UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Manufactura, modelado y control de un cuadricóptero con comunicación inalámbrica Wi-Fi integrado al ecosistema Robotat

Trabajo de graduación presentado por Carlos Antonio Alonzo Martínez para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2022

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Manufactura, modelado y control de un cuadricóptero con comunicación inalámbrica Wi-Fi integrado al ecosistema Robotat

Trabajo de graduación presentado por Carlos Antonio Alonzo Martínez para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

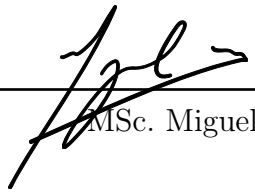
2022

Vo.Bo.:

(f) 

MSc. Miguel Zea

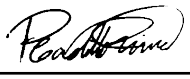
Tribunal Examinador:

(f) 

MSc. Miguel Zea

(f) 

Ing. Luis Jose Pinillos Motta

(f) 

MSc. Pedro Ivan Castillo Rivera

Fecha de aprobación: Guatemala, 06 de enero de 2022.

Prefacio

Durante mi tiempo en la Universidad del Valle aprendí muchas cosas, participé y trabajé en proyectos interesantes, me divertí, conocí personas increíbles y coleccioné momentos que recordaré siempre. Agradezco a todas las personas que contribuyeron para cada una de ellas.

Quiero agradecer a toda mi familia, quienes aportaron cada uno de forma diferente algo para que haya tenido una gran experiencia en la universidad. Principalmente, quiero agradecer a mis padres, Juan Carlos y Helen, quienes son mi mayor ejemplo en la vida. Me enseñaron siempre a dar lo mejor de mí y me dieron la oportunidad de escoger y descubrir mi camino para cumplir mis metas. A mis hermanos, Andrea y Cristian, quienes me impulsan a mejorar cada día, aprender de ellos y que deseo ser un ejemplo de que se puede alcanzar cualquier cosa que te propongas.

De igual manera, quiero agradecer a mi asesor de tesis, Miguel Zea, quien compartió su conocimiento y consejos conmigo, para poder así concretar este proyecto exitosamente. A mis amigos y compañeros de carrera, les deseo éxito, agradezco por su ayuda y los buenos momentos que pasamos durante los 5 años.

Índice

Lista de figuras

Lista de cuadros

Resumen

En este trabajo se planteó el desarrollo de un vehículo volador no tripulado de cuatro motores que pueda ser utilizado para el diseño y evaluación de controladores de vuelo dentro del ecosistema Robotat del laboratorio de Robótica de la Universidad del Valle de Guatemala. Para el desarrollo del dispositivo se planteó el bajo costo económico y la accesibilidad de todos los componentes de forma local (Guatemala) como las características más importantes. El trabajo se dividió en tres partes principales. En la primera parte se describe el proceso de selección de los motores sin escobillas, controladores electrónicos de velocidad y hélices, así como el dimensionamiento utilizado para generar un marco estructural capaz de soportar el peso de los componentes y asegurarlos sin deformarse significativamente.

En la segunda parte se detalla el entorno de simulación desarrollado con la herramienta de MATLAB, y la forma de interactuar con los archivos para que el usuario pueda editar de forma sencilla el controlador que se desea evaluar en la simulación. Asimismo, se presenta el controlador proporcional-derivativo utilizado para mantener el drone en estado de *hovering* y su implementación en el microcontrolador ESP32 utilizando lenguaje C. Para validar el comportamiento del controlador se utilizaron tres pruebas de vuelo diferentes, incluyendo una serie de gráficos que muestran la respuesta de velocidad de los cuatro motores respecto a los ángulos de entrada de una Unidad de Medición Inercial.

Por último, se realizó la integración entre el sistema de captura de movimiento OptiTrack y el drone para recibir información en tiempo real de la posición y orientación dentro del espacio que cubre el sistema.

Cada una de las partes se desarrolló con la investigación previa, el desarrollo e implementación, y finalmente un proceso de validación de resultados comparados con estándares fijados al inicio.

Abstract

In this document, the development of a quadrotor unmanned flying vehicle that can be used for the design and evaluation of flight controllers within the Robotat ecosystem of the Robotics laboratory of the Universidad del Valle de Guatemala is proposed. For the development of the device, the low economic cost and the accessibility of all the components locally (Guatemala) were considered as the most important characteristics. The work was divided into three main parts. The first part describes the selection process for brushless motors, electronic speed controllers and propellers, as well as the sizing used to generate a structural framework capable of supporting the weight of the components and securing them without significantly deforming.

The second part details the simulation environment developed with the MATLAB tool, and how to interact with the files so that the user can easily edit the controller to be evaluated during the simulation. Likewise, the proportional-derivative controller used to keep the drone hovering and its implementation in the ESP32 microcontroller using C language is presented. To validate the behavior of the controller, a series of graphs were used that show the speed response of the four motors with respect to the entry angles of an Inertial Measurement Unit.

Finally, the integration between the OptiTrack motion capture system and the drone was carried out to receive real-time information on the position and orientation within the space that the system covers.

Each of the parts was developed with prior research, development and implementation, and finally a process of validation of results compared to standards set at the beginning.

CAPÍTULO 1

Introducción

Durante las últimas décadas, los vehículos aéreos no tripulados (UAV) han cobrado gran importancia en industrias fuera del sector militar. Goldman Sach Research pronosticó que el mercado para estos vehículos superaría los 100 mil millones de dólares para 2020, donde una parte importante de los consumidores serían del sector de construcción y agricultura [1]. Esto provocó que se invirtiese en la investigación, desarrollo y manufactura de estos dispositivos. Una gran ventaja de los UAVs con respecto a vehículos tripulados como helicópteros, es que por sus significativamente más pequeñas dimensiones son vehículos capaces de maniobrar en espacios restringidos, ya sea en exteriores o interiores. Además, presentan una mayor facilidad de navegación en lugares donde vehículos robóticos terrestres no tienen acceso.

Este trabajo de investigación se centró en el desarrollo, modelado y control de un cuadricóptero de bajo costo económico, el cual pueda ser utilizado para investigación dentro del ecosistema Robotat de la Universidad del Valle de Guatemala. Debido a la complejidad del trabajo, este se dividió en dos áreas. Una trabajada por Hans Burmester y la otra por el autor este trabajo. A su vez, el área investigada en este documento se separó en tres etapas. La primera, Capítulo 7, consistió en el diseño, análisis y manufactura del marco estructural poniendo particular atención en que los componentes sean accesibles localmente. La segunda, Capítulo 8, se enfocó en la determinación del modelo dinámico del dispositivo, el cual emplea información de una Unidad de Medición Inercial y motores sin escobillas para modificar la orientación del drone. Por último, Capítulo 9, se integró el multirotor al sistema de captura de movimiento OptiTrack de forma inalámbrica, lo cual permitió obtener información de su orientación y posición en tiempo real para planificar y ejecutar trayectorias dentro del espacio designado del entorno.

Las etapas se trabajaron siguiendo tres pasos. El primer paso consistió en investigación de los productos y tecnologías que podían ser aplicados para cubrir la necesidad planteada. El segundo paso fue manufacturar o implementar la solución seleccionada en el paso anterior. Finalmente, se realizó la validación del producto final para determinar si era necesario realizar modificaciones o cumplía con los estándares establecidos para cada prueba.

CAPÍTULO 2

Antecedentes

2.1. Megaproyecto SEEQ Robohelicóptero

El SEEQ Robohelicóptero surge por la motivación de abrir una línea de investigación en la Universidad del Valle de Guatemala en ramas de conocimiento científico que se han desarrollado en otros países, pero que en Guatemala no se han explorado a fondo. El proyecto que se trabaja en este informe será una continuación de esta misma línea de investigación, por lo que existen detalles importantes que se tomarán en cuenta del proyecto anterior. En este megaproyecto de investigación presentado por: Mario Búrbano, José Morales, Melisa Sandoval, Roberto Saravia y Daniel Suazo, se tuvo como objetivo diseñar y construir un robot aéreo con 4 rotores. Se utilizó un sensor láser para que él pudiese localizarse, estabilizarse y navegar dentro del ambiente en el que se encontraba. Utilizando la información de este sensor se generaba un mapa tridimensional y procesaba esta información en una etapa de visualización [2].

De este trabajo se observó la metodología utilizada para contemplar una estructura robusta, en la cual se consideraron esfuerzos, vibraciones y deflexiones.

Las mayores limitaciones que presentó este proyecto fueron:

- El ambiente a explorar debe ser un entorno cerrado.
- La rotación del robot respecto a los ejes X, Y & Z debe ser limitada.
- Los cambios de orientación deben ser a una velocidad baja.

Entre las conclusiones y recomendaciones de los autores, las que presentan una mayor importancia para este proyecto son:

- Se recomienda utilizar un controlador y motor que presenten una respuesta transitoria rápida, lo cual permite tener una mejor respuesta del sistema de control.

- Se recomienda utilizar módulos de comunicación inalámbrica para aumentar la versatilidad del dispositivo.

2.2. Diseño e implementación de una red de comunicación Wi-Fi e interfaz gráfica para una mesa de pruebas de robótica de enjambre

En el trabajo de graduación de Castañeda [3] se plantea el desarrollo del software para una mesa de pruebas de robótica de enjambre con un bajo costo y alta eficiencia. Esta mesa es utilizada para evaluar algoritmos de robótica de enjambre y ser puestos a prueba en entornos reales y sometidos a agentes físicos.

La sección cuyo aporte es de mayor importancia para este trabajo es la forma en que se implementa una red de comunicación dentro del sistema. Las características destacadas de la red son:

- Diseño e implementación de una red de comunicación bidireccional por medio de WiFi que permita la programación, el control y diagnóstico de micro robots.
- Implementación de un sistema de carga de programa que permite configurar todos los robots conectados a la red de forma inalámbrica y simultánea utilizando un bootloader interno del microcontrolador.
- La red puede ser diseñada por el usuario y creada por los agentes cuando una simulación se inicia.
- Se compara el nuevo protocolo de comunicación MQTT y su aumento en el rendimiento en las simulaciones comparado con un protocolo diferente utilizado en una versión previa del proyecto.

2.3. Diseño de estructura en configuración Y4 para dron

En el trabajo de graduación presentado por García [4] se diseñó la estructura de un drone en configuración Y4 la cual pudiese ser manufacturada rápidamente y que fuese resistente a fuerzas de flexión que se generan tanto por el peso de los motores, así como por la fuerza de sustentación que estos generan al rotar.

Los siguientes detalles se toman en consideración para este trabajo:

- Estructura fabricada mediante impresión 3D utilizando como material el PLA.
- Análisis de esfuerzos en la estructura utilizando Autodesk Inventor determinaron una estructura con deflexiones cercanas a cero. La fuerza de cada motor era de 0.55 N y una fuerza de empuje de 22.5 N por motor.

2.4. Generación de trayectorias y control de cuadricópteros

La tesis doctoral de Warren [5] realizada en conjunto con la Universidad de Pennsylvania presenta el control de un drone utilizando dos métodos diferentes. El primero permite controlar el vehículo aéreo en cercanía al punto donde el drone flota en el mismo lugar (punto de equilibrio) con los rotores apuntando verticalmente. El segundo método permite el control del vehículo en condiciones iniciales extremas. Es decir, se le puede exigir una alta aceleración e iniciar en casi cualquier posición y orientación.

Este trabajo se utiliza como guía sobre los parámetros que se deben tomar en cuenta para el sistema de control, así como una base importante de la matemática que permite la representación del drone como un sistema dinámico. Cabe destacar los siguientes aspectos para este trabajo:

- Para este trabajo se utiliza el sistema de control utilizando el método cercano al punto de equilibrio.
- Se presentan los supuestos realizados para la simplificación del sistema de control.
- Se presentan las restricciones de maniobrabilidad del drone para que los supuestos tomados en cuenta anteriormente se cumplan.
- Se utiliza el modelado dinámico del cuadricóptero como base para el modelado desarrollado para este trabajo.

CAPÍTULO 3

Justificación

Como se mencionó en los Antecedentes, los vehículos aéreos no tripulados tienen una gran capacidad de acceder y maniobrar en espacios reducidos, ya sea interiores o exteriores. Esta capacidad abre un amplio panorama de aplicaciones para los drones como recolección de información en lugares que presentan peligros para los humanos, vigilancia en lugares con adversarios armados, búsqueda y rescate durante catástrofes, control de calidad automatizada en campos de cultivo, entre muchas otras.

Debido a la gran cantidad de aplicaciones que tienen los drones, en la Universidad del Valle de Guatemala se creó una línea de investigación de los multirotores con el Megaproyecto SEEQ Robohelicóptero [2]. Los resultados obtenidos presentaron limitantes como velocidades de orientación restringidas, así como requerimiento de cambios de orientación a velocidad baja. El siguiente proyecto relacionado fue el diseño y fabricación de un drone en configuración Y4 [4]. Sin embargo, ese proyecto se enfocó más en la estructura física del drone, utilizando un controlador de vuelo comercial. El propósito de este trabajo es continuar con la línea de investigación de drones al manufacturar, modelar y controlar un cuadricóptero con productos que pueden conseguirse localmente y con un control de vuelo desarrollado para su utilización dentro del laboratorio de Robótica de la Universidad del Valle de Guatemala.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Modelar y controlar un cuadricóptero capaz de operar dentro del sistema de captura de movimiento inalámbrico del ecosistema Robotat.

4.2. Objetivos específicos

- Diseñar, analizar y manufacturar el marco estructural para un cuadricóptero adecuado a los componentes que se pueden adquirir localmente.
- Obtener el modelo dinámico del cuadricóptero y diseñar un sistema de control que emplee los motores BLDC para estabilizar la orientación de la plataforma empleando información de una unidad de medición inercial.
- Integrar al cuadricóptero dentro del sistema de captura de movimiento OptiTrack de forma inalámbrica para permita la planificación y ejecución de trayectorias de posición.

CAPÍTULO 5

Alcance

Este trabajo desarrolló únicamente tres áreas del cuadricóptero para el ecosistema Robotat. La primera, enfocada en el diseño y manufactura de un marco estructural capaz de sostener y asegurar los componentes del drone sin presentar una deformación significativa. La segunda, se enfocó en la programación e implementación de un controlador de vuelo en el ESP32 con lenguaje C, el cual permitirá al drone alcanzar el estado de *hovering* (flotar). En esta misma área se desarrolla también un entorno de simulación de vuelo utilizando MATLAB, el cual permite al usuario evaluar un controlador de vuelo antes de ser implementado en el dispositivo físico. Por último, la tercera área se enfocó en la integración del drone al ecosistema Robotat de forma inalámbrica a través de WiFi y el protocolo de comunicación MQTT.

El desarrollo de las áreas mencionadas previamente se complementan con el trabajo desarrollado por Hans Burmester, quien se encargó de la programación e implementación de librerías para la lectura de datos de una Unidad de Medición Inercial (IMU) y el manejo de los controladores electrónicos de velocidad para modificar las velocidades de motores sin escobillas. Así también, su trabajo se enfocó en seleccionar la batería y manufactura de PCB para la distribución de potencia del drone.

Debido a la pandemia del COVID-19 y la variación de colores del semáforo de nivel de alerta por región en Guatemala el acceso al laboratorio de robótica de la Universidad del Valle de Guatemala se vio atrasado al inicio del semestre e interrumpido durante el transcurso del mismo. Por lo tanto, los resultados de pruebas del drone en vuelo se vieron retrasadas. La modificación de constantes del controlador obtenidas por medio de experimentación más extensa permiten un desemepeño más eficiente.

CAPÍTULO 6

Marco teórico

6.1. *UAV*s en la actualidad

Durante los últimos 30 años, el mercado de UAVs (Unmanned Aerial Vehicle), también conocidos como drones, ha tenido un incremento exponencial. En 2010 se pronosticó que sería una industria de 10 mil millones de dólares. La Federal Aviation Administration (FAA) en ese año pronosticó que habrían 15,000 para propósito civil en 2020. En 2016, Goldman Sach Research pronosticó el aumento a un mercado de 100 mil millones de dólares para el 2020, donde la construcción, agricultura, entre otras ramas civiles formarían una parte importante de los consumidores de estos dispositivos [1].

Actualmente, la industria militar sigue liderando el mercado de los dispositivos aéreos no tripulados. Sin embargo, este trabajo pretende analizar una clase especial de UAV, los micro vehículos aéreos. Estos dispositivos son bastante más compactos que sus contrapartes MALE (Medium-Altitude Long-Endurance) UAVs y HALE (High-Altitude Long-Endurance) UAVs. Su diseño les permite navegar tanto en ambientes exteriores como en interiores, así como una maniobrabilidad adecuada en ambientes sin una estructura.

De estos micro vehículos aéreos existen diferentes tipos, dependiendo de la forma en que se propulsan. Los hay de alas fijas, alas móviles y rotores. En este proyecto, el objetivo principal son los de rotores, más específicamente, los multirotores. Los vehículos multirotores, entre los más comunes: cuadracóptero y hexacóptero, utilizan el empuje creado por cada uno de sus rotores apuntando hacia arriba para impulsarse. La combinación de dirección de giro de los rotores, así como la fuerza y momento que cada uno produce respecto al centro de masa del vehículo, le permite al dispositivo manipular su orientación y poder desplazarse en los ambientes descritos.

