]   */

/* user bank 3 register */
#define REG_ADD_I2C_SLV0_ADDR 0x03
#define REG_ADD_I2C_SLV0_REG 0x04
#define REG_ADD_I2C_SLV0_CTRL 0x05
#define REG_VAL_BIT_SLV0_EN 0x80
#define REG_VAL_BIT_MASK_LEN 0x07
#define REG_ADD_I2C_SLV0_DO 0x06
#define REG_ADD_I2C_SLV1_ADDR 0x07
#define REG_ADD_I2C_SLV1_REG 0x08
#define REG_ADD_I2C_SLV1_CTRL 0x09
#define REG_ADD_I2C_SLV1_DO 0x0A

/* define ICM-20948 Register  end */

/* define ICM-20948 MAG Register  */
#define REG_ADD_MAG_WIA1 0x00
#define REG_VAL_MAG_WIA1 0x48
#define REG_ADD_MAG_WIA2 0x01
#define REG_VAL_MAG_WIA2 0x09
#define REG_ADD_MAG_ST2 0x10
#define REG_ADD_MAG_DATA 0x11
#define REG_ADD_MAG_CNTL2 0x31
#define REG_VAL_MAG_MODE_PD 0x00
#define REG_VAL_MAG_MODE_SM 0x01
#define REG_VAL_MAG_MODE_10HZ 0x02
#define REG_VAL_MAG_MODE_20HZ 0x04
#define REG_VAL_MAG_MODE_50HZ 0x05
#define REG_VAL_MAG_MODE_100HZ 0x08
#define REG_VAL_MAG_MODE_ST 0x10
/* define ICM-20948 MAG Register  end */

#define MAG_DATA_LEN    6

#ifdef __cplusplus
extern "C" {
#endif

  typedef enum
  {
      IMU_EN_SENSOR_TYPE_NULL = 0,
```

```
115       IMU_EN_SENSOR_TYPE_ICM20948,
116       IMU_EN_SENSOR_TYPE_MAX
117   }IMU_EN_SENSOR_TYPE;
118
119   typedef struct imu_st_angles_data_tag
120   {
121       float fYaw;
122       float fPitch;
123       float fRoll;
124   }IMU_ST_ANGLES_DATA;
125
126   typedef struct imu_st_sensor_data_tag
127   {
128       int16_t s16X;
129       int16_t s16Y;
130       int16_t s16Z;
131   }IMU_ST_SENSOR_DATA;
132
133   typedef struct icm20948_st_avg_data_tag
134   {
135     uint8_t u8Index;
136     int16_t s16AvgBuffer[8];
137   }ICM20948_ST_AVG_DATA;
138
139   void imuInit(IMU_EN_SENSOR_TYPE *penMotionSensorType);
140   void imuDataGet(IMU_ST_ANGLES_DATA *pstAngles,
141           IMU_ST_SENSOR_DATA *pstGyroRawData,
142           IMU_ST_SENSOR_DATA *pstAccelRawData,
143           IMU_ST_SENSOR_DATA *pstMagnRawData);
144   char I2C_ReadOneByte(char reg);
145   void I2C_WriteOneByte(char reg, char val);
146
147 #ifdef __cplusplus
148 }
149 #endif
150
151 #endif
152
```

Código

```
1  #include "icm20948.h"
2  #include <string.h>
3
4  #define I2C_PORT i2c1
5  IMU_ST_SENSOR_DATA gstGyroOffset ={0,0,0};
6  #ifdef __cplusplus
7  extern "C" {
8  #endif
9
10 void imuAHRSupdate(float gx, float gy, float gz, float ax, float ay
       , float az, float mx, float my, float mz);
11 float invSqrt(float x);
12
13 void icm20948init(void);
14 bool icm20948Check(void);
15 void icm20948GyroRead(int16_t* ps16X, int16_t* ps16Y, int16_t*
```