6.2. Marco estructural y análisis de esfuerzos

6.2.1. Diseño mecánico

El diseño mecánico puede llegar a ser una tarea complicada si no se trabaja con un orden adecuado. Para facilitar su análisis se divide el trabajo en tareas más sencillas, que al seguir una secuencia ordenada, permita alcanzar un producto que cumpla las especificiaciones económicas, de seguridad y funcionalidad establecidas por el diseñador. El proceso de diseño consiste en gran parte en el seguimiento de normas, códigos o estándares y convenciones para obtener un resultado provisional, el cual se mejora después de muchas iteraciones. Este proceso se ha facilitado en la actualidad gracias a la vastedad de información disponible y al aumento de herramientas de software para diseño cada vez más poderosas y precisas. La funcionalidad y capacidad de un producto de cumplir con su objetivo durante un tiempo de vida requerido está relacionado con el esfuerzo y la resistencia del mismo [6].

El diseño mecánico se ve afectado por incertidumbre debido a su naturaleza física, ya que hay factores muy complejos que se estudian como una simplificación o que simplemente son desconocidos. Para contrarrestar este efecto se utiliza el factor de seguridad. El factor de seguridad se establece matemáticamente como la multiplicación de una carga establecida preliminarmente como nominal por un número (factor de seguridad) mayor que uno [7].

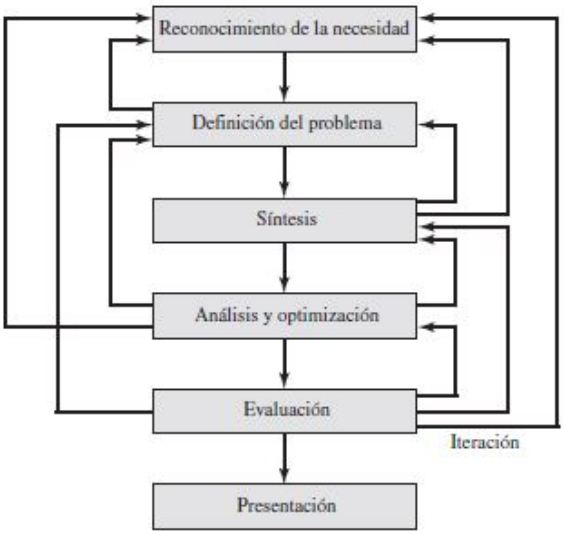


Figura 1: Fases del proceso de diseño que reconocen múltiples retroalimentaciones e iteraciones [6].

La Figura 1 muestra un diagrama con el proceso iterativo que involucra el proceso de diseño. El primer paso, y más importante para tener un proceso ordenado, es el reconocimiento de la necesidad. Al tener la necesidad clara el producto diseñado podrá ser diseñado para cubrirla. Los siguientes cuatro pasos: definición del problema, síntesis, análisis y optimización y evaluación son los que presentan la mayor cantidad de cambios, ya que permiten que con cada iteración se modifiquen los parámetros utilizados y se mejore el diseño hasta llegar a la última etapa, la presentación. En esta se presenta el mejor resultado obtenido en las etapas anteriores y se cumple con todos los requisitos establecidos previamente.

6.2.2. Esfuerzo y resistencia

La funcionalidad de un producto en una aplicación depende de la relación entre el esfuerzo al que se somete y la resistencia de este ante ese esfuerzo [6]. Como diseñador se busca que la resistencia del producto supere al esfuerzo con suficiente margen de seguridad debido a las incertidumbres mencionadas anteriormente, y de esta forma reducir la frecuencia de la falla.

La resistencia se define como una propiedad inherente de una pieza. Esta propiedad está constituida y depende del material empleado para la pieza y el proceso específico de formación que tuvo [8].

Otra propiedad perteneciente a una pieza es el esfuerzo. Esta es una propiedad de estado. Es una función que depende de la carga aplicada, la geometría de la pieza, la temperatura del objeto y el proceso de manufactura [6].

Para el diseño se realiza la comparación entre el esfuerzo y la resistencia de la pieza diseñada estableciendo como un punto crítico el punto donde sus dimensiones son iguales. Este punto es de alto interés para el análisis de las piezas porque en él ocurren eventos característicos del material como el límite de proporcionalidad (punto sobre el cual la relación entre carga-deformación ya no es lineal) o fractura. La importancia de este punto radica en que estos eventos, a su vez, representan el punto en el que la pieza pierde funcionalidad para la aplicación para la cual fue diseñada [7].

6.2.3. Herramienta computacional Autodesk Inventor

La ingeniería asistida por computadora (*CAE*, por sus siglas en inglés,*Computer Aided Engineering*) es un término utilizado para describir cualquier aplicación de ingeniería que se realice con ayuda de una computadora. Presenta beneficios como reducción de tiempo, mayor eficiencia, reducción de actividades monótonas y reducción de errores humanos en los cálculos matemáticos, entre otros [6].

Un subconjunto de esta área es el diseño asistido por computadora (*CAD*, por sus siglas en inglés, *Computer-Aided Design*), el cual se enfoca en la creación de diseños y modelos en tres dimensiones (3D). Este modelado presenta beneficios, como por ejemplo, cálculos de propiedades físicas y propiedades de materiales como masa, momentos de inercia, densidad, dimensiones físcias, ubicación de centro de masas, entre muchas otras. Existen muchas opciones de software *CAD* disponibles como:

- Aries.
- AutoCAD.
- SolidWorks.
- Autodesk Inventor.

Un subconjunto del software *CAD* está formado por las herramientas computacionales

que permiten simulaciones o análisis de ingeniería, conocido como *FEA* (por sus siglas en inglés, *Finite Element Analysis*).

El método de análisis del elemento finito (*FEA*) permite realizar una variedad amplia de simulaciones para el análisis de esfuerzos, deflexión, vibración y transferencia de calor [9]. Autodesk Inventor incluye una herramienta para realizar los análisis mencionados previamente, por lo que resulta una herramienta computacional adecuada para el diseño y análisis de esfuerzos necesarios para el diseño del marco estructural del drone trabajado en este proyecto.

6.2.4. Procesos y materiales de impresión 3D

Otra ventaja del diseño asistido por computadora, *CAD*, es que utilizando el modelo tridimensional de las piezas se pueden utilizar métodos de manufactura y prototipado rápido, tal como es el caso de la impresión 3D. La impresión 3D puede ser analizada bajo 5 grandes cualidades o propiedades: costo del equipo, dimensiones de la máquina, material con el que trabaja, precisión, y otras [10]. Debido a que para este trabajo se utilizarán las impresoras 3D que posee el departamento de Ingeniería Mecánica y de Ingeniería Electrónica, Mecatrónica y Biomédica, la propiedad a analizar es el material con el que se fabricará la pieza.

Existen diferentes materiales para la fabricación de modelos físicos tridimensionales a partir de impresión 3D. El material seleccionado depende del tipo de impresión que se utilice. Entre ellos están:

- Aglutinadoras de material sin cambio de estado.
- Extrusión o inyectoras de material.
- Solidificación de material con cambio de estado.

La Universidad del Valle cuenta con los dos últimos tipos de impresora mencionados. Las impresoras de extrusión o inyectoras funcionan con dos cabezales. El primero se encarga de depositar un material de soporte, el cual es un material con bajo punto de fusión y que es soluble en un disolvente especial detallado por el fabricante. El segundo cabezal deposita el material de fabricación utilizado, el cual puede ser cera con alto punto de fusión, un termoplástico como ABS o una resina como PLA. Los beneficios de este tipo de impresión son el bajo costo, un acabado superficial de alta calidad, y que el material puede reciclarse. Al tener un solo cabezal, presentan el inconveniente de tener que eliminar los soportes manualmente, sin embargo, es un proceso sencillo.

El segundo tipo de impresora 3D con que se cuenta en el departamento es la de solidificación de material con cambio de estado o polimerización [10]. La resina utilizada se polimeriza a través de luz ultravioleta. El material de fabricación comienza en estado líquido el cual se solidifica con la luz. Este método presenta ventajas como una alta calidad de acabado superficial y se puede obtener un producto traslúcido, sin embargo, presenta dificultades para remover los soportes que son del mismo material que la pieza. Así también, permite trabajar únicamente con un color y no es posible reciclar el material.

6.3. Modelado dinámico

6.3.1. Sistemas de coordenas y marcos de referencia

Un vector consta de un valor numérico más una direccción, utilizado para especificar la posición de un punto respecto a otro punto en el espacio. Un cuerpo rígido es la composición infinita de puntos, por lo que también posee una orientación. Para describir completamente la ubicación de un cuerpo rígido se combina la posición y orientación respecto a un marco de referencia, y se conoce como pose. Un marco de referencia o sistema de coordenadas Cartesianas, es un *set* de ejes ortogonales entre sí los cuales se intersecan en un punto denominado origen. Si se agrega un marco de referencia a un cuerpo rígido (objeto), se puede referenciar cada punto que conforma al objeto como un vector constante desde ese marco de referencia. Por último, se describe la pose, posición y orientación, de este marco respecto al sistema de coordenadas Cartesianas o marco de refencia global. Para no crear confusión entre los marcos de referencia se etiquetan por medio de letras mayúsculas entre llaves, por ejemplo, {B}, para el caso del cuerpo rígido. El resultado se observa en la Figura 2 [11].

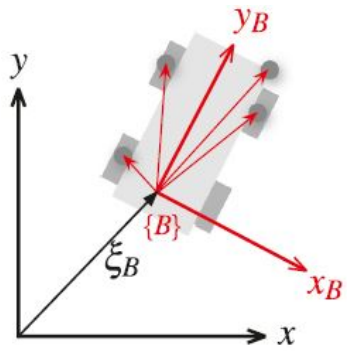


Figura 2: Puntos del cuerpo rígido referenciados al marco de referencia del objeto {B} y este referenciado al marco de referencia global [11].

6.3.2. Diagrama de cuerpo libre y coordenadas

Utilizando los conceptos anteriores, se representa la pose del marco de referencia colocado en el centro de masa del dron {B}, Figura 3, con ejes x_B hacia adelante y z_B apuntando perpendicularmente hacia arriba del plano de los rotores. El marco de referencia global se denomina {W}, con ejes x_W, y_W y z_W.

En la Figura 3 se dibujan todas las fuerzas (F_1, F_2, F_3, F_4) y momentos (M_1, M_2, M_3, M_4) producidas por los 4 rotores. El largo de cada brazo es L y la fuerza de la gravedad que actúa sobre el centro de masa del drone se encuentra en el eje $-z_W$. De esta forma, se representan completamente al dron, las fuerzas y los momentos que actúan sobre él y su pose respecto a un sistema de coordenadas global. Esta representación será utilizada para reconocer los términos de las ecuaciones presentadas más adelante [5].

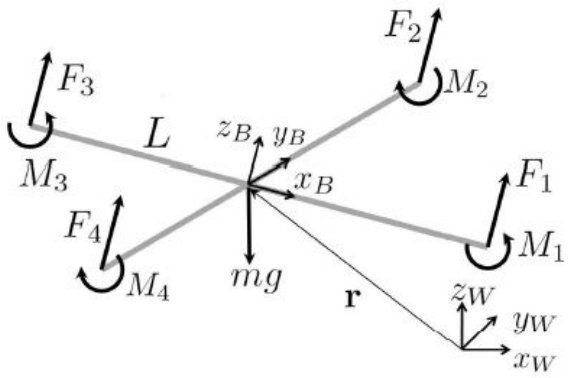


Figura 3: Diagrama de cuerpo libre y sistemas de coordenadas para modelo del drone [11].

6.3.3. Modelo dinámico del drone

Debido a que se desea indicar la pose del drone respecto a un marco de referencia global, Figura 3, es necesario hacer una transformación que permita pasar de las coordenadas respecto de un marco, {W}, a otro, {B}. Esta transformación se realiza utilizando una matriz de rotación la cual combina las rotaciones necesarias para llegar de una orientación a otra. El teorema de rotación de Euler indica que cualquier orientación en el espacio 3D se puede representar por no más de tres rotaciones consecutivas sobre ejes coordenados [12]. Existen 12 posibles combinaciones de secuencias para los ángulos de Euler, sin embargo, para este caso se utilizará la combinación ZXY. Las rotaciones respecto a los ejes coordenados tienen nombres establecidos en la industria aeroespacial:

- Rotación respecto eje x: roll - ϕ.
- Rotación respecto eje y: pitch - θ.
- Rotación respecto eje z: yaw - ψ.

La rotación necesaria para llegar de {W} a {B} se realiza de la siguiente forma: Rotación de ψ respecto z_W, seguida de rotación de ϕ respecto x y finalmente, una rotación de θ respecto y_B [5]. Estas rotaciones agrupadas se pueden representar como la siguiente matriz de rotación:

$$R = \begin{bmatrix} c(\psi)s(\theta) - s(\phi)s(\psi)s(\theta) & -c(\phi)s(\psi) & c(\psi)s(\theta) + c(\theta)s(\phi)s(\psi) \\ c(\theta)s(\psi) + c(\psi)s(\phi)s(\theta) & c(\phi)c(\psi) & s(\psi)s(\theta) - c(\psi)c(\theta)s(\phi) \\ -c(\phi)s(\theta) & s(\phi) & c(\phi)c(\theta) \end{bmatrix} \tag{1}$$

Se utiliza c como abreviación de coseno, y s como abreviación de seno, para tener una matriz más sencilla de leer.

La aceleración del centro de masa, $\ddot{r}$, se representa como:

$$m\ddot{r} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + R \begin{bmatrix} 0 \\ 0 \\ \sum F_i \end{bmatrix}, \tag{2}$$

donde F_i corresponde a la fuerza generada por cada uno de los rotores, m corresponde a la masa del drone y g, a la gravedad.

La velocidad angular se describe por sus componentes p, q y r en el marco de referencia sobre el drone {B}, como valores relacionados con los ángulos de giro *roll*, *pitch* y *yaw*.

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} c(\theta) & 0 & -c(\phi)s(\theta) \\ 0 & 1 & s(\phi) \\ s(\theta) & 0 & c(\phi)c(\theta) \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}. \tag{3}$$

Por último, se establece la relación matemática para la aceleración angular utilizando la ecuación de Euler:

$$I \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} L(F_2 - F_4) \\ L(F_3 - F_1) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times I \begin{bmatrix} p \\ q \\ r \end{bmatrix}. \tag{4}$$

6.3.4. Modelado de los motores

Al rotar la hélice colocada en el eje de cada motor, estos producen una fuerza de empuje vertical determinada por F_i, donde i es el indicador del número de rotor al que pertence (F_1, F_2, F_3, F_4). Esta fuerza está definida por la relación:

$$F_i = k_F w_i^2, \tag{5}$$

de igual forma, cada motor produce un momento definido por:

$$M_i = k_M w_i^2, \tag{6}$$

donde, k_F y k_M son constantes que dependen de las características físicas y de construcción del motor, y w_i es la velocidad angular de cada rotor o hélice.

El modelado real del motor depende de una complicada función entre la dinámica del motor, su controlador y la hélice. Además, depende de una función de velocidad del motor y de si la velocidad está en ascenso o descenso. Sin embargo, puede hacerse una aproximación del modelo del motor utilizando una ecuación diferencial de primer orden:

$$\dot{w}_i = k_m(w_i^{deseada} - w_i), \tag{7}$$

donde k_m es una constante determinada experimentalmente que permite sincronizar el desempeño real del motor con el desempeño simulado del controlador [5].

6.4. Sistema de control

6.4.1. Espacio de estados

Debido al incremento en la complejidad y precisión de las tareas requeridas por sistemas de ingeniería se desarrolló la teoría del control moderno. Esta teoría se basa en el concepto de estado.

Una gran ventaja de trabajar con este enfoque es que permite formular problemas de múltiples salidas con múltiples entradas, tal como lo es el caso del drone. Además, estos sistemas pueden ser lineales, no lineales, variables en el tiempo o invariantes en el tiempo. Para una mejor comprensión de este concepto se presentan las siguientes definiciones:

- **Estado:** El estado de un sistema dinámico se conforma por el conjunto más pequeño de variables que, al conocer su valor en un tiempo inicial t_0 y conocer el valor de la entrada al sistema para $t \geq t_0$, permite describir el comportamiento del sistema en cualquier instante $t \geq t_0$.

- **Vector de estado:** Si la cantidad de variables de estado necesarias para describir el comportamiento de un sistema es n, entonces el vector de estado $\mathbf{x}$ está formado por n componentes, donde cada componente es una de las variables de estado (x_1, x_2,...,x_n).

De esta forma, el espacio de estados es un espacio n-dimensional donde cada uno de sus ejes coordenados está formado por las variables de estado. Esta representación permite que cualquier estado del sistema se presente como un punto en el espacio de estados [13].

6.4.2. Diagrama de control

La Figura 4 muestra el lazo cerrado que se debe realizar para poder controlar la posición y orientación del drone, teniendo como entrada la posición deseada $r_T(t)$. En los sistemas de control en lazo cerrado, también conocidos como sistemas de control realimentados, se alimenta al controlador con una señal de entrada deseada y una señal de realimentación (puede ser la salida o una función relacionada con la salida y/o sus derivadas o integrales) [13]. Se calcula la diferencia entre estas dos entradas y se determina un error el cual entra al controlador. El propósito de este lazo cerrado es reducir el error, lo que a su vez conduce la salida al valor deseado. Una de las mayores ventajas de utilizar el control con realimentación es que la respuesta del sistema controlado, en este caso el control del drone, se vuelve robusta ante perturbaciones externas.

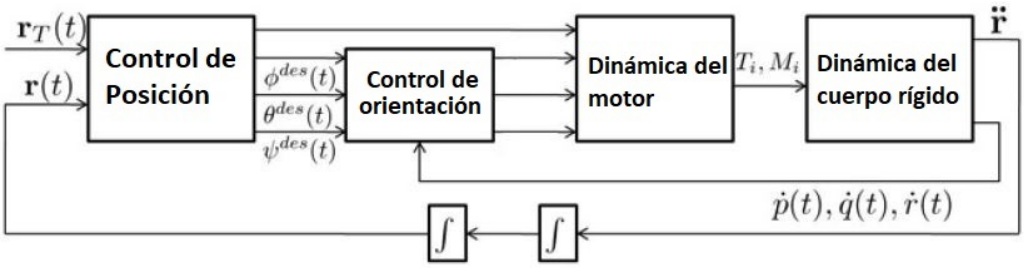


Figura 4: Lazos de control anidados para control de posición y orientación [5].

6.4.3. Sistema de control de ángulo pequeño o cerca del punto de equilibrio (*hover*)

Tal como se muestra en la Figura 4, existen dos lazos cerrados anidados (uno dentro de otro) en el controlador. El lazo interno se encarga de de la oritentación utilizando una Unidad de Medición Inercial (sensor con giroscopio y acelérometro) a bordo del drone para controlar los ángulos de *roll*, *pitch* y *yaw*. El lazo externo se encarga de controlar la posición y velocidad del centro de masa y permite seguir trayectorias en el espacio 3D.

El sistema de control utilizado se denomina de ángulo pequeño porque las ecuaciones de movimiento y del modelo de los motores se linealizan respecto a un punto de operación donde el drone flota en el mismo lugar, es decir, $\mathbf{r} = r_0$. En este punto de operación se cumplen las siguientes condiciones:

- $\theta = \phi = 0$.
- $\psi = \psi_0$.
- $\dot{r} = 0$.
- $\dot{\phi} = \dot{\theta} = \dot{\psi} = 0$.
- $cos(\phi) \approx 1$.
- $cos(\theta) \approx 1$.
- $sen(\phi) \approx \phi$.
- $sen(\theta) \approx \theta$.

En el punto donde el drone flota sobre el mismo punto, la propulsión de los motores debe ser [5]:

$$F_{i,0} = \frac{mg}{4}, \tag{8}$$

y la velocidad angular de los motores está dada por:

$$w_{i,0} = w_h = \sqrt{\frac{mg}{4k_F}}. \tag{9}$$

Utilizando esta aproximación, se analizan los dos lazos de control por separado:

6.4.4. Control de orientación:

Las condiciones asumidas previamente llevan a nuevas asumciones: Los momentos de inercia son pequeños por ser ángulos pequeños de diferencia entre los ejes coordenados y los acutales, y $I_{xx} \approx I_{yy}$ debido a la simetría del drone.

Por lo tanto:

$$I_{xx}\dot{p} = u_2 - qr(I_{zz} - I_{yy}), \tag{10}$$

$$I_{yy}\dot{q} = u_3 - pr(I_{xx} - I_{zz}), \tag{11}$$

$$I_{zz}\dot{r} = u_4. \tag{12}$$

El siguiente supuesto es que la componente de la velocidad angular en el eje z_B, r, es pequeña. Esto como consecuencia hace que los términos de resta en el lado derecho de las ecuaciones (10) y (11) sean muy pequeños comparados con los otros términos. Se observa que en este punto de operación, $\dot{\phi} \approx p$, $\dot{\theta} \approx q$ y $\dot{\psi} \approx r$ [5]. Gracias a estas razones, se utiliza una ley de control proporcional-derivativo de la forma:

$$u_{2,des} = k_{p,\phi}(\phi^{des} - \phi) + k_{d,\phi}(p^{des} - p), \tag{13}$$

$$u_{3,des} = k_{p,\theta}(\theta^{des} - \theta) + k_{d,\theta}(q^{des} - q), \tag{14}$$

$$u_{4,des} = k_{p,\psi}(\psi^{des} - \psi) + k_{d,\psi}(r^{des} - r), \tag{15}$$

Finalmente, se obtiene el vector de velocidades de los motores utilizando la fuerza neta deseada ($u_{1,des}$) y los momentos ($u_{2,des}$, $u_{3,des}$ y $u_{4,des}$) invertidos:

$$\mathbf{u}_{des} = \begin{bmatrix} k_F & k_F & k_F & k_F \\ 0 & k_F L & 0 & -k_F L \\ -k_F L & 0 & k_F L & 0 \\ k_M & -k_M & k_M & -k_M \end{bmatrix} \begin{bmatrix} w_{1,des}^2 \\ w_{2,des}^2 \\ w_{3,des}^2 \\ w_{4,des}^2 \end{bmatrix}. \tag{16}$$

6.4.5. Control de posición:

El método de control para la posición implementado se utiliza para el rastreo de trayectorias en un espacio tridimensional. Es importante destacar que las aceleraciones deben ser moderadas para que las aproximaciones del punto cercano al equilibrio sigan cumpliéndose [5].

Se define el error de posición como:

$$e_p = [(r_T - r) \cdot \hat{n}]\hat{n} + [(r_T - r) \cdot \hat{b}]\hat{b}, \tag{17}$$

y, el error de velocidad como:

$$e_v = \dot{r_T} - \dot{r}, \tag{18}$$

donde:

- r_T: punto en la trayectoria deseada.
- r: posición actual.
- $\dot{r_T}$: vector de velocidad deseada.
- $\hat{n}$: componente normal del vector unidad tangente asociado al punto.
- $\hat{b}$: componente binormal del vector unidad tangente asociado al punto.

Al tener estos errores se calcula la aceleración deseada para el drone con un controlador PD:

$$\ddot{r_i}^{des} = k_{p,i}e_{i,p} + k_{d,i}e_{i,v} + \ddot{r}_{i,T}. \tag{19}$$

De igual forma, se computan los valores deseados de *roll* y *pitch* a partir de la aceleración deseada $\ddot{r}$:

$$\phi_{des} = \frac{1}{g}(\ddot{r_1}^{des} sen(\psi_T) - \ddot{r_2}^{des} cos(\psi_T)), \tag{20}$$

$$\theta_{des} = \frac{1}{g}(\ddot{r_1}^{des} cos(\psi_T) - \ddot{r_2}^{des} sin(\psi_T)), \tag{21}$$

$$u_{1,des} = m\ddot{r_3}^{des}. \tag{22}$$

6.4.6. Microcontrolador ESP32:

El ESP32, Figura 5, es una tarjeta de desarrollo de la compañía y plataforma ESPRESSIF. Esta tarjeta cuenta con un microcontrolador que tiene de forma integrada teconología wifi, *Bluetooth* y *Bluetooth LE*. Está diseñada para una gran variedad de aplicaciones. Se destaca por su capacidad de operar con bajo consumo de potencia pero también por su capacidad de realizar tareas demandantes como codificación de voz, entre otras [14].

Una de sus mayores ventajas es la aplicación en proyectos de internet de las cosas (IoT), por su facilidad de integración wifi y su facilidad de programación. Además, cuenta una gran cantidad de módulos que permiten el manejo de sensores y generación de señales, otorgando una gran versatilidad para esta tarjeta.

Entre las características más importantes del microcontrolador se encuentran las siguientes [15]:

- **Microprocesador:** LX6 Tensilica (32 bits y núcleo dual)
- **Voltaje de operación:** 2.2 a 3.6 V
- **Corriente de *Deep Sleep*:** 2.5 µA
- **Memoria Flash:** 4 MB
- **Frecuencia de Reloj:** hasta 240 MHz (en IDE solo hasta 80mhz)
- **Cifrado con Hardware Acelerado:** (AES, SHA2, ECC, RSA-4096)

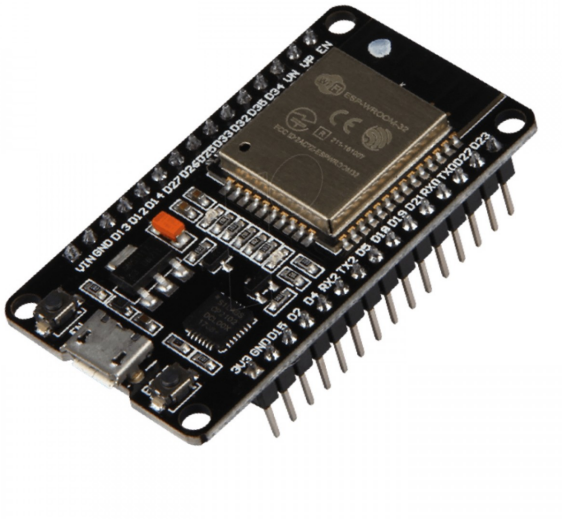


Figura 5: Tarjeta de desarrollo ESP32 [15].

A su vez, la tarjeta posee las siguientes características:

- 28 GPIO

- 3 UART
- 18 Canales ADC
- 3 Interfaces SPI
- 2 Interfaces I2C
- Salidas 16 PWM
- 2 DAC de 8 bits
- SRAM Interno : 520 KB
- Transceptor Integrado de WiFi de 802.11 b/g/n
- Bluetooth Integrado de Modo Dual (Clásico y BLE)

6.5. Sistema de captura de movimiento

6.5.1. OptiTrack para robótica

OptiTrack es un sistema de rastreo en tiempo real de baja latencia (tiempo de retraso entre la solicitud de información y el envío de esta), alta precisión y 6 grados de libertad. En robótica, es utilizado para el rastreo de robots terrestres y vehículos aéros no tripulados *UAV*s dentro de un volumen limitado. Provee la información necesaria para que el sistema de control pueda saber la posición del objeto en interiores [16].

Figura 6: Cámara $Prime^x$ 22 utilizada por el sistema OptiTrack [16].

El sistema de captura de movimiento OptiTrack utiliza un conjunto de cámaras $Prime^x$ 22, Figura 6, de alta velocidad las cuales utilizan disparadores globales que detienen la imagen y *LED*s de descarga rápida para tomar la imagen que posteriormente es procesada por un *software* iterno y que permite pasar de fotones a representaciones de cuerpos rígidos.

En la Figura 7 se muestra una sección de la interfaz gráfica del sistema de captura de movimiento OptiTrack, la cual permite el rastreo de marcadores incluidos con el equipo y colocados sobre el objeto a seguir para poder obtener su posición en tiempo real respecto a un eje de coordenadas establecido.

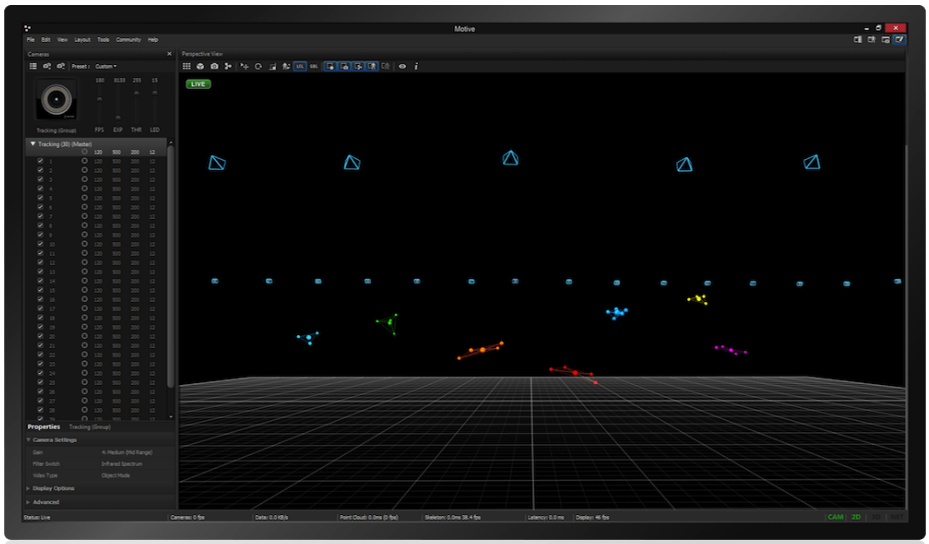


Figura 7: *Software* de OptiTrack para rastreo de marcadores dentro del espacio formado por las cámaras [16].

En este trabajo se utilizará el *software* OptiTrack para obtener la información de la posición del drone repescto a un eje de coordenadas establecido en el programa.

CAPÍTULO 7

Diseño mecánico del marco estructural

7.1. Metodología

El diseño del marco estructural se realizó lo más sencillo posible para que el resultado de la manufactura utilizando material de impresión 3D (PLA o ABS) sea ligero. Este diseño fue sometido a un software de análisis de esfuerzos para validar su efectividad para mantener las piezas en su posición y ser capaz de resistir los esfuerzos provocados por el peso de las piezas y la fuerza de empuje generada por cada motor. Una vez finalizado el modelado 3D, se fabricó con el material seleccionado y se sometió a una prueba de deformación utilizando los pesos reales de los dispositivos que debe soportar. Utilizando el criterio de diseñador de Shigley [6], la deformación no debe ser mayor al 1 % de la longitud total de la pieza. De esta forma, se confirmó que el diseño de la estructura cumple exitosamente la funcionalidad para la que fue diseñado. En esta área de trabajo es importante realizar un diseño que sea compacto para poder interactuar dentro del ecosistema Robotat, el cual se encuentra dentro de un laboratorio cerrado. La Figura 8 muestra el proceso que se siguió para obtener el marco estructural del drone y su manufactura.

7.2. Selección de componentes y determinación de medidas

La restricción más importante para la selección de los componentes a utilizar en el drone es su disponibilidad de adquisición dentro del país (Guatemala). Esto se debe a que se desea que en el futuro, se fabriquen múltiples cuadricópteros y poder adquirir los materiales fácilmente, evitando así el tiempo y cobros de envío e importación.

Otro aspecto importante a considerar durante el dimensionamiento de la estructura fueron las restricciones físicas como dimensiones y peso de los objetos y dispositivos que

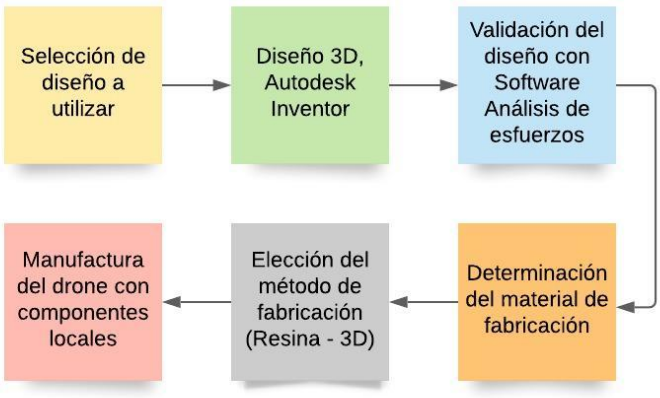


Figura 8: Diagrama de flujo del diseño estructural.

deben ir sobre el marco.

Componente	Dimensiones (cm)	Peso (g)
ESP32	2.8 x 5.4	6.80
IMU MPU-9250	1.5 x 2.5	2.72

Cuadro 1: Dimensiones de dispositivos sobre el drone. En este cuadro se presentan las restricciones de peso y dimensiones que requiere cada uno de los elementos que son indispensables para el funcionamiento del cuadricóptero.

El ancho y largo de los elementos determina el área mínima que se requiere en el marco estructural y el peso determina la fuerza mínima de empuje que deben entregar los motores para levantar el drone. En el Cuadro 1 aún no se toma en cuenta dimensiones de brazos del drone ni elementos que no van sobre el centro del marco estructural (controladores de velocidad *ESC*s y batería). De este mismo cuadro se obtuvo que el área mínima para el centro del drone debía ser de $18.87cm^2$, y motores con capacidad de levantar al menos $9.52g$ entre los 4.

Tomando en cuenta los aspectos mencionados, se procedió a seleccionar uno de los motores para drones más accesibles y robustos del mercado, el BLDC 2212/13T 1000KV, Figura 9, cuyas características se presentan en el Cuadro 2.

Indicador	Significado
22	diámetro de motor (22 mm)
12	Altura de motor (12 mm)
13T	Número de vueltas por polo (13)
1000KV	1000 revoluciones por voltio aplicado

Cuadro 2: Motor seleccionado 2212/13T 1000KV. En este cuadro se presentan las propiedades que presenta el *brushless motor* según el fabricante Digey-key [17].

Ya que se cuenta con el motor específico para la aplicación, se seleccionó la hélice utilizando como base los resultados obtenidos en [18]. En el cuadro: *"Quad copter frame and*

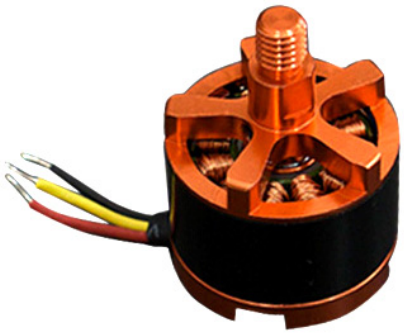


Figura 9: Motor BLDC A2212/13T 1000KV.

Lipo battery and motor and propeller size matching table" se presenta la recomendación para el motor que se seleccionó, el rango de tamaño del marco estructural y la batería. Por lo tanto, se selecciona la hélice 1045, Figura 10. El 10 respresenta el diámetro de hélice ($10in$) y el 45 el paso de la hélice ($4.5in$).

Figura 10: Hélices 1045.

La hélice 1045 se encuentra disponible con múltiples proveedores en Guatemala. A su vez, según el cuadro *"Motor Thrust Data table"* [18], esta hélice tiene capacidad de levantar hasta $734g$ por motor. Por lo tanto, cumple con los requisitos mínimos establecidos.