```
    ps16Z);
void icm20948AccelRead(int16_t* ps16X, int16_t* ps16Y, int16_t*
    ps16Z);
void icm20948MagRead(int16_t* ps16X, int16_t* ps16Y, int16_t* ps16Z
    );
bool icm20948MagCheck(void);
void icm20948CalAvgValue(uint8_t *pIndex, int16_t *pAvgBuffer,
    int16_t InVal, int32_t *pOutVal);
void icm20948GyroOffset(void);
void icm20948ReadSecondary(uint8_t u8I2CAddr, uint8_t u8RegAddr,
    uint8_t u8Len, uint8_t *pu8data);
void icm20948WriteSecondary(uint8_t u8I2CAddr, uint8_t u8RegAddr,
    uint8_t u8data);
bool icm20948Check(void);

char I2C_ReadOneByte(char reg)
{
  char buf;
  i2c_write_blocking(I2C_PORT,I2C_ADD_ICM20948,&reg,1,true);
  i2c_read_blocking(I2C_PORT,I2C_ADD_ICM20948,&buf,1,false);
  return buf;
}

void I2C_WriteOneByte( char reg, char value)
{
  char buf[]={reg,value};
  i2c_write_blocking(I2C_PORT,I2C_ADD_ICM20948,buf,2,false);
}

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

 * IMU module
              *

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

#define Kp 4.50f   // proportional gain governs rate of convergence
     to accelerometer/magnetometer
#define Ki 1.0f    // integral gain governs rate of convergence of
    gyroscope biases

float angles[3];
float q0, q1, q2, q3;

void imuInit(IMU_EN_SENSOR_TYPE *penMotionSensorType)
{
  bool bRet = false;
  i2c_init(I2C_PORT,400*1000);
  gpio_set_function(6, GPIO_FUNC_I2C);
    gpio_set_function(7, GPIO_FUNC_I2C);
    gpio_pull_up(6);
    gpio_pull_up(7);
  bRet = icm20948Check();
  if( true == bRet)
  {
    *penMotionSensorType = IMU_EN_SENSOR_TYPE_ICM20948;
    icm20948init();
```

```
  }
  else
  {
    *penMotionSensorType = IMU_EN_SENSOR_TYPE_NULL;
  }
  q0 = 1.0f;
  q1 = 0.0f;
  q2 = 0.0f;
  q3 = 0.0f;

  return;
}
void imuDataGet(IMU_ST_ANGLES_DATA *pstAngles,
                IMU_ST_SENSOR_DATA *pstGyroRawData,
                IMU_ST_SENSOR_DATA *pstAccelRawData,
                IMU_ST_SENSOR_DATA *pstMagnRawData)
{
  float MotionVal[9];
  int16_t s16Gyro[3], s16Accel[3], s16Magn[3];

  icm20948AccelRead(&s16Accel[0], &s16Accel[1], &s16Accel[2]);
  icm20948GyroRead(&s16Gyro[0], &s16Gyro[1], &s16Gyro[2]);
  icm20948MagRead(&s16Magn[0], &s16Magn[1], &s16Magn[2]);

  MotionVal[0]=s16Gyro[0]/32.8;
  MotionVal[1]=s16Gyro[1]/32.8;
  MotionVal[2]=s16Gyro[2]/32.8;
  MotionVal[3]=s16Accel[0];
  MotionVal[4]=s16Accel[1];
  MotionVal[5]=s16Accel[2];
  MotionVal[6]=s16Magn[0];
  MotionVal[7]=s16Magn[1];
  MotionVal[8]=s16Magn[2];
  imuAHRSupdate((float)MotionVal[0] * 0.0175, (float)MotionVal[1] *
     0.0175, (float)MotionVal[2] * 0.0175,
                (float)MotionVal[3], (float)MotionVal[4], (float)
    MotionVal[5],
                (float)MotionVal[6], (float)MotionVal[7], MotionVal
    [8]);


  pstAngles->fPitch = asin(-2 * q1 * q3 + 2 * q0* q2)* 57.3; //
    pitch
  pstAngles->fRoll = atan2(2 * q2 * q3 + 2 * q0 * q1, -2 * q1 * q1
    - 2 * q2* q2 + 1)* 57.3; // roll
  pstAngles->fYaw = atan2(-2 * q1 * q2 - 2 * q0 * q3, 2 * q2 * q2 +
     2 * q3 * q3 - 1) * 57.3;