El último componente a seleccionar fue el controlador electrónico de velocidad, *ESC*s, el cual se encarga de la velocidad y dirección de giro de los motores *Brushless:* . Se utilizó la misma restricción de estar disponible en el país local, por lo que el Módulo ESC 30A, Figura 11, cumple con la corriente que necesita el motor *brushless*. Además, es de costo reducido ($\approx Q110$) y maneja voltajes entrada y salida comúnes con microcontroladores, baterías y motores sin escobillas. En el Cuadro 3 se presentan más características.

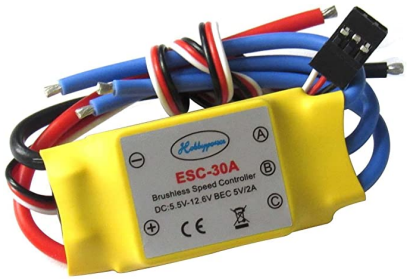


Figura 11: Controlador Electrónico de Velocidad - ESC 30A.

Voltaje lógico	$5V$
Voltaje de alimentación	$7.4V$ - $12.6V$
Corriente continua máxima	$30A$
Peso	$25g$
Dimensiones	4.5 x 2.4 cm

Cuadro 3: Características controlador electrónico de velocidad - ESC 30A.

7.3. Modelado 3D CAD/CAM

Al tener los componentes del drone seleccionados en la sección anterior, el proceso de diseño se basó en ajustarse a las dimensiones de estos y al área mínima de trabajo para el centro del drone. Por ejemplo, la parte de los brazos del drone en donde se colocan los motores sin escobillas se dimensionó con agujeros para encajar con las entradas de los motores. Así mismo, se dimensionó un espacio a la mitad del brazo para que se coloque el controlador de velocidad y se asegure por medio de cinchos.

Las dimensiones exactas y planos se presentan en el repositorio de GitHub para los planos.

A continuación se detallan algunas consideraciones tomadas para el diseño:

- Distancia suficiente entre eje del motor y centro del drone para que las hélices puedan girar libremente, sin colisionar con ningún objeto o entre ellas.
- Aberturas a los lados del espacio de los *ESC*s para ajustarse con cinchos de seguridad o velcro.
- Dejar un espacio de al menos $2cm$ entre dispositivos para permitir conexiones seguras y más sencillas al momento de armarlo físicamente.
- Disminuir el peso del brazo al retirar material. Esto se logró por medio de 2 métodos, los cuales se explican más adelante.
- Reducir la deformación de los brazos provocada por el peso del motor, hélice y controlador de velocidad, así como la fuerza de empuje del motor hacia arriba. Esto se logró haciendo una sección transversal que aumenta su tamaño mientras se acerca hacia el centro del drone. Esto asegura que el área transversal más alejada de la fuerza aplicada tenga un área más grande, disminuyendo así el esfuerzo de tensión por flexión.

- El área del centro del drone debe ser tener al menos $18.87cm^2$ para colocar los dispositivos mencionados anteriormente. También se consideró que se desea un espacio para colocar las conexiones de potencia y control entre la batería, ESP32 y controladores electrónicos de velocidad.

Remoción de material, método 1: El primer método para remover material y reducir el peso de los brazos del drone consistió en crear un patrón de triángulos el cual al momento de manufacturar queda en forma de espacios vacíos. La forma de los triángulos permitió que la distancia entre espacios vacíos fuese más pequeña sin comprometer la resistencia de la pieza. Las esquinas no provocan concentración de esfuerzos porque no están alineadas con la dirección de la fuerza, y por lo tanto la de los esfuerzos.

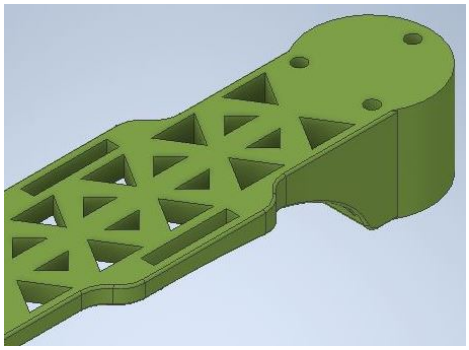


Figura 12: Método 1, remoción de material.

Remoción de material, método 2: El método 1 se aplica comúnmente en piezas que son formadas cómo un sólido. En el caso de piezas plásticas, se utiliza con el método de plástico inyectado. Sin embargo, el método empleado para la manufactura de este drone es el de impresión 3D, el cual se realiza capa por capa. Esta forma de fabricación permite controlar el porcentaje de relleno (*infill*) que tendrá la pieza. Es decir, la pieza se mira como un sólido, pero por dentro, algunas de las capas tienen vacío en forma de triángulo para que el peso se reduzca considerablemente.

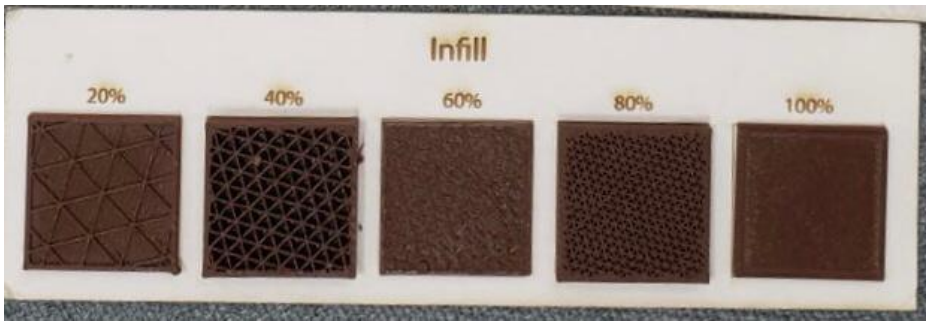


Figura 13: Método 2, remoción de material.

Sin embargo, es importante destacar que al retirar material de esta forma se disminuye la resistencia de la pieza, por lo que se debe utilizar cuidadosamente.

Para esta aplicación se decidió utilizar una combinación de ambos métodos. El método 1 permite retirar material, pero más importante cada contorno o pared se creará como un

sólido sin importar el relleno que se seleccione al momento de la impresión 3D. Por lo tanto, el brazo tiene una forma parecida a un esqueleto que es completamente sólido. Posteriormente, se complementa el sólido con un relleno de 20 %. La combinación de estos métodos permite tener una pieza resistente a los esfuerzos de tensión por flexión y a su vez sea liviana.

7.3.1. Resultados del diseño 3D

Base superior e inferior

El drone se diseñó para tener 2 niveles, una base superior y una inferior, las cuales permiten sujetar los brazos del drone por medio de tornillo. Esto elimina la necesidad de utilizar una única pieza sólida.

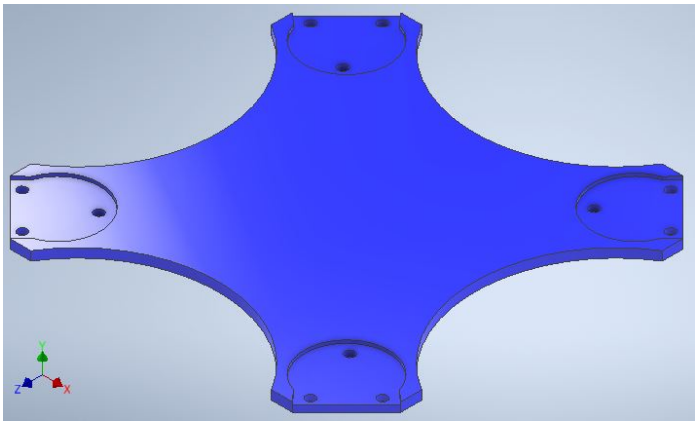


Figura 14: Base del cuadricóptero. La base superior e inferior tienen el mismo diseño.

Brazos

El diseño del brazo tiene la sección más gruesa en el punto donde se une al centro de la estructura del drone. El grosor extra se utiliza para crear un espacio utilizable en el medio del drone y a su vez para disminuir los esfuerzos normales por flexión en la zona crítica del brazo. Se utilizan 3 tornillos $M2.5$ y sus tuercas para mantenerlos asegurados a la base superior e inferior. Este tipo de unión permite que la estructura se arme y desarme las veces que sea necesario sin comprometer la integridad de las piezas.

Elementos de sujeción

Debido a que el drone se diseñó para tener partes separadas, fue necesario utilizar elementos de sujeción que permitiesen formar una estructura sólida y segura. El listado de partes de sujeción se muestra en el Cuadro 4.

Cantidad	Componente	Detalles
24	Tornillo M3x10	Sujeción de BLDC al brazo.

Continúa en la siguiente página

Cuadro 4 – *Continuación de la página previa*

Función	Parámetros	Descripción
12	Tornillo M2.5x30	Sujeta brazos a base.
12	Tuercas M3	Asegurar brazos y patas a base.
12	Tuercas M2.5	Asegurar brazos a base.
12	Conectores bala*	Conexión de cableado eléctrico de fácil acceso.
4	Cinchos de seguridad	Puede ser sustituido por tiras de velcro.
1	Tira de velcro.	1/2 yd. Sujeción de batería a base.

Cuadro 4: Componentes de sujeción para elementos del cuadricóptero. Los elementos utilizados están en Sistema Internacional o métrico, manteniendo las mismas unidades que el diseño 3D. *Conectores bala se trabajan como pares macho y hembra. Se recomienda un tamaño mediano para cubrir un rango amplio de cable.

Ensamblaje completo

El resultado de unir las piezas diseñadas se encuentra en la Figura 16. Se puede apreciar que las piezas del centro tienen la misma forma y solo se invierte su dirección para que la ranura que poseen en las esquinas puedan casar con la forma de los brazos.

7.4. Análisis de esfuerzos: Método elementos finitos

La herramienta de *software*, *Autodesk Inventor 2021*, se utilizó para el análisis de esfuerzos y deformación en la sección crítica del drone. El análisis se realizó sobre el brazo del drone el cual es la pieza sometida a esfuerzos normales por flexión.

En la Figura 18 se presentan las fuerzas aplicadas y las restricciones utilizadas para simular la situación a la que se someterá la pieza durante la aplicación. La parte más ancha se encuentra restringida completamente debido a que es la sección que estará acoplada al centro del drone, por lo que no puede moverse en ninguna dirección.

La fuerza aplicada en el centro del círculo es la fuerza provocada por el peso del BLDC, $50g \approx 0.510N$ [17]. La segunda fuerza corresponde al peso del controlador electrónico de velocidad, el cual tiene un peso de $25g \approx 0.250N$, Cuadro 3.

El eje Y es ortogonal al plano donde se aplican los esfuerzos (XZ), por lo que para el estudio de la deformación, el desplazamiento en el eje Y es el que determina si el diseño es adecuado para manufactura.

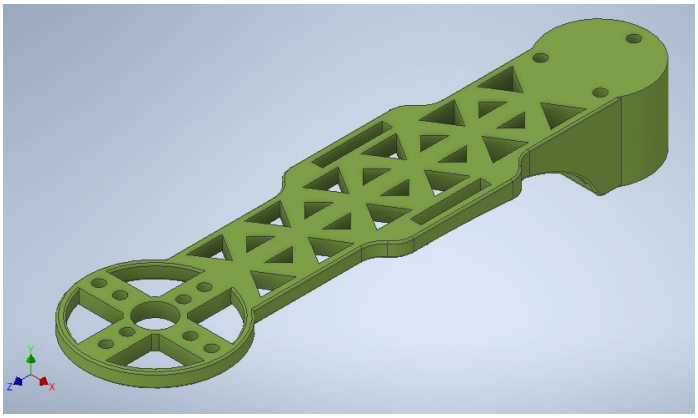


Figura 15: Diseño 3D de brazos para cuadricóptero.

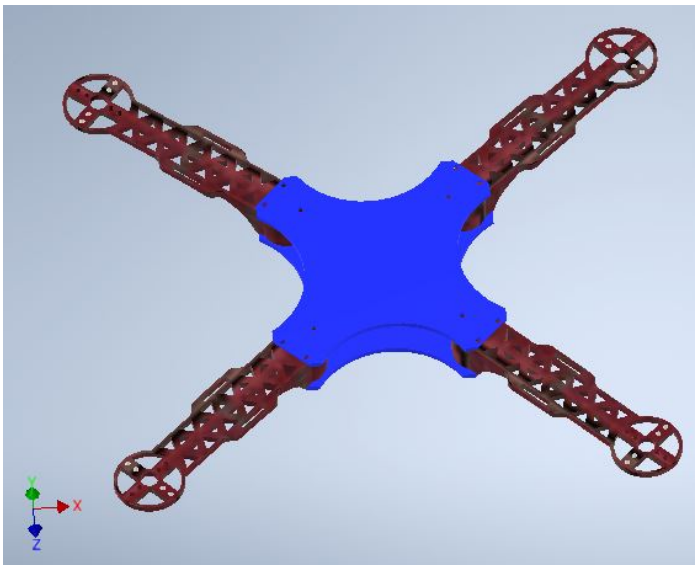


Figura 16: Ensamble completo.

Para realizar el análisis se utilizó el ABS como material de manufactura. Este presentó la resistencia menor entre los materiales de impresión 3D planteados para este trabajo, ABS y PLA. Por lo tanto, la conclusión sobre los resultados obtenidos para el material más débil serán concluyentes para el material más resistente.

7.4.1. Resultados del análisis de elementos finitos

En la Figura 19 se encuentran los esfuerzos Von Mises a los que está sometido el brazo cuando se colocan los componentes sobre él. El valor máximo que alcanza es de $1.138MPa$, el cual se encuentra en la localidad del controlador de velocidad. Se observa que la sección crítica, la cual es la base del brazo restringida de movimiento, se encuentra en color azul, indicando que el esfuerzo en esa sección es menor a $0.228MPa$.

En la barra de colores de la Figura 20 se observa que el valor máximo de deformación es

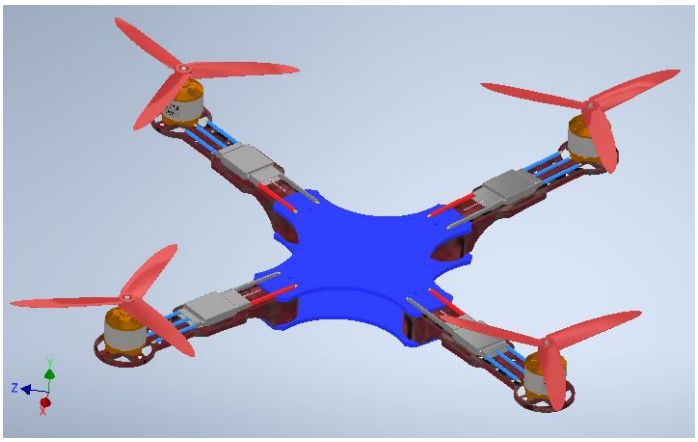


Figura 17: Ensamble completo con componentes seleccionados.

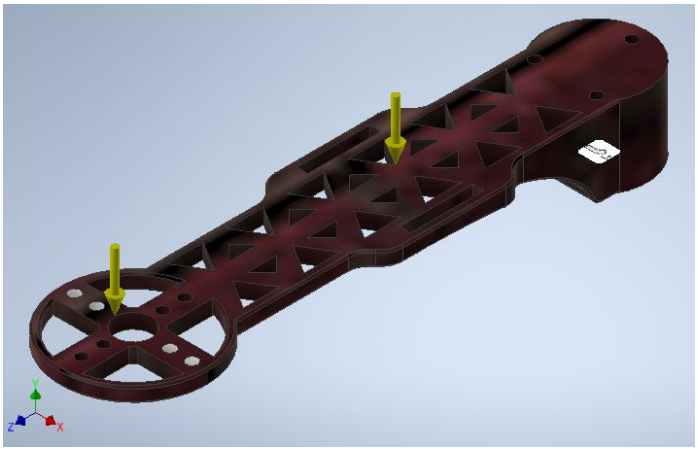


Figura 18: Fuerzas y restricciones para análisis de esfuerzos y deformación.

de $0.807mm$. Este punto se encuentra en la punta contraria al punto de anclaje del brazo. Y el desplazamiento máximo concuerda con la teoría, pues es el punto con mayor distancia en voladizo. La relación entre el desplazamiento máximo y el largo total de la pieza se determinó a partir de:

$$razón \quad de \quad deformación(\,\%) = \frac{deformación}{longitud \quad del \quad brazo} = \frac{0.807mm}{160mm} * 100 = 0.504\,\%. \quad (23)$$

7.5. Manufactura del cuadricóptero

Para la impresión de las piezas se escogió el *PLA*, pues era el material con mayor resistencia y dureza comparado con el *ABS*. El *PLA* se encuentran a disposición en el laboratorio *D-Hive* de la Universidad del Valle de Guatemala por lo que cumple con el requisito de disponibilidad para su fabriación inmediata.

En la Figura 21 se encuentra el producto manufacturado a través de la impresión 3D. Tal

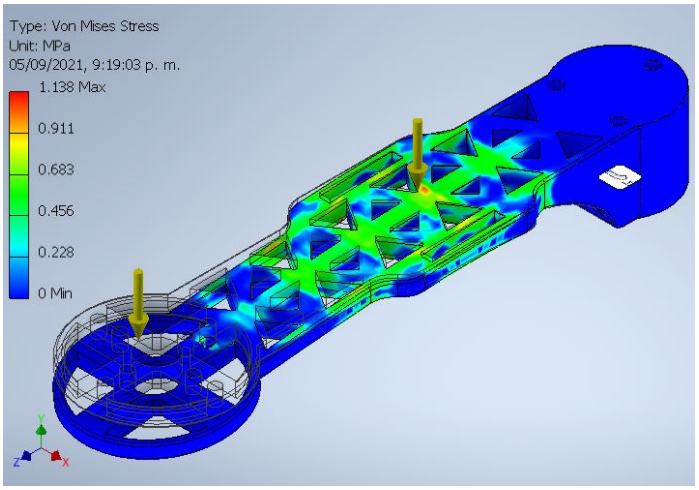


Figura 19: Esfuerzo Von Mises en brazo debido al peso de los componentes

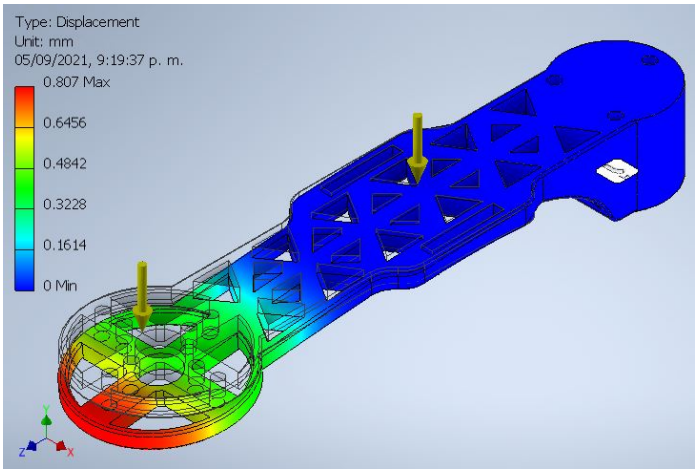


Figura 20: Desplazamiento eje Y en brazo del drone.

como se observa, la forma, el detalle y dimensiones cumplen con el modelado 3D desplegado en la Figura 15.

En la Figura 22 se muestra el drone con la base superior e inferior unidas a los brazos por medio de los tornillos y tuercas detallados en el Cuadro 4. Los motores sin escobillas también se aseguran en sus posiciones por medio de tornillos.

En la Figura 23 se observa el producto final del diseño para el drone. La conexión de los controladores electrónicos de velocidad con los motores sin escobillas se realizó por medio de las terminales con tornillos (pieza gris), desplegados en la Figura 24. Se utilizaron los cinchos plásticos para sujetar los *ESC*s en los carriles diseñados para crear una sujeción segura.

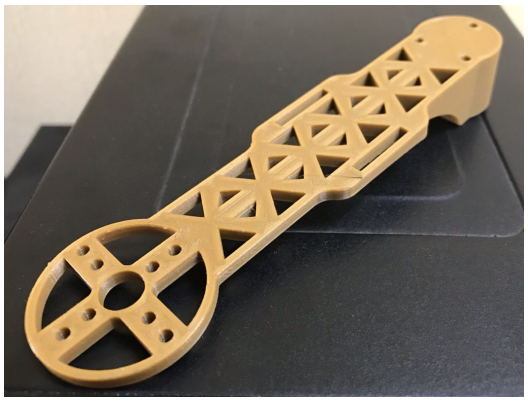


Figura 21: Brazo manufacturado por impresión 3D.

Figura 22: Estructura del cuadricóptero unida con elementos de sujeción.

7.5.1. Resultados del análisis de deformación

Para la validación del marco estructural del drone se realizó un estudio de deformación utilizando la herramienta de análisis de vídeo y herramienta de modelado para física en educación *Tracker* [19].

La prueba de deformación consistió en el análisis de dos fotografías. La primera fotografía fue el brazo del drone sin componentes sobre él, Figura 25. La base del brazo se encuentra restringida de movimiento gracias a un sargento (herramienta) sujeto a la mesa.

Se coloca un eje de coordenadas X y Y para que *Tracker* tenga un punto de referencia para realizar las mediciones. El eje X se hace coincidir con una de las líneas de referencia traseras para asegurar que está alineado con el brazo. Además, se añade una vara de calibración entre un par de líneas de la hoja colocada atrás del brazo. La distancia entre cada una de las líneas es de $1cm = 0.01m$. Esta vara de calibración es utilizada por la herramienta de análisis de vídeo para convertir las mediciones de píxeles a metros.

Una vez preparado el entorno de trabajo, se utilizó una regla provista por *Tracker* para medir la distancia de un punto específico en la zona de mayor deformación del brazo, la cual es la punta, respecto al eje X. Para asegurar que este punto sea el mismo a analizar en el brazo con pesos colocados, se utilizó la esquina de una de las aperturas donde se coloca el motor sin escobillas. El punto seleccionado se observa más claramente en la Figura 26. La distancia obtenida fue de $6.69E-3m = 6.69mm$.

El procedimiento anterior se repite para la imagen del brazo con los componentes en su

Figura 23: Cuadricóptero ensamblado y conexión de motores con controladores electrónicos de velocidad.

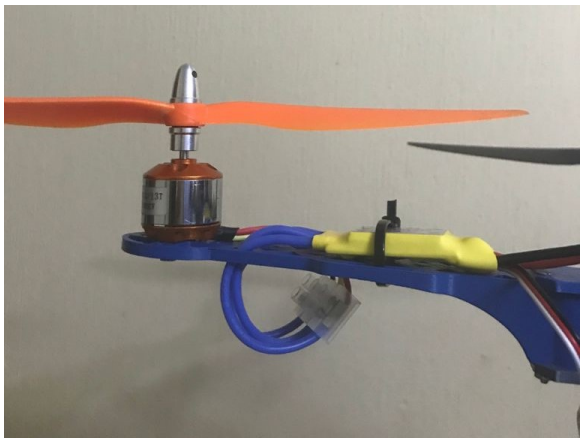


Figura 24: Componentes colocados en sobre el brazo del drone.

lugar, Figura 27.

Se utilizó nuevamente la regla de *Tracker* para obtener la distancia del mismo punto seleccionado previamente respecto al eje X, Figura 28. La medida obtenida fue $5.757E{-}3m = 5.757mm$.

Al tener las dos distancias, la diferencia entre ellas representó la deformación que hubo debido a la fuerza provocada por los dispositivos sobre el brazo.

$$deformación = distancia1 - distancia2 = 6.690mm - 5.757mm = 0.933mm \tag{24}$$

La deformación encontrada en la ecuación (24) equivale al 0.583 % de desplazamiento respecto a la longitud total del brazo. Este valor está dentro de un rango de deformación seguro por lo que el diseño se valida exitosamente.

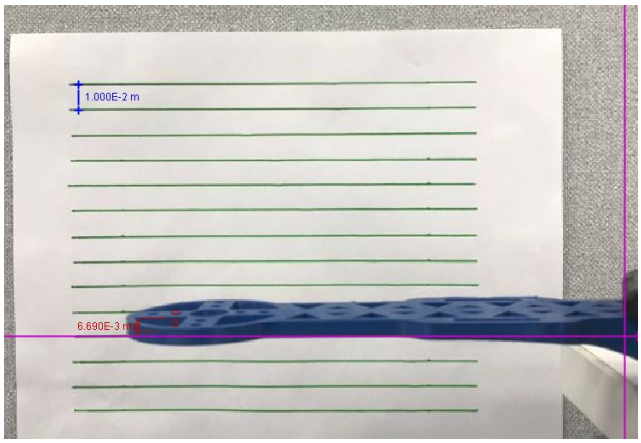


Figura 25: Disposición para prueba de deformación, brazo sin peso.

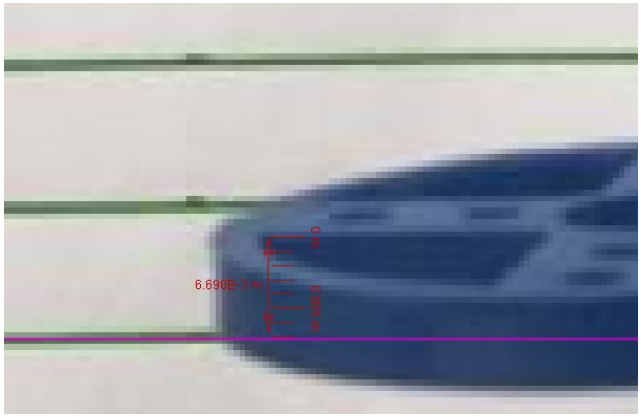


Figura 26: Distancia inicial de un punto del brazo respecto al eje X (6.690E-3m).

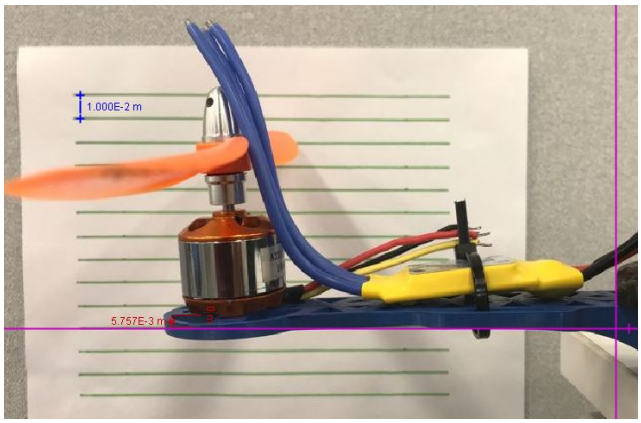


Figura 27: Disposición para prueba de deformación, brazo con pesos.

Comparación del método de elementos finitos con la deformación experimental

En el Cuadro 5 se presenta una comparación entre los métodos de análisis de deformación presentados previamente. La deformación determinada de forma experimental fue más

Figura 28: Distancia final de un punto del brazo respecto al eje X (5.757E-3m).

grande que la determinada por *software*, con una diferencia de $0.126mm$. A pesar de que el porcentaje de error es aparentemente alto (13.5 %), la diferencia en décimas de milímetro no tiene un impacto en la integridad de la estructura.

A pesar de la pequeña diferencia que se obtuvo con el método de elementos finitos, este método mostró ser bastante exacto pues toma en cuenta factores como material, dimensiones, fuerzas y restricciones y la diferencia en la deformación fue únicamente $0.126mm$. Esto demuestra que la herramienta de *Autodesk Inventor 2020* presenta una opción adecuada para el análisis de deformación de un modelo 3D previo a la manufactura, y así poder ajustar el modelo hasta que se obtengan los resultados deseados.

Elementos finitos	Experimental	% de error
Deformación	(mm)	
0.807	0.933	13.50 %.

Cuadro 5: Comparación entre deformaciones obtenidas a partir de proceso asistido por computadora vs experimentalmente.

El porcentaje de error de 13.5 % se adjudica a diferentes factores físicos que el *software* no puede tomar en cuenta. Entre ellos, el material es tomado como uniforme por el programa de computadora, pero al ser impreso en 3D existen imperfecciones en el producto final. También hay una pequeña diferencia entre el peso presentado por el proveedor y el peso real de los diferentes motores sin escobillas y controladores electrónicos de velocidad utilizados para cada brazo. Por último, en el método experimental existen un factor de diferencia como la selección del píxel exacto para comparar entre ambas imágenes, el cual puede verse afectado por la calidad y resolución de la imagen.

CAPÍTULO 8

Controlador de orientación y posición del cuadricóptero

8.1. Metodología

Para obtener el modelo diámico del cuadricóptero se estableció su modelo en el espacio de estados. Este análisis permitió representar el drone de una forma adecuada para el desarrollo de un controlador, que permita hacer que un sistema inestable se pueda estabilizar. Este modelo toma como entradas la información de sensores como la Unidad de Medición Inercial, o la pose y orientación del sistema de captura de movimiento OptiTrack. La salidas son las velocidades que deben tener los motores para que se genere la fuerza de empuje y momento necesario para modificar su orientación y posición.

El diseño del controlador se dividió en dos partes principales. La primera, fue la creación de un entorno de simulación utilizando el *software* MATLAB. En este espacio se generó una serie de archivos que permiten la creación de figuras para la visualización en tiempo real del comportamiento del drone y, más importante, la creación y diseño de controladores que pueden ser probados en esta plataforma previo a su implementación en el cuadricóptero físico.

La segunda parte consistió en la programación del controlador de vuelo que se implementará en el microntrolador ESP32. Se trabajó en un archivo principal *main.c*. Debido a que C en un lenguaje de nivel medio, es necesario realizar un procedimiento más extenso para operaciones de álgebra lineal en comparación con uno de alto nivel como MATLAB. Por lo tanto, se utilizó una librería de álgebra lineal desarrollada para el ecosistema Robotat por MSc. Miguel Zea. El controlador desarrollado en este trabajo se implementó en conjunto con las librerías y código principal desarrollados por Hans Burmester.

La presentación del controlador en este trabajo se realiza asumiendo que se cuenta con la información de entrada necesaria (ángulos, coordenadas X,Y y Z en el espacio 3D) para que

el cuadricóptero determine su posición y orientación. Esta información le permite, gracias al controlador diseñado, controlar la velocidad de los BLCD y estabilizar su orientación.

Para realizar la validación del controlador se realizaron tres pruebas diferentes. La primera consistió en la impresión de los datos de las velocidades de cada uno de los motores en conjunto con los ángulos *roll*, *pitch* y *yaw* en cada instante. Se graficaron estos valores y se realizó un análisis cualitativo del comportamiento que generaron en la orientación del drone. La segunda prueba consistió en la modificación en tiempo real del ángulo *yaw* deseado, y se observó la trayectoria que siguió el cuadricóptero hasta alcanzar el valor deseado. Por último, la tercera prueba consisitió en modificar los valores de *roll* deseado y *pitch* deseado, lo cual provocó que el drone siguiera una dirección en el sentido que fue modificado cada ángulo deseado. Finalmente, se realizó una prueba de velocidades solicitadas y obtenidas realmente a la salida de cada uno de los motores utilizando un tacómetro láser. Esta prueba permitió conocer una descompensación que tienen los motores por su construcción física y presenta una razón por la cual el controlador cuenta con dificultades para estabilizar en un espacio cerrado.

8.2. Entorno de simulación de vuelo en MATLAB

8.2.1. Archivos y carpetas

Se seleccionó el *software* de MATLAB para la programación del entorno de simulación debido a que es una plataforma que permite un fácil manejo de matrices, creación de gráficas y diagramas a partir de series de datos, y un lenguaje de programación de alto nivel. Esto último permitió utilizar funciones preprogramadas en MATLAB para realizar operaciones complejas de una forma sencilla.

El entorno de simulación del vuelo de un cuadricóptero se basó en un entorno similar al desarrollado en *University of Pennsylvania*. Este se encuentra en el curso *Robotics: Aerial Robotics*, Semana 4, de la plataforma de Coursera, impartido por el Dr. Vijay Kumar [20].

Para facilidad de uso y comprensión del usuario se creó el entorno como una serie de archivos individuales, los cuales son llamados desde el archivo principal mientras avanza la simulación. La carpeta principal se denomina "Simulación Drone 3D". Es necesario que al utilizar MATLAB, esta carpeta esté seleccionada como folder activo. La relación entre los archivos se describe en la Figura 30.

Como se observa en la Figura 29, la simulación se lleva a cabo con 4 archivos principales y una subcarpeta donde se encuentran 13 archivos extra. Se trabajó de esta forma para que el usuario, al momento de diseñar un controlador, tenga acceso a los archivos que manejan la simulación y no se confunda o realice cambios en los archivos que se encargan de la generación de figuras, transformaciones, etc. En la Figura 31 se despliegan los 13 archivos que se utilizan para generar las figuras de salida de la simulación, así como funciones desarrolladas para el correcto funcionamiento del entorno.

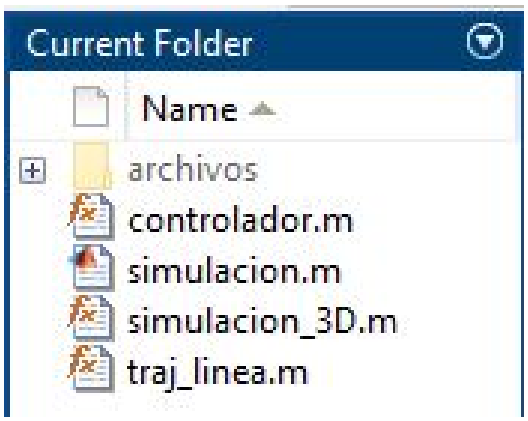


Figura 29: Archivos utilizados para creación del entorno de simulación.