  pstGyroRawData->s16X = s16Gyro[0];
  pstGyroRawData->s16Y = s16Gyro[1];
  pstGyroRawData->s16Z = s16Gyro[2];

  pstAccelRawData->s16X = s16Accel[0];
  pstAccelRawData->s16Y = s16Accel[1];
  pstAccelRawData->s16Z  = s16Accel[2];

  pstMagnRawData->s16X = s16Magn[0];
```

```
112   pstMagnRawData->s16Y = s16Magn[1];
113   pstMagnRawData->s16Z = s16Magn[2];
114
115   return;
116 }
117
118 void imuAHRSupdate(float gx, float gy, float gz, float ax, float ay
        , float az, float mx, float my, float mz)
119 {
120   float norm;
121   float hx, hy, hz, bx, bz;
122   float vx, vy, vz, wx, wy, wz;
123   float exInt = 0.0, eyInt = 0.0, ezInt = 0.0;
124   float ex, ey, ez, halfT = 0.024f;
125
126   float q0q0 = q0 * q0;
127   float q0q1 = q0 * q1;
128   float q0q2 = q0 * q2;
129   float q0q3 = q0 * q3;
130   float q1q1 = q1 * q1;
131   float q1q2 = q1 * q2;
132   float q1q3 = q1 * q3;
133   float q2q2 = q2 * q2;
134   float q2q3 = q2 * q3;
135   float q3q3 = q3 * q3;
136
137   norm = invSqrt(ax * ax + ay * ay + az * az);
138   ax = ax * norm;
139   ay = ay * norm;
140   az = az * norm;
141
142   norm = invSqrt(mx * mx + my * my + mz * mz);
143   mx = mx * norm;
144   my = my * norm;
145   mz = mz * norm;
146
147   // compute reference direction of flux
148   hx = 2 * mx * (0.5f - q2q2 - q3q3) + 2 * my * (q1q2 - q0q3) + 2 *
        mz * (q1q3 + q0q2);
149   hy = 2 * mx * (q1q2 + q0q3) + 2 * my * (0.5f - q1q1 - q3q3) + 2 *
        mz * (q2q3 - q0q1);
150   hz = 2 * mx * (q1q3 - q0q2) + 2 * my * (q2q3 + q0q1) + 2 * mz *
       (0.5f - q1q1 - q2q2);
151   bx = sqrt((hx * hx) + (hy * hy));
152   bz = hz;
153
154   // estimated direction of gravity and flux (v and w)
155   vx = 2 * (q1q3 - q0q2);
156   vy = 2 * (q0q1 + q2q3);
157   vz = q0q0 - q1q1 - q2q2 + q3q3;
158   wx = 2 * bx * (0.5 - q2q2 - q3q3) + 2 * bz * (q1q3 - q0q2);
159   wy = 2 * bx * (q1q2 - q0q3) + 2 * bz * (q0q1 + q2q3);
160   wz = 2 * bx * (q0q2 + q1q3) + 2 * bz * (0.5 - q1q1 - q2q2);
161
162   // error is sum of cross product between reference direction of
       fields and direction measured by sensors
163   ex = (ay * vz - az * vy) + (my * wz - mz * wy);
```

```
  ey = (az * vx - ax * vz) + (mz * wx - mx * wz);
  ez = (ax * vy - ay * vx) + (mx * wy - my * wx);

  if(ex != 0.0f && ey != 0.0f && ez != 0.0f)
  {
    exInt = exInt + ex * Ki * halfT;
    eyInt = eyInt + ey * Ki * halfT;
    ezInt = ezInt + ez * Ki * halfT;

    gx = gx + Kp * ex + exInt;
    gy = gy + Kp * ey + eyInt;
    gz = gz + Kp * ez + ezInt;
  }

  q0 = q0 + (-q1 * gx - q2 * gy - q3 * gz) * halfT;
  q1 = q1 + (q0 * gx + q2 * gz - q3 * gy) * halfT;
  q2 = q2 + (q0 * gy - q1 * gz + q3 * gx) * halfT;
  q3 = q3 + (q0 * gz + q1 * gy - q2 * gx) * halfT;