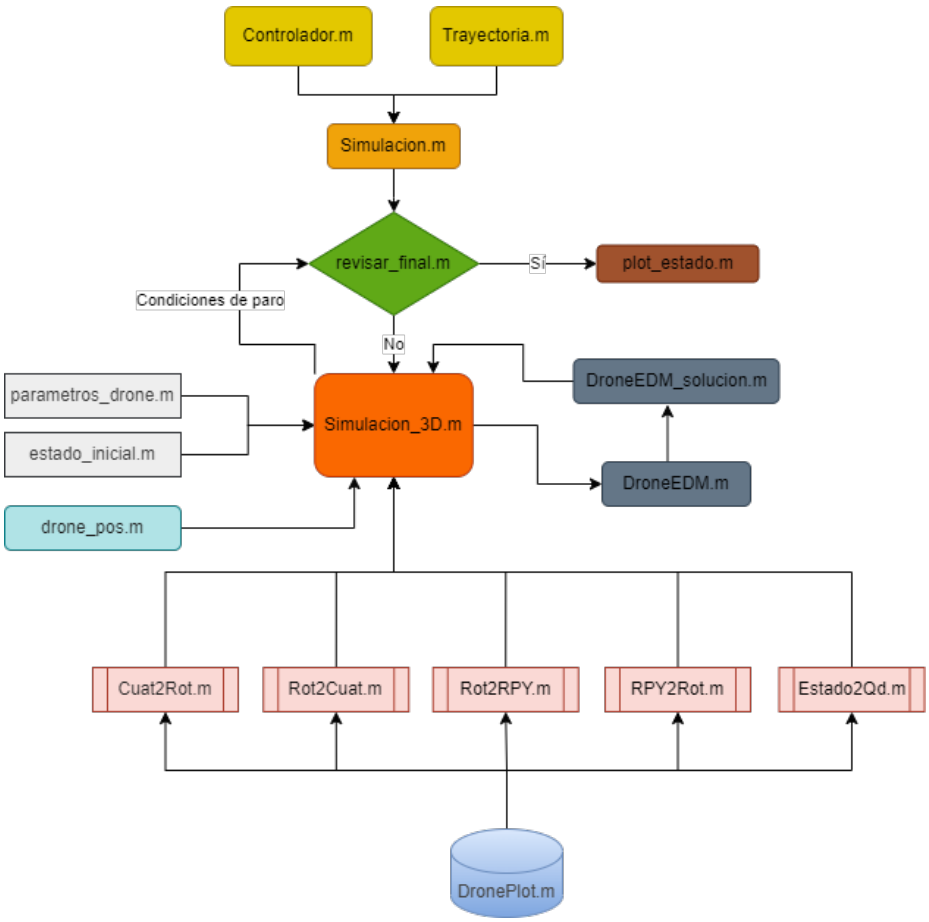


Figura 30: Relación entre archivos para simulación de vuelo en MATLAB.

8.2.2. Breve explicación de archivos

Todos los archivos utilizados en MATLAB cuentan con su documentación como comentarios. El usuario puede guiarse con esos detalles para realizar las modificaciones a la simulación. Por lo tanto, a continuación se realiza únicamente una explicación breve de la

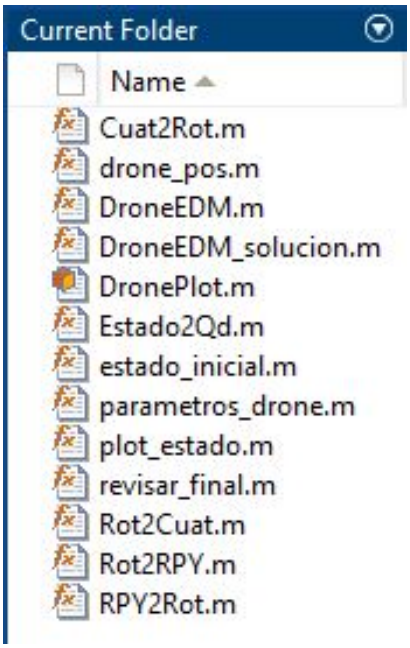


Figura 31: Funciones y archivos para generación de la simulación.

función de cada archivo y detalles importantes a tomar en cuenta. Los archivos se pueden acceder desde el repositorio de Github en Anexos.

Archivos principales

En esta sección se encuentran los 4 archivos desplegados en la Figura 29. El símbolo f_x que se encuentra sobre el ícono a la izquierda del nombre de cada archivo indica que 3 de los 4 son funciones. Por lo tanto, el archivo pirncipal es "simulacion.m".

1. **controlador.m**:

   En este archivo, Listado (8.1), se encuentra el espacio dedicado únicamente a la programación del controlador de vuelo. Se tiene como entradas de la función: el vector del tiempo para mantener la simulación en tiempo real, el estado acutal del drone (vector de estados), el estado al que sea desea llegar y los parámetros físicos del drone. Las salidas son la fuerza de empuje y los momentos que se rquieren para que el drone se mantenga o llegue a la posición deseada por el usuario. Por motivos de conflicto al introducir el código en este documento, se ignoran las acentuaciones en los comentarios. Sin embargo, en los archivos sí se acentúan las palabras.

```
1 function [F, M] = controlador(t, state, des_state, params)
2 % ----------------- Aqui empieza controlador --------------------
3 % Fuerza de empuje
4 F = params.mass*params.gravity; %Hover
5 % Momento
6 % M = [L(F2 - F4); L(F3 - F1); M1-M2+M3-M4]; M = [u2, u3, u4]';
7 %Constantes Roll:
```

```
8  kpr = 1;
9  kdr = 1;
10 u2 = kpr*(-state.rot(1))+kdr*(-state.omega(1));
11  %Constantes Pitch:
12 kpp = 1;
13 kdp = 1;
14 u3 = kpp*(-state.rot(2))+kdp*(-state.omega(2));
15  %Constantes Yaw:
16 kpy = 1;
17 kdy = 1;
18 u4 = kpy*(-state.rot(3))+kdy*(-state.omega(3));
19 M = [u2; u3; u4];  %-- Aqui termina controlador -----------
20 end
```

Listing 8.1: Código para controlador de vuelo.

Las variables u1, u2, u3 y u4, son las entradas para el controlador de vuelo, determinadas a través de las ecuaciones (13), (14), (15) y (22). El controlador implementado en este trabajo fue un proporcional-derivativo. Para cada uno de los ángulos (*roll*, *pitch*, *yaw*) se aplicó un PD, por lo que se realiza la comparación entre el estado actual y el estado deseado. El estado desado es igual a 0, para *hovering*, por lo que no aparece el término.

A pesar de haberse implementado un controlador PD para este trabajo, en un futuro puede aplicarse cualquier controlador que se desee. Únicamente es necesario realizar las conversiones necesrias para que la salida sea la fuerza de empuje (F) y los momentos respecto de cada eje (M).

2. **simulacion.m**

Este archivo, Listado (8.2), fue el encargado de asignar los valores deseados a través de *handles*. Estos permitieron utilizar funciones que requieren de otros archivos para funcionar correctamente. Como se observa en las líneas 13 y 16, se asigna la trayectoria que se desea seguir y el controlador diseñado, respectivamente. La última línea, 24, presenta la llamada a la función `simulacion_3D`, la cual lleva a cabo todas las funciones que combinan la ecuación de movimiento y el gráfico del drone.

```
1  %% Simulacion de vuelo Drone 3D
2  % Carlos Alonzo
3  % Ingenieria Mecatronica
4
5  %% Limpieza de Workspace y creacion de path
6  % Esta seccion se utiliza para limpiar el workspace
7  % e indicarle a Matlab que hay archivos que se utilizaran
8  % en la carpeta indicada.
9
10 close all;
11 clear;
12 addpath('archivos');
13
14 %% Seleccion de trayectoria
15 trajhandle = @traj_linea;
16
17 %% Seleccion de controlador
18 controlhandle = @controlador;
```

```
19
20 %% Empezar simulacion
21 % state - n x 13, con [x, y, z, xdot, ydot, zdot, qw, qx, qy, qz, p, ...
       q, r]
22
23 [t, state] = simulacion_3D(trajhandle, controlhandle);
```

Listing 8.2: Código de simulación. Este código prepara el ambiente de MATLAB para llamar a los archivos encargados de la simulación.

3. **simulacion_3D.m**

Para facilitar la comprensión de este documento, se separó el archivo por secciones las cuales se presentan a continuación. Las entradas de la función son: la trayectoria que se desea seguir y el controlador a utilizar, Listado (8.3). La salidas son el tiempo de simulación y el vector de estado del drone.

```
1  function [t_out, s_out] = simulacion_3D(trajhandle, controlhandle)
2
3  %Se agrega la ubicacion de archivos utiles para la simulacion y ...
       generacion de figuras
4  addpath('archivos');
5
6  % Simulacion en tiempo real activada
7  real_time = true;
8
9  % Tiempo maximo
10 max_time = 20;
11
12 % Asignacion de los parametros del drone
13 params = parametros_drone;
```

Listing 8.3: Parámetros de función simulación_3D. Entradas y salidas de la función.

El siguiente proceso, en el Listado (8.4) es la generación de figuras, en la cual se graficó el drone con una forma simplificada.

```
1  %% Creacion de Figuras
2  disp('Inicializando figuras...');
3  h_fig = figure;
4  h_3d = gca;          %Obtiene el eje actual de la figura a utilizar
5  axis equal
6  grid on
7  view(3);
8  xlabel('x [m]'); ylabel('y [m]'); zlabel('z [m]')
9  quadcolors = lines(1);
10 set(gcf,'Renderer','OpenGL')
```

Listing 8.4: Generación de figuras.

Utilizando la información de la trayectoria que se definió y el estado inicial del drone deseado se procedió a establecer las condiciones iniciales en el Listado (8.5) de la simulación. Cabe destacar que la función `estado_inicial` se describe más adelante en los archivos secundarios.

```
%% Condiciones iniciales
disp('Estableciendo condiciones iniciales...');
tstep    = 0.01; % determina time step en el que se devuelve la solucion
cstep    = 0.05; % intervalo de tiempo para captura de imagen
max_iter = max_time/cstep; % iteracion maxima
nstep    = cstep/tstep;
time     = 0; % tiempo actual
err = []; % errores durante la simulacion

% Obtencion de la posicion de inicio y paro
des_start = trajhandle(0, []);
des_stop  = trajhandle(inf, []);
stop_pos  = des_stop.pos;
x0    = estado_inicial(des_start.pos, 0);        %Yaw = 0
xtraj = zeros(max_iter*nstep, length(x0));
ttraj = zeros(max_iter*nstep, 1);
x = x0;    % estado inicial

%Tolerancia de error para posicion y velocidad
pos_tol = 0.01;
vel_tol = 0.01;
```

Listing 8.5: Establecimiento de las condiciones iniciales.

La sección del Listado (8.6) muestra el ciclo utilizado para modelar, controlar y graficar el comportamiento del drone desde el tiempo inicial hasta la máxima iteración. La primera iteración se encuentra dentro de una condición para separarla de las demás, pues es necesario hacer un proceso de inicialización del drone y condiciones del entorno. Además, este ciclo contó con tres revisiones de condiciones. La primera, se asegura de que el solucionador de la ecuación de movimiento ODE45 no se vuelva inestable. La segunda, se asegura de mantener la condición de tiempo real, es decir, pausa la simulación hasta asegurarse que la siguiente iteración está lista para funcionar. Por último, la tercera, se encarga de revisar si ya se alcanzó la condición final de la trayectoria y estado del drone tomando en cuenta la tolerancia establecida.

```
%% Simulacion
disp('Simulacion en curso....');
% Loop principal
for iter = 1:max_iter
    timeint = time:tstep:time+cstep;
    tic;         %Inicia medicion del tiempo
    % Inicializacion de plot para el cuadrupedo
    if iter == 1
        QP = DronePlot(1, x0, 0.1, 0.04, quadcolors(1,:), max_iter, ...
            h_3d);
        current_state = Estado2Qd(x);
        desired_state = trajhandle(time, current_state);
        QP.ActualizarDronePlot(x, [desired_state.pos; ...
            desired_state.vel], time);
        h_title = title(sprintf('iteracion: %d, tiempo: %4.2f', ...
            iter, time));
    end

    % Corre simulacion
    [tsave, xsave] = ode45(@(t,s) DroneEDM(t, s, controlhandle, ...
```

```
            trajhandle, params), timeint, x);
18      x    = xsave(end, :)';
19
20      % Guardar trayectoria
21      xtraj((iter-1)*nstep+1:iter*nstep,:) = xsave(1:end-1,:);
22      ttraj((iter-1)*nstep+1:iter*nstep) = tsave(1:end-1);
23
24      % Actualizar plot del drone
25      current_state = Estado2Qd(x);
26      desired_state = trajhandle(time + cstep, current_state);
27      QP.ActualizarDronePlot(x, [desired_state.pos; ...
            desired_state.vel], time + cstep);
28      set(h_title, 'String', sprintf('iteracion: %d, tiempo: %4.2f', ...
            iter, time + cstep))
29
30      time = time + cstep; % Actualizar tiempo de simulacion
31      t = toc;         %Termina medicion del tiempo
32
33      % Revision para asegurar que ode45 no se queda sin tiempo
34      if(t> cstep*500)
35          err = 'Ode45 Inestable';
36          break;
37      end
38
39      % Pausa para hacer simulacion en tiempo real
40      if real_time && (t < cstep)
41          pause(cstep - t);
42      end
43
44      % Check termination criteria
45      if revisar_final(x, time, stop_pos, pos_tol, vel_tol, max_time)
46          break
47      end
48  end
```

Listing 8.6: Ciclo para iteraciones de simulación.

Por último, en el Listado (8.7) se realiza un proceso de análisis de los datos obtenidos durante la simulación. Realizar este proceso después de la solución de la ecuación de movimiento permitió que el proceso fuese más estable. En esta sección se realizaron las figuras de la posición y la velocidad del drone durante el tiempo de simulación.

```
1  %% Post-procesamiento de resultados
2  % Truncar xtraj y ttraj
3  xtraj = xtraj(1:iter*nstep,:);
4  ttraj = ttraj(1:iter*nstep);
5
6  % Truncar las variables guardadas
7  QP.TruncarHistorial();
8
9  % Plot de posicion
10 h_pos = figure('Name', ['Posicion del drone']);
11 plot_estado(h_pos, QP.state_hist(1:3,:), QP.time_hist, 'pos', 'vic');
12 plot_estado(h_pos, QP.state_des_hist(1:3,:), QP.time_hist, 'pos', ...
       'des');
13 % Plot de velocidad
14 h_vel = figure('Name', ['Velocidad del drone']);
```

```
15 plot_estado(h_vel, QP.state_hist(4:6,:), QP.time_hist, 'vel', 'vic');
16 plot_estado(h_vel, QP.state_des_hist(4:6,:), QP.time_hist, 'vel', ...
       'des');
17
18 if(¬isempty(err))
19     error(err);
20 end
21
22 disp('Completada.')
23 t_out = ttraj;
24 s_out = xtraj;
25 end
```

Listing 8.7: Análisis de datos posterior a la simulación.

4. **traj_linea.m**

   El archivo del Listado (8.8) se utilizó para establecer la trayectoria que se deseaba seguir con el drone. En este caso, se utilizó el código para una línea, por lo que utilizando una función de grado 5 para la posición y sus derivadas para la velocidad y aceleración se estableció el comportamiento deseado del drone. Por último, se establecen los valores en el objeto correspondiente de estado deseado. Este código puede ser modificado con la trayectoria que se desee o generar otro archivo, cambiando el *handle* que se utilizó en simulacion.m.

```
1  function [desired_state] = traj_linea(t, ¬)
2  t_max = 4;
3  t = max(0, min(t, t_max));
4  t = t/t_max;
5
6  pos = 10*t.^3 - 15*t.^4 + 6*t.^5;
7  vel = (30/t_max)*t.^2 - (60/t_max)*t.^3 + (30/t_max)*t.^4;
8  acc = (60/t_max^2)*t - (180/t_max^2)*t.^2 + (120/t_max^2)*t.^3;
9
10 % vector deseado de salida
11
12 desired_state.pos = [pos; pos; pos];
13 desired_state.vel = [vel; vel; vel];
14 desired_state.acc = [acc; acc; acc];
15 desired_state.yaw = pos;
16 desired_state.yawdot = vel;
17
18 end
```

Listing 8.8: Archivo encargado de generar la trayectoria a seguir para cada iteración.

Archivos secundarios

A continuación se presenta una breve descripción de cada acrhivo secundario y código, de los que se considera necesario para mayor claridad. Los archivos que se utilizan para conversiones no se presentan con el código porque puede ser obtenido desde el repositorio de GitHub.

1. **Cuat2Rot.m**

   Convierte un cuaternión en una matriz de rotación. El código fue escrito por Mellinger [5].

2. **drone_pos.m**

   La función del Listado (8.9) calcula las coordenadas de posición del drone respecto del marco de referencia global utilizando transformaciones con matrices de rotación.

```
function [ quad ] = drone_pos( pos, rot, L, H )

% pos       3x1 vector de posicion [x; y; z];
% rot       3x3 matriz de rotacion del cuerpo al marco global
% L         1x1 largo del drone

%If para revisar si se provee la altura del drone. De lo contrario, ...
    asigna un H = 0.05
if nargin < 4; H = 0.05; end

% wRb   = RPYtoRot_ZXY(euler(1), euler(2), euler(3))';
wHb   = [rot pos(:); 0 0 0 1]; % transformacion homogenea del cuerpo al
 %marco global

quadBodyFrame  = [L 0 0 1; 0 L 0 1; -L 0 0 1; 0 -L 0 1; 0 0 0 1; 0 0 ...
    H 1]';
quadWorldFrame = wHb * quadBodyFrame;
quad           = quadWorldFrame(1:3, :);
end
```

Listing 8.9: Cálculo de posición actual del drone.

3. **DroneEDMt.m**

   El listado (8.10) es una función para resolver la ecuación de movimiento del drone. Argumentos de entrada: tiempo, vector de estado, controlador, generador de trayectoria y los parámetros físicos. Salida: Derivada del vector de estado. Esta función únicamente prepara los argumentos y la salida. El cálculo de la derivada se realiza en DroneEDM_solucion.m.

```
function sdot = DroneEDM(t, s, controlhandle, trajhandle, params)
% Argumentos de entrada:
% t             - 1 x 1, tiempo
% s             - 13 x 1, vector de estado = [x, y, z, xd, yd, zd, ...
    qw, qx, qy, qz, p, q, r]
% controlhandle - handle de funcion definida en el controlador
% trajhandle    - handle de funcion en el generador de trayectoria
% params        - struct, salida de parametros_drone.m
%
% Salida:
% sdot          - 13 x 1, derivada del vector de estado s

% Convierte estado al struct qd para control
current_state = Estado2Qd(s);

% Obtiene el estado deseado
```

```
16 desired_state = trajhandle(t, current_state);
17
18 % Obtiene las salidas de control
19 [F, M] = controlhandle(t, current_state, desired_state, params);
20
21 % Obtencion de la derivada de vector de estado s
22 sdot = DroneEDM_solucion(t, s, F, M, params);
23 end
```

Listing 8.10: Función de preparación para argumentos de ecuación de movimiento.

4. **DroneEDM_solucion.m**

En el Listado (8.11) se recibe como entrada el estado actual del drone y se utilizan las entradas de control (F y M) obtenidas con el controlador. Se revisa que la capacidad de la fuerza y momento exigidos por el controlador esté adentro de los límites de los motores utilizados. Se convierte la información del estado en cuaternión y se obtiene la matriz de rotación que permite conocer la posición del drone respecto al marco referencial. Con la aceleración, velocidad angular y aceleración angular se determina el nuevo estado del drone y se obtiene el estado actual del cuadricóptero.

```
1  function sdot = DroneEDM_solucion(t, s, F, M, params)
2
3  %************ Ecuaciones de movimiento ***********************
4  % Limitacion de fuerza y momentos debido a limite de los actuadores
5  A = [0.25,                        0, -0.5/params.arm_length;
6       0.25,  0.5/params.arm_length,                        0;
7       0.25,                        0,  0.5/params.arm_length;
8       0.25, -0.5/params.arm_length,                        0];
9
10 prop_thrusts = A*[F;M(1:2)]; % Sin utilizar momentos alrededor del ...
       eje Z por limites
11 prop_thrusts_clamped = max(min(prop_thrusts, params.maxF/4), ...
       params.minF/4);
12
13 B = [                  1,                  1,                  1, ...
                       1;
14                       0, params.arm_length,                  0, ...
                             -params.arm_length;
15      -params.arm_length,                  0, params.arm_length, ...
                           0];
16 F = B(1,:)*prop_thrusts_clamped;
17 M = [B(2:3,:)*prop_thrusts_clamped; M(3)];
18
19 % Se asignan los estados
20 x = s(1);
21 y = s(2);
22 z = s(3);
23 xdot = s(4);
24 ydot = s(5);
25 zdot = s(6);
26 qW = s(7);
27 qX = s(8);
28 qY = s(9);
29 qZ = s(10);
30 p = s(11);
```

```
q = s(12);
r = s(13);

quat = [qW; qX; qY; qZ];
bRw = Cuat2Rot(quat);
wRb = bRw';

% Aceleracion
accel = 1 / params.mass * (wRb * [0; 0; F] - [0; 0; params.mass * ...
    params.gravity]);

% Velocidad angular
K_quat = 2; %fuerza la restriccion de magnitud 1 para el cuaternion
quaterror = 1 - (qW^2 + qX^2 + qY^2 + qZ^2);
qdot = -1/2*[0, -p, -q, -r;...
             p,  0, -r,  q;...
             q,  r,  0, -p;...
             r, -q,  p,  0] * quat + K_quat*quaterror * quat;

% Aceleracion angular
omega = [p;q;r];
pqrdot   = params.invI * (M - cross(omega, params.I*omega));

% Ensamble de derivada de s (sdot)
sdot = zeros(13,1);
sdot(1)  = xdot;
sdot(2)  = ydot;
sdot(3)  = zdot;
sdot(4)  = accel(1);
sdot(5)  = accel(2);
sdot(6)  = accel(3);
sdot(7)  = qdot(1);
sdot(8)  = qdot(2);
sdot(9)  = qdot(3);
sdot(10) = qdot(4);
sdot(11) = pqrdot(1);
sdot(12) = pqrdot(2);
sdot(13) = pqrdot(3);

end
```

Listing 8.11: Código para solución de ecuación de movimiento utilizando ode45 [21].

5. **DronePlot.m**

En los Listados (8.12) y (8.13) se definió una clase dentro de MATLAB. De esta forma se pueden asignar propiedades, y dentro de métodos definir funciones. A continuación se presentan las propiedades públicas que tiene la clase. El uso de clases facilita el manejo de datos y el acceso a las propiedades de forma sencilla.

```
classdef DronePlot < handle
    %Definicion de clase para visualizacion del drone

    properties (SetAccess = public)
        k = 0;
        qn;             % numero de drone
        time = 0;       % tiempo
```

```
        state;          % estado
        des_state;      % estado deseado [x; y; z; xdot; ydot; zdot];
        rot;            % matriz de rotacion del cuerpo al mundo

        color;          % color del drone
        wingspan;       % extension del drone (wingspan)
        height;         % altura del drone
        motor;          % posicion del motor

        state_hist        % historial de posicion
        state_des_hist;   % historial de posicion deseada
        time_hist;      % historial de tiempo
        max_iter;       % iteracion maxima
    end
    ...
end
```

Listing 8.12: Definición de clase en Matlab para facilitar el manejo de datos.

Las funciones asignadas a la clase son:

```
    methods
        % Constructor
        function Q = DronePlot(qn, state, wingspan, height, color,
        ...
        end
        % Actualizar el estado del drone
        function ActualizarEstadoDrone(Q, state, time)
        ...
        end
        % Actualizar estado deseado del drone
        function ActualizarEstadoDeseadoDrone(Q, des_state)
        ...
        end
        % Actualizar historial del drone
        function ActualizarHistorialDrone(Q)
        ...
        end
        % Actualizar posicion del motor
        function ActualizarPosicionMotor(Q)
        ...
        end
        % Truncar historial
        function TruncarHistorial(Q)
        ...
        end
        % Actualizar o del drone
        function ActualizarDronePlot(Q, state, des_state, time)
        ...
        end
    end
```

Listing 8.13: Funciones y métodos de clase en Matlab para facilitar el manejo de datos.

6. **Estado2Qd.m**

En el Listado (8.14) se tomaron datos del vector de estado y se forma un struct llamado qd, el cual permite acceder más fácilmente a los datos en operaciones de la simulación.

Esto se realiza una vez lugar de tener que hacer la separación del vector de estado cada vez que se realiza una operación lógica.

```
function [qd] = Estado2Qd(x)
%x es un vector de 1 x 13 con variables de estado [pos vel quat omega]
% qd es una struct que incluye campos de pos, vel, euler, y omega
%estado actual
qd.pos = x(1:3);
qd.vel = x(4:6);
Rot = Cuat2Rot(x(7:10)');
[phi, theta, yaw] = Rot2RPY(Rot);
qd.rot = [phi; theta; yaw];
qd.omega = x(11:13);
end
```

Listing 8.14: Obtención de datos del vector de estado y conversión a esctructura.

7. **estado_inicial.m**

La función descrita en el Listado (8.15) se utilizó en la primera iteración de la simulación y es la encargada de tomar el primer punto de la trayectoria deseada (posición, velocidad y aceleración) y las condiciones iniciales de los ángulos para colocar al drone en el punto de inicio.

```
function [ s ] = estado_inicial( start, yaw )

s     = zeros(13,1);
phi0   = 0.150;       %Valor dentro de la region de atraccion
theta0 = 0.150;      %Valor dentro de la region de atraccion
psi0   = yaw;
Rot0   = RPY2Rot(phi0, theta0, psi0);
Quat0  = Rot2Cuat(Rot0);
s(1)   = start(1);  %x
s(2)   = start(2);  %y
s(3)   = start(3);  %z
s(4)   = 0;         %xdot
s(5)   = 0;         %ydot
s(6)   = 0;         %zdot
s(7)   = Quat0(1);  %qw
s(8)   = Quat0(2);  %qx
s(9)   = Quat0(3);  %qy
s(10)  = Quat0(4);  %qz
s(11)  = 0;         %p
s(12)  = 0;         %q
s(13)  = 0;         %r

end
```

Listing 8.15: Establecimiento de las condiciones iniciales deseadas para la simulación.

8. **parametros_drone.m**

En el Listado (8.16) se colocaron los parámetros físicos del drone necesarios para poder tener una simulación que asemeje a las condiciones del drone real.

```
function params = parametros_drone()
m = 0.18; % kg
g = 9.81; % m/s/s
I = [0.00025,   0,          2.55e-6;
      0,        0.000232,   0;
      2.55e-6,  0,          0.0003738]; %Inercias respecto X, Y y Z
params.mass = m;
params.I    = I;
params.invI = inv(I);
params.gravity = g;
params.arm_length = 0.086; % m
params.minF = 0.0;
params.maxF = 2.0*m*g;
end
```

Listing 8.16: Declaración de parámetros físicos del drone y el entorno.

9. **plot_estado.m**

   Esta función es encargada de generar las figuras que despliegan la velocidad y posición del drone al final de la simulación. Automatizan el proceso de nombres de ejes, y unidades de medida para las variables, así como colores para distinguir fácilmente las gráficas.

10. **revisar_final.m**

    La función del Listado (8.17) se encarga de revisar si ya se han cumplido los criterios para detener la simulación, los cuales son: posición, velocidad y tiempo. De ser un proceso exitoso devuelve una condición de 1, y si existe alguna falla devuelve 2. Si aún no se ha llegado al final y no hay fallas, devuelve 0.

```
function [ terminate_cond ] = revisar_final( x, time, stop, pos_tol, ...
    vel_tol, time_tol)
% Inicializar
pos_check = true;    %Activa revision de posicion
vel_check = true;    %Activa revision de velocidad
pos_col_check = zeros(1, 3);
% Revisa posicion, velocidad y tiempo de espera para cada drone
pos_check = pos_check && (norm(x(1:3) - stop) < pos_tol);
vel_check = vel_check && (norm(x(4:6)) < vel_tol);
pos_col_check(1,:) = x(1:3)';
% revisa el tiempo de simulacion total
time_check = time > time_tol;

if (pos_check && vel_check)
    terminate_cond = 1; % El drone alcanza la meta, detiene. Exito.
elseif time_check
    terminate_cond = 2; % El drone no alcanza la meta dentro del ...
        tiempo establecido. Incompleto.
else
    terminate_cond = 0;
end
end
```

Listing 8.17: Código para revisión de condiciones de finalización para la simulación.

11. **Rot2Cuat.m**

    En este código se convirtió una matriz de rotación en un cuaternión. Función escrita por Mellinger [5].

12. **Rot2RPY.m**

    Utilizando esta función se convierte la matriz de rotación del cuerpo rígido respecto del marco referencial a los ángulos *roll*, *pitch* y *yaw*. Función en la que está basado el código escrita por Mellinger [5].

13. **RPY2Rot.m**

    Esta función tomó como entrada los ángulos *roll*, *pitch* y *yaw* como entrada y se devuelve la matriz de rotación correspondiente a al cuerpo rígido respecto al marco referencial.

8.2.3. Resultados de la implementación de controlador PD para *hovering*

El entorno de simulación devuelve el comportamiento del drone durante el tiempo designado de iteración. La Figura 32 muestra el segundo 0.05, y la iteración No. 1 para el drone. Se observa que el drone no estaba completamente vertical, esto debido a las condiciones iniciales determinadas para los ángulos, $\phi = 0.0150, \theta = 0.0150, \psi = 0$.

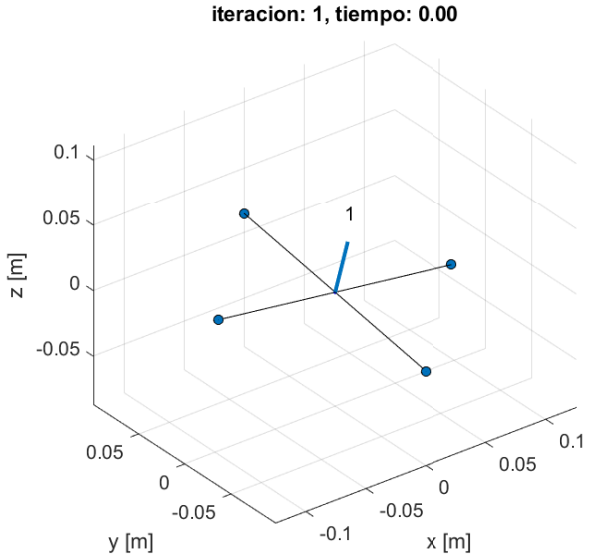


Figura 32: Iteración 1 del drone, 0.05 segundos. Condiciones iniciales seleccionadas.

Después de 20 segundos de simulación se obtuvo la ubicación final del drone. El entorno de simulación deja un punto color rojo con la posición del drone durante cada iteración, por lo que se puede observar el camino que se recorrió. Esta trayectoria se observa en la Figura 33.

Al finalizar la simulación, se obtuvieron los gráficos de posición y velocidad del drone durante los 20 segundos de vuelo, desplegados en las Figuras 34 y 35.

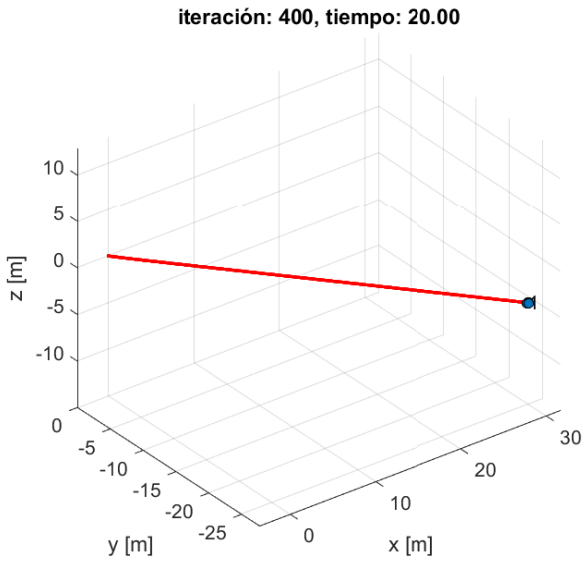


Figura 33: Iteración 400 del drone, 20 segundos. Condiciones iniciales seleccionadas. Posición y orientación final.

Debido a que en esta ocasión se utilizó un controlador únicamente para *hovering*, el cuadricóptero no tiene asignada una trayectoria a seguir. Por lo tanto, se colocan en cero estos valores. En caso de programar un controlador para seguimiento de una trayectoria, estos gráficos permiten comparar el valor acutal con el deseado por cada iteración.

8.3. Implementación del controlador en microcontrolador ESP32

La implementación del controlador PD desarrollado para mantener el drone en *hovering* requiere una implementación diferente en lenguaje C a la descrita en MATLAB. La diferencia radica en que existen funciones en MATLAB preprogramadas (lenguaje de alto nivel), las cuales no existen en C de forma nativa sino que deben ser desarrolladas. Para satisfacer esta necesidad se desarrollaron librerías. Estas cuentan con funciones que eficientizan una operación, al permitir que el proceso sea reutilizable en cualquier sección del código con un simple llamado a la función correspondiente.

Las librerías se dividen en 2 subconjuntos. El primero de ellos, desarrollado por MSc. Miguel Zea, contiene dos librerías. Una de ellas, `robotat_control.c`, y la segunda, `robotat_linalg.c`.

El segundo subconjunto, desarrollado por Hans Burmester, contiene 6 librerías. Estas en combinación son utilizadas para obtener los datos desde una Unidad de Medición Inercial MPU9250, procesar y enviar los datos al controlador, recibir las velocidades deseadas para los motores y, por último, generar las señales de PWM correspondientes para que los motores funcionen correctamente.

El código del controlador se programó en un archivo principal denominado *main.c*. La

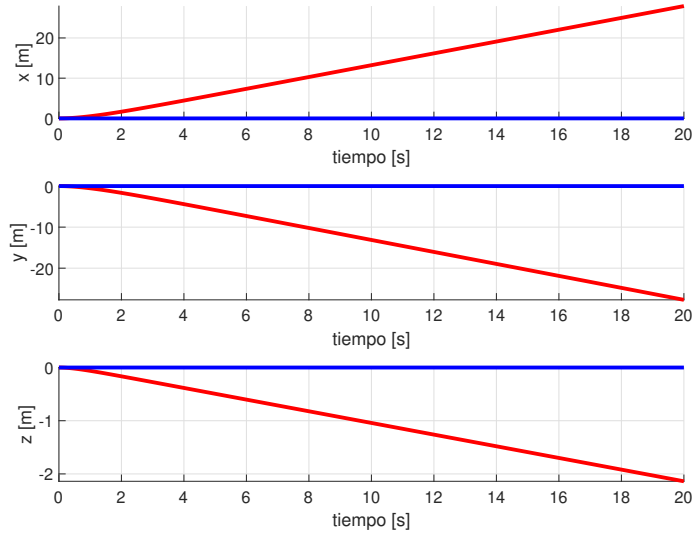


Figura 34: Posición del drone durante 20 segundos de vuelo

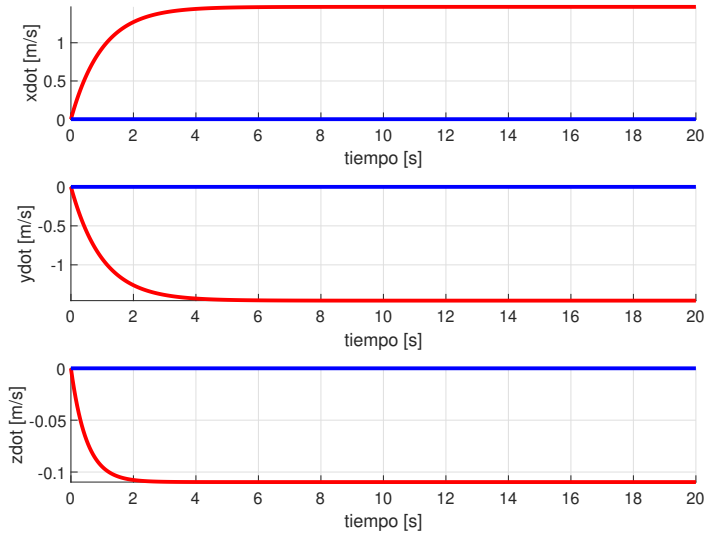


Figura 35: Velocidad del drone durante 20 segundos de vuelo

Figura 36 muestra la relación que existe entre este archivo y las librerías utilizadas.