  norm = invSqrt(q0 * q0 + q1 * q1 + q2 * q2 + q3 * q3);
  q0 = q0 * norm;
  q1 = q1 * norm;
  q2 = q2 * norm;
  q3 = q3 * norm;
}

float invSqrt(float x)
{
  float halfx = 0.5f * x;
  float y = x;

  long i = *(long*)&y;                //get bits for floating value
  i = 0x5f3759df - (i >> 1);          //gives initial guss you
  y = *(float*)&i;                    //convert bits back to float
  y = y * (1.5f - (halfx * y * y));   //newtop step, repeating
    increases accuracy

  return y;
}

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

 * icm20948 sensor device
              *

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

void icm20948init(void)
{

  /* user bank 0 register */
  I2C_WriteOneByte( REG_ADD_REG_BANK_SEL, REG_VAL_REG_BANK_0);
  I2C_WriteOneByte( REG_ADD_PWR_MIGMT_1,  REG_VAL_ALL_RGE_RESET);
  sleep_ms(10);
  I2C_WriteOneByte( REG_ADD_PWR_MIGMT_1,  REG_VAL_RUN_MODE);

  /* user bank 2 register */
```

```
  I2C_WriteOneByte( REG_ADD_REG_BANK_SEL, REG_VAL_REG_BANK_2);
  I2C_WriteOneByte( REG_ADD_GYRO_SMPLRT_DIV, 0x07);
  I2C_WriteOneByte( REG_ADD_GYRO_CONFIG_1,
                  REG_VAL_BIT_GYRO_DLPCFG_6 |
    REG_VAL_BIT_GYRO_FS_1000DPS | REG_VAL_BIT_GYRO_DLPF);
  I2C_WriteOneByte( REG_ADD_ACCEL_SMPLRT_DIV_2,  0x07);
  I2C_WriteOneByte( REG_ADD_ACCEL_CONFIG,
                  REG_VAL_BIT_ACCEL_DLPCFG_6 |
    REG_VAL_BIT_ACCEL_FS_2g | REG_VAL_BIT_ACCEL_DLPF);

  /* user bank 0 register */
  I2C_WriteOneByte( REG_ADD_REG_BANK_SEL, REG_VAL_REG_BANK_0);

  sleep_ms(100);
  /* offset */
  icm20948GyroOffset();

  icm20948MagCheck();

  icm20948WriteSecondary( I2C_ADD_ICM20948_AK09916|
    I2C_ADD_ICM20948_AK09916_WRITE,
                              REG_ADD_MAG_CNTL2,
    REG_VAL_MAG_MODE_20HZ);
  return;
}

bool icm20948Check(void)
{
    bool bRet = false;
    if(REG_VAL_WIA == I2C_ReadOneByte( REG_ADD_WIA))
    {
        bRet = true;
    }
    return bRet;
}
void icm20948GyroRead(int16_t* ps16X, int16_t* ps16Y, int16_t*
    ps16Z)
{
   uint8_t u8Buf[6];
    int16_t s16Buf[3] = {0};
    uint8_t i;
    int32_t s32OutBuf[3] = {0};
    static ICM20948_ST_AVG_DATA sstAvgBuf[3];
    static int16_t ss16c = 0;
    ss16c++;

    u8Buf[0]=I2C_ReadOneByte(REG_ADD_GYRO_XOUT_L);
    u8Buf[1]=I2C_ReadOneByte(REG_ADD_GYRO_XOUT_H);
    s16Buf[0]=  (u8Buf[1]<<8)|u8Buf[0];

    u8Buf[0]=I2C_ReadOneByte(REG_ADD_GYRO_YOUT_L);
    u8Buf[1]=I2C_ReadOneByte(REG_ADD_GYRO_YOUT_H);
    s16Buf[1]=  (u8Buf[1]<<8)|u8Buf[0];

    u8Buf[0]=I2C_ReadOneByte(REG_ADD_GYRO_ZOUT_L);
    u8Buf[1]=I2C_ReadOneByte(REG_ADD_GYRO_ZOUT_H);
    s16Buf[2]=  (u8Buf[1]<<8)|u8Buf[0];
```

```
268
269     for(i = 0; i < 3; i ++)
270     {
271         icm20948CalAvgValue(&sstAvgBuf[i].u8Index, sstAvgBuf[i].
        s16AvgBuffer, s16Buf[i], s32OutBuf + i);
272     }
273     *ps16X = s32OutBuf[0] - gstGyroOffset.s16X;
274     *ps16Y = s32OutBuf[1] - gstGyroOffset.s16Y;
275     *ps16Z = s32OutBuf[2] - gstGyroOffset.s16Z;
276
277     return;
278 }
279 void icm20948AccelRead(int16_t* ps16X, int16_t* ps16Y, int16_t*
        ps16Z)
280 {
281     uint8_t u8Buf[2];
282     int16_t s16Buf[3] = {0};
283     uint8_t i;
284     int32_t s32OutBuf[3] = {0};
285     static ICM20948_ST_AVG_DATA sstAvgBuf[3];
286
287     u8Buf[0]=I2C_ReadOneByte(REG_ADD_ACCEL_XOUT_L);
288     u8Buf[1]=I2C_ReadOneByte(REG_ADD_ACCEL_XOUT_H);
289     s16Buf[0]=  (u8Buf[1]<<8)|u8Buf[0];
290
291     u8Buf[0]=I2C_ReadOneByte(REG_ADD_ACCEL_YOUT_L);
292     u8Buf[1]=I2C_ReadOneByte(REG_ADD_ACCEL_YOUT_H);
293     s16Buf[1]=  (u8Buf[1]<<8)|u8Buf[0];
294
295     u8Buf[0]=I2C_ReadOneByte(REG_ADD_ACCEL_ZOUT_L);
296     u8Buf[1]=I2C_ReadOneByte(REG_ADD_ACCEL_ZOUT_H);
297     s16Buf[2]=  (u8Buf[1]<<8)|u8Buf[0];
298
299     for(i = 0; i < 3; i ++)
300     {
301         icm20948CalAvgValue(&sstAvgBuf[i].u8Index, sstAvgBuf[i].
        s16AvgBuffer, s16Buf[i], s32OutBuf + i);
302     }
303     *ps16X = s32OutBuf[0];
304     *ps16Y = s32OutBuf[1];
305     *ps16Z = s32OutBuf[2];
306
307     return;
308
309 }
310 void icm20948MagRead(int16_t* ps16X, int16_t* ps16Y, int16_t* ps16Z
        )
311 {
312     uint8_t counter = 20;
313     uint8_t u8Data[MAG_DATA_LEN];
314     int16_t s16Buf[3] = {0};
315     uint8_t i;
316     int32_t s32OutBuf[3] = {0};
317     static ICM20948_ST_AVG_DATA sstAvgBuf[3];
318     while( counter>0 )
319     {
320         sleep_ms(10);
```