8.3.1. Pseudocódigo de controlador PD

Se utilizó un pseudocódigo para simplificar y explicar de forma eficiente el procedimiento requerido para realizar el control de vuelo. La Figura 37 despliega la sección del pseudocódigo encargada del enlace entre las librerías mencionadas anteriormente con el archivo principal.

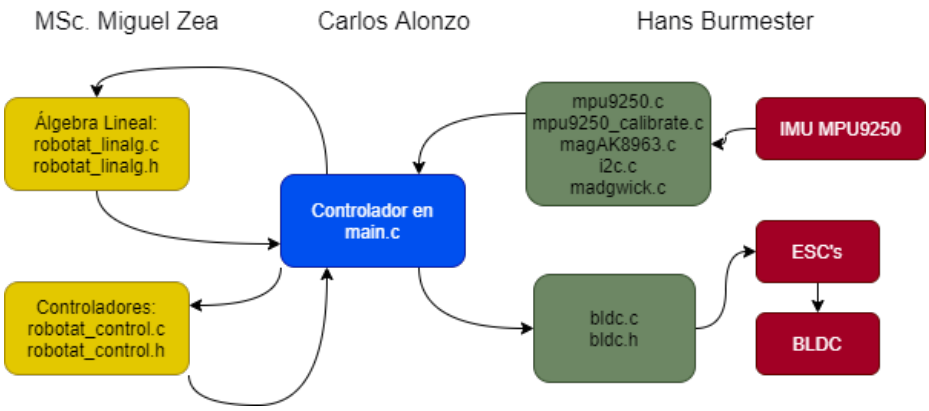


Figura 36: Interacción entre controlador desarrollado en archivo principal y librerías utilizadas

El ciclo encargado de repetir la lectura de los ángulos y por medio del controlador PD obtener las velocidades correspondientes de cada motor se observa en la Figura 38. Se destaca nuevamente que las acentuaciones en el código no se colocaron debido a que el editor de texto no las detecta correctamente en este documento. Sin embargo, en los códigos del repositorio de GitHub se encuentran las palabras correctamente acentuadas.

Pseudocodigo controlador PD - Declaraciones

```
//Carlos Alonzo
//Las acentuaciones no se colocaron porque el lector de texto no es capaz de leerlas

//Inclusion de librerias a archivo principal
#include "robotat_linealg.h"

declaracion de instancia de matrices tipo -->  matf32_t

declaracion de datos para matrices como arrays --> float

//declaracion de par metros del drone (masa, gravedad)
float m, g;

//Declaracion de variables para lectura de datos
float roll, pitch, yaw; //etc...

//Variables para PID (Hovering)
float desired_roll, desired_pitch; //etc...

//Declaracion de variables para velocidades de motor
float Vel_motor_des[4*1]; //etc...
```

Figura 37: Declaración de instancias y variables para controlador

8.3.2. Consideraciones del código en C

La librería de álgebra lineal utilizando lenguaje de programación C permite trabajar matrices como un vector de vectores. A su vez, las matrices utilizadas son estáticas lo que significa que debe indicarse su dimensión al declararlas y no puede modificarse después de esto. Este método vuelve ineficiente el trabajo con matrices dinámicas. Es decir, matrices que cambian de tamaño después de compilado el código. Para eficientizar el uso de memoria del ESP32 se utilizaron arrays de una sola dimensión, y se declaró un tipo de variable `matf32_-`

Ciclo de while para controlador

```
1
2  void main(){
3
4  //Lectura de yaw actual
5
6    while(1){
7
8    //Calculo de velocidad para roll, pitch y yaw
9    rolld = (roll_prev - roll)/dt;       //(ejemplo)
10     roll_prev = roll;
11
12     //Calculo de p, q, r
13
14     //Multiplicacion de matrices utilizando libreria algbera lineal
15
16     //Actualizacion de vector de estado X
17     X.p_data[0] = roll; //(ejemplo)
18
19     //C lculo del controlador restando Xss - x
20     matf32_sub(&Xss, &X,&e);
21     matf32_mul(&K, &e, &u2);
22
23     // calculo de u1 con aceleracion en z
24     u1 = wh + wF;
25
26     //Composicion del vector y matriz para obtener velocidades de motor
27     matf32_mul(w_transf, w_u, Vel_motor_des);
28
29   //Revision de velocidades dentro de capacidades de motores
30   w_motor1 = saturation(Vel_motor_des.p_data[0], 0, 7500);
31   }
32 }
```

Figura 38: Ciclo encargado de lectura de ángulos y cálculo de velocidades para motores

`t`. Una breve explicación de las funciones y estructura se presenta en el Cuadro 6. Dentro del código se asignó el array con los datos deseados como tipo *float*, y mediante la función `matf32_init()` se asigna el valor utilizando punteros a la variable `matf32_t`. Gracias a la declaración de número de filas y columnas en la función, la librería trabaja los datos como si fuese una matriz ordinaria.

Función	Parámetros	Descripción
`matf32_t`	Atributo *num_rows* Atributo *num_cols* Atributo **p_data*	Esta estructura se define para facilidad de lectura y manejo de datos de matrices. Contiene el número de columnas y filas de la matriz, así como un puntero hacia la información que contiene la matriz.

Continúa en la siguiente página

Cuadro 6 – *Continuación de la página previa*

Función	Parámetros	Descripción
`matf32_-init`	*instance* *num_rows* *num_cols* *p_data*	Esta función es utilizada para inicializar una matriz de tipo `matf32_t` con la información que se desea almacenar dentro de ella con *p_data*. Se utiliza el número de filas y columnas para el manejo de los datos.
`matf32_mul`	*p_srca* *p_srca* *p_dst*	Esta función realiza la multiplicación de dos matrices, *p_srca* y *p_srcb*. El número de columnas de la primera matriz debe ser igual al número de filas de la segunda matriz. La matriz de salida *p_dst* no puede ser la misma que alguna de las matrices de entrada.
`matf32_sub`	**p_srca* **p_srca* *p_dst*	Esta función realiza la resta entre dos matrices utilizando los punteros **p_srca* y **p_srcb* y almacena el resultado en *p_dst*. Las dimensiones de ambas matrices deben ser iguales.

Cuadro 6: Estructuras y funciones utilizadas para controlador implementado en lenguaje C.

```
1 //Descripcion para declaracion de matrices en el controlador
2 matf32_t A;
3 float Adata[] = { 1, 2, 3,
4                   4, 5, 6};
5 matf32_init(&A, 2, 3, Adata);
```

Listing 8.18: Código para delcaración y manejo de matrices en C.

En la sección de código desplegada el Listado (8.18) se asignó a la variable A, los datos del array Adata. Al realizar operaciones entre matrices, fue necesario conocer las dimensiones del resultado y de esta forma reservar un espacio de memoria para almacenarlo. A continuación se presenta un ejemplo en el Listado (8.19).

```
1 //Variables para  velocidades de motor
2 float w_transf[] = {1,  0, -1,  1,
3                     1,  1,  0, -1,
4                     1,  0,  1,  1,
5                     1, -1,  0, -1};
```

```
6  float Vel_motor_des[4 * 1];    %Se reserva memoria (4x4*4x1=4x1)
7  float w_u[4 * 1];
8      w_u = {u1, pidx, pidy, pidz};
9      matf32_mul(w_transf, w_u, Vel_motor_des);
```

Listing 8.19: Código para enlace de datos y matrices en librería de C.

8.3.3. Resultados de las pruebas de vuelo para *hovering*

Para realizar la validación del controlador se subió el código compilado a través de VS Code. El ESP32 se conectó a la Unidad de Medición Inercial MPU9250 y a los controladores electrónicos de velocidad y motores sin escobillas mencionados en la sección 7.2. De esta forma, los ángulos *roll*, *pitch* y *yaw* se actualizan en tiempo real, el controlador de vuelo calcula las velocidades de cada motor necesarias para llevar el drone a la orientación deseada y las envía a los controladores para la señal de PWM. En esta validación se analiza el comportamiento del controlador y no el tiempo de respuesta del mismo, por lo que se utilizaron valores de constantes KP y KD que cumplieran el objetivo de mantener el drone en *hovering* pero la selección de las constantes adecuadas para una respuesta más rápida se seleccionarán más adelante.

Parámetros de la prueba

- **Estado deseado:** $X_{ss} = [\phi_{des} \quad \theta_{des} \quad \psi_{des} \quad p_{des} \quad q_{des} \quad r_{des}]' = [0 \quad 0 \quad yaw^*_{actual} \quad 0 \quad 0 \quad 0]'$.
  *El yaw_{actual} se toma la primera lectura de la IMU para determinar el yaw deseado. Esto debido a que no se desea que el drone rote sobre su eje vertical, sino mantenga su vuelo en el punto de inicio.
- **w_F:** 5000 RPM.
- **Frecuencia de muestreo:** 100 Hz.
- **escala:** 0.2.
- **KP_Roll:** 1000*escala.
- **KD_Roll:** 3000*escala.
- **KP_Pitch:** 1000*escala.
- **KD_Pitch:** 3000*escala.
- **KP_Yaw:** 1000*escala.
- **KD_Yaw:** 3000*escala.
- **Datos analizados:** 1000.

En la Figura 39 se observan los resultados de una prueba inclinando el drone únicamente respecto al eje X, lo cual modifica el valor de roll. La relación entre velocidad de cada motor y el ángulo roll se comprende de mejor forma en las Figuras 40 y 41.

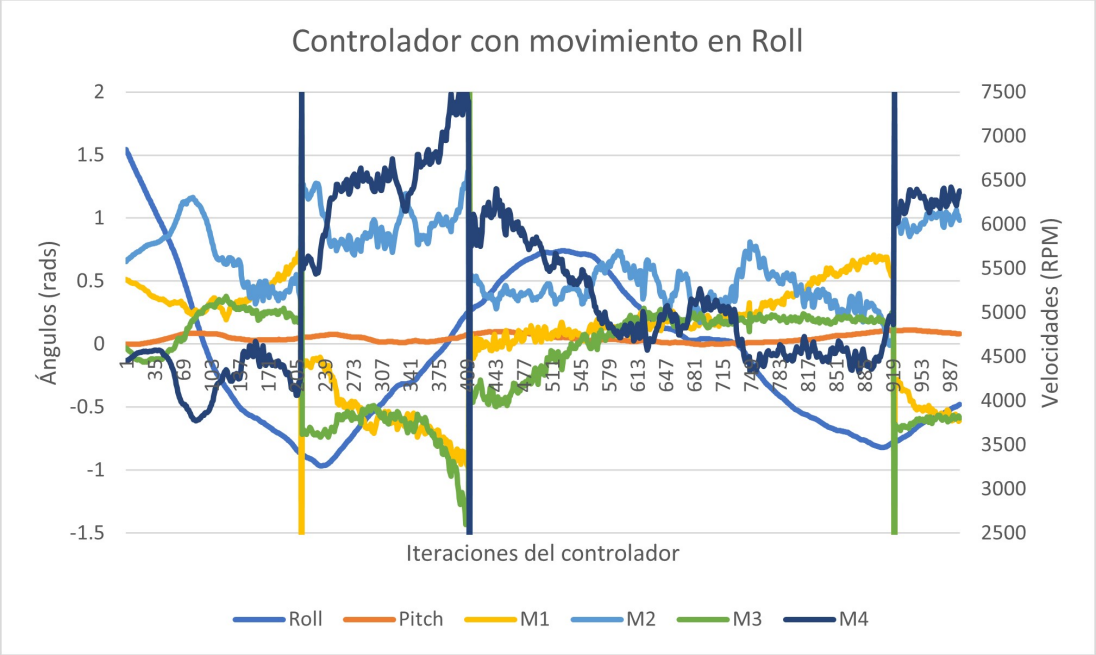


Figura 39: Prueba de controlador modificando únicamente el valor de roll.

En la Figura 40 se observa el cambio de roll a través de las 1000 iteraciones que se realizaron con el controlador. Se realizó un cambio de ángulo positivo (1.5 radianes) llegando a -1 radianes. Llegando al mínimo se rotó nuevamente en dirección opuesta. En total, se analizaron dos mínimos y un máximo para observar la respuesta de los motores. El ángulo de pitch se dejó lo más cercano posible a 0 radianes para que la respuesta de los motores fuese más clara al analizar un solo eje de rotación.

Analizando la primera pendiente negativa de roll, se observa que los motores 2 y 4 son los que realizan el mayor cambio. El motor 2 aumenta su velocidad y el motor 4 la disminuye. Esto hace sentido ya que se encuentran cruzados, por lo que al generar más fuerza en uno y disminuirla en el otro se genera el momento requerido sobre el eje X para girar el dron. Al mismo tiempo, los motores 1 y 3 mantienen una velocidad similar entre ellos, pues si la velocidad variase, se crearía un momento sobre el eje vertical (Yaw).

En la siguiente etapa se observa una pendiente positiva, la cual pasa del mínimo negativo hacia cero. El controlador detecta el cambio y envía más velocidad a los motores 2 y 4, y la disminuye en los motores 1 y 3. La pendiente de velocidad (aceleración) es aproximadamente la misma pero con signo opuesto entre ambos pares de motores.

La siguiente etapa es un punto cercano al cero (roll deseado) por lo que las velocidades de los 4 motores se van acercando unas a otras, y llegan a un punto donde se quedan iguales (iteración 721), Figura 41, que es el punto donde roll es cero en la Figura 40.

La última etapa presenta el mismo comportamiento que la etapa 2, donde los motores 2 y 4 aumentan su velocidad y los motores 1 y 3 la disminuyen.

La misma prueba de controlador se realizó modificando el ángulo de pitch y dejando estable en cero el de roll. De esta forma se observó el comportamiento del controlador

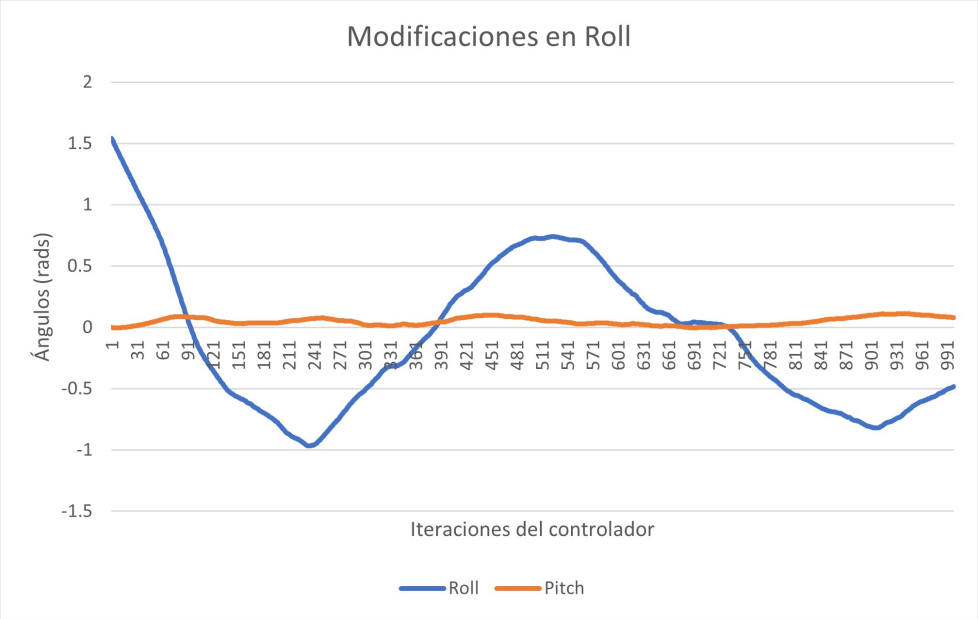


Figura 40: Comparación de ángulos modificando únicamente el valor de roll.

respecto al eje Y del drone. En la Figura 42 se observa el resultado superpuesto de los ángulos y la velocidad de los motores como reacción del controlador ante la variación entre la orientación actual y la deseada. Para un análisis seccionado se presentan las Figuras 43 y 44, donde se separa el comportamiento de los ángulos y la respuesta de la velocidad de los motores.

Para la prueba respecto al eje Y se utilizaron dos máximos y un mínimo en el comportamiento de pitch.

En la primera etapa, la pendiente positiva de pitch provoca que los motores 1 y 3 presenten comportamientos opuestos. El motor 1 aumenta su velocidad mientras el 3 la disminuye, teniendo sus picos y valles cerca de la misma iteración. Sin embargo, al momento de tener una pendiente negativa desde el primer máximo local hasta el cero el comportamiento es opuesto. El motor 1 disminuye su velocidad drásticamente, mientras