```
321         icm20948ReadSecondary( I2C_ADD_ICM20948_AK09916|
    I2C_ADD_ICM20948_AK09916_READ,
322                                    REG_ADD_MAG_ST2, 1, u8Data);
323 
324         if ((u8Data[0] & 0x01) != 0)
325             break;
326 
327         counter--;
328     }
329 
330     if(counter != 0)
331     {
332         icm20948ReadSecondary( I2C_ADD_ICM20948_AK09916|
    I2C_ADD_ICM20948_AK09916_READ,
333                                    REG_ADD_MAG_DATA,
334                                    MAG_DATA_LEN,
335                                    u8Data);
336         s16Buf[0] = ((int16_t)u8Data[1]<<8) | u8Data[0];
337         s16Buf[1] = ((int16_t)u8Data[3]<<8) | u8Data[2];
338         s16Buf[2] = ((int16_t)u8Data[5]<<8) | u8Data[4];
339     }
340 
341     for(i = 0; i < 3; i ++)
342     {
343         icm20948CalAvgValue(&sstAvgBuf[i].u8Index, sstAvgBuf[i].
    s16AvgBuffer, s16Buf[i], s32OutBuf + i);
344     }
345 
346     *ps16X =  s32OutBuf[0];
347     *ps16Y = -s32OutBuf[1];
348     *ps16Z = -s32OutBuf[2];
349     return;
350 }
351 
352 void icm20948ReadSecondary(uint8_t u8I2CAddr, uint8_t u8RegAddr,
    uint8_t u8Len, uint8_t *pu8data)
353 {
354     uint8_t i;
355     uint8_t u8Temp;
356 
357     I2C_WriteOneByte( REG_ADD_REG_BANK_SEL,  REG_VAL_REG_BANK_3);
    //swtich bank3
358     I2C_WriteOneByte( REG_ADD_I2C_SLV0_ADDR, u8I2CAddr);
359     I2C_WriteOneByte( REG_ADD_I2C_SLV0_REG,  u8RegAddr);
360     I2C_WriteOneByte( REG_ADD_I2C_SLV0_CTRL, REG_VAL_BIT_SLV0_EN|
    u8Len);
361 
362     I2C_WriteOneByte( REG_ADD_REG_BANK_SEL, REG_VAL_REG_BANK_0); //
    swtich bank0
363 
364     u8Temp = I2C_ReadOneByte(REG_ADD_USER_CTRL);
365     u8Temp |= REG_VAL_BIT_I2C_MST_EN;
366     I2C_WriteOneByte( REG_ADD_USER_CTRL, u8Temp);
367     sleep_ms(5);
368     u8Temp &= ~REG_VAL_BIT_I2C_MST_EN;
369     I2C_WriteOneByte( REG_ADD_USER_CTRL, u8Temp);
370 
```

```
371     for(i=0; i<u8Len; i++)
372     {
373         *(pu8data+i) = I2C_ReadOneByte( REG_ADD_EXT_SENS_DATA_00+i)
        ;
374 
375     }
376     I2C_WriteOneByte( REG_ADD_REG_BANK_SEL, REG_VAL_REG_BANK_3); //
        swtich bank3
377 
378     u8Temp = I2C_ReadOneByte(REG_ADD_I2C_SLV0_CTRL);
379     u8Temp &= ~((REG_VAL_BIT_I2C_MST_EN)&(REG_VAL_BIT_MASK_LEN));
380     I2C_WriteOneByte( REG_ADD_I2C_SLV0_CTRL,  u8Temp);
381 
382     I2C_WriteOneByte( REG_ADD_REG_BANK_SEL, REG_VAL_REG_BANK_0); //
        swtich bank0
383 
384 }
385 
386 void icm20948WriteSecondary(uint8_t u8I2CAddr, uint8_t u8RegAddr,
        uint8_t u8data)
387 {
388   uint8_t u8Temp;
389   I2C_WriteOneByte( REG_ADD_REG_BANK_SEL,  REG_VAL_REG_BANK_3); //
        swtich bank3
390   I2C_WriteOneByte( REG_ADD_I2C_SLV1_ADDR, u8I2CAddr);
391   I2C_WriteOneByte( REG_ADD_I2C_SLV1_REG,  u8RegAddr);
392   I2C_WriteOneByte( REG_ADD_I2C_SLV1_DO,   u8data);
393   I2C_WriteOneByte( REG_ADD_I2C_SLV1_CTRL, REG_VAL_BIT_SLV0_EN|1);
394 
395   I2C_WriteOneByte( REG_ADD_REG_BANK_SEL, REG_VAL_REG_BANK_0); //
        swtich bank0
396 
397   u8Temp = I2C_ReadOneByte(REG_ADD_USER_CTRL);
398   u8Temp |= REG_VAL_BIT_I2C_MST_EN;
399   I2C_WriteOneByte( REG_ADD_USER_CTRL, u8Temp);
400   sleep_ms(5);
401   u8Temp &= ~REG_VAL_BIT_I2C_MST_EN;
402   I2C_WriteOneByte( REG_ADD_USER_CTRL, u8Temp);
403 
404   I2C_WriteOneByte( REG_ADD_REG_BANK_SEL, REG_VAL_REG_BANK_3); //
        swtich bank3
405 
406   u8Temp = I2C_ReadOneByte(REG_ADD_I2C_SLV0_CTRL);
407   u8Temp &= ~((REG_VAL_BIT_I2C_MST_EN)&(REG_VAL_BIT_MASK_LEN));
408   I2C_WriteOneByte( REG_ADD_I2C_SLV0_CTRL,  u8Temp);
409 
410   I2C_WriteOneByte( REG_ADD_REG_BANK_SEL, REG_VAL_REG_BANK_0); //
        swtich bank0
411 
412     return;
413 }
414 
415 void icm20948CalAvgValue(uint8_t *pIndex, int16_t *pAvgBuffer,
        int16_t InVal, int32_t *pOutVal)
416 {
417   uint8_t i;
418 
```

```
  *(pAvgBuffer + ((*pIndex) ++)) = InVal;
    *pIndex &= 0x07;

    *pOutVal = 0;
  for(i = 0; i < 8; i ++)
    {
      *pOutVal += *(pAvgBuffer + i);
    }
    *pOutVal >>= 3;
}

void icm20948GyroOffset(void)
{
  uint8_t i;
  int16_t s16Gx = 0, s16Gy = 0, s16Gz = 0;
  int32_t s32TempGx = 0, s32TempGy = 0, s32TempGz = 0;
  for(i = 0; i < 32; i ++)
  {
    icm20948GyroRead(&s16Gx, &s16Gy, &s16Gz);
    s32TempGx += s16Gx;
    s32TempGy += s16Gy;
    s32TempGz += s16Gz;
    sleep_ms(10);
  }
  gstGyroOffset.s16X = s32TempGx >> 5;
  gstGyroOffset.s16Y = s32TempGy >> 5;
  gstGyroOffset.s16Z = s32TempGz >> 5;
  return;
}

bool icm20948MagCheck(void)
{
    bool bRet = false;
    uint8_t u8Ret[2];

    icm20948ReadSecondary( I2C_ADD_ICM20948_AK09916|
    I2C_ADD_ICM20948_AK09916_READ,
                                REG_ADD_MAG_WIA1, 2,u8Ret);
    if( (u8Ret[0] == REG_VAL_MAG_WIA1) && ( u8Ret[1] ==
    REG_VAL_MAG_WIA2) )
    {
        bRet = true;
    }

    return bRet;
}

#ifdef __cplusplus
}
#endif
```

Clase del LPS22HB

En estos documentos se incorporan las macros, clases, métodos y funciones a utilizar por el LPS22HB, que es el sensor de presión del sistema.

Encabezado

```
#ifndef _LPS22HB_H
#define _LPS22HB_H

#include "hardware/i2c.h"
#include "pico/stdlib.h"

//i2c address
#define LPS22HB_I2C_ADDRESS   0x5C
//
#define LPS_ID                  0xB1
//Register
#define LPS_INT_CFG             0x0B        //Interrupt register
#define LPS_THS_P_L             0x0C        //Pressure threshold
    registers
#define LPS_THS_P_H             0x0D
#define LPS_WHO_AM_I            0x0F        //Who am I
#define LPS_CTRL_REG1           0x10        //Control registers
#define LPS_CTRL_REG2           0x11
#define LPS_CTRL_REG3           0x12
#define LPS_FIFO_CTRL           0x14        //FIFO configuration
    register
#define LPS_REF_P_XL            0x15        //Reference pressure
    registers
#define LPS_REF_P_L             0x16
#define LPS_REF_P_H             0x17
#define LPS_RPDS_L              0x18        //Pressure offset
    registers
#define LPS_RPDS_H              0x19
#define LPS_RES_CONF            0x1A        //Resolution register
#define LPS_INT_SOURCE          0x25        //Interrupt register
#define LPS_FIFO_STATUS         0x26        //FIFO status register
#define LPS_STATUS              0x27        //Status register
#define LPS_PRESS_OUT_XL        0x28        //Pressure output
    registers
#define LPS_PRESS_OUT_L         0x29
#define LPS_PRESS_OUT_H         0x2A
#define LPS_TEMP_OUT_L          0x2B        //Temperature output
    registers
#define LPS_TEMP_OUT_H          0x2C
#define LPS_RES                 0x33        //Filter reset register


char I2C_readByte(char reg);
unsigned short I2C_readU16(char reg);
void I2C_writeByte(char reg, char val);
void LPS22HB_RESET();
void LPS22HB_START_ONESHOT();
uint8_t LPS22HB_INIT();
#endif
```

Código

```
#include <stdio.h>
#include <math.h>
#include "lps22hb.h"
```

```
char I2C_readByte(char reg)
{
  char buf[] = { reg };
  i2c_write_blocking(i2c1,LPS22HB_I2C_ADDRESS,&reg,1,true);
  i2c_read_blocking(i2c1,LPS22HB_I2C_ADDRESS,buf,1,false);
  return buf[0];
}
unsigned short I2C_readU16(char reg)
{
  char buf[] = { reg,0 };
  i2c_write_blocking(i2c1,LPS22HB_I2C_ADDRESS,&reg,1,true);
  i2c_read_blocking(i2c1,LPS22HB_I2C_ADDRESS,buf,2,false);
  int value = buf[1] * 0x100 + buf[0];
  return value;
}
void I2C_writeByte(char reg, char val)
{
  char buf[] = { reg,val };
  i2c_write_blocking(i2c1,LPS22HB_I2C_ADDRESS,&buf,2,false);
}
void LPS22HB_RESET()
{   uint8_t Buf;
    Buf=I2C_readU16(LPS_CTRL_REG2);
    Buf|=0x04;
    I2C_writeByte(LPS_CTRL_REG2,Buf);                  //SWRESET
    Set 1
    while(Buf)
    {
        Buf=I2C_readU16(LPS_CTRL_REG2);
        Buf&=0x04;
    }
}
void LPS22HB_START_ONESHOT() {
    uint8_t Buf;
    Buf=I2C_readU16(LPS_CTRL_REG2);
    Buf|=0x01;                                         //ONE_SHOT
    Set 1
    I2C_writeByte(LPS_CTRL_REG2,Buf);
}
uint8_t LPS22HB_INIT()
{
    if(I2C_readByte(LPS_WHO_AM_I)!=LPS_ID) return 0;    //Check
    device ID
    LPS22HB_RESET();                                    //Wait for
    reset to complete
    I2C_writeByte(LPS_CTRL_REG1 ,   0x02);              //Low-pass
    filter disabled , output registers not updated until MSB and
    LSB have been read , Enable Block Data Update , Set Output Data
     Rate to 0
    return 1;
}

//int main()
//{   float PRESS_DATA=0;
//    float TEMP_DATA=0;
//    uint8_t u8Buf[3];
//    if (!bcm2835_init()) return 1;
```

```
//    printf("\nPressure Sensor Test Program ...\n");
//    if(!LPS22HB_INIT())
//    {
//        printf("\nPressure Sensor Error\n");
//        return 0;
//    }
//    printf("\nPressure Sensor OK\n");
//    while(1)
//    {
//        LPS22HB_START_ONESHOT();        //Trigger one shot data acquisition
//        if((I2C_readByte(LPS_STATUS)&0x01)==0x01)   //a new pressure data is generated
//        {
//            u8Buf[0]=I2C_readByte(LPS_PRESS_OUT_XL);
//            u8Buf[1]=I2C_readByte(LPS_PRESS_OUT_L);
//            u8Buf[2]=I2C_readByte(LPS_PRESS_OUT_H);
//            PRESS_DATA=(float)((u8Buf[2]<<16)+(u8Buf[1]<<8)+u8Buf[0])/4096.0f;
//        }
//        if((I2C_readByte(LPS_STATUS)&0x02)==0x02)   // a new pressure data is generated
//        {
//            u8Buf[0]=I2C_readByte(LPS_TEMP_OUT_L);
//            u8Buf[1]=I2C_readByte(LPS_TEMP_OUT_H);
//            TEMP_DATA=(float)((u8Buf[1]<<8)+u8Buf[0])/100.0f;
//        }
//
//       printf("Pressure = %6.2f hPa , Temperature = %6.2f  C \r\n", PRESS_DATA, TEMP_DATA);
//    }
//    return 0;
//}
```

Con estos dos componentes, se puede crear el sistema IMU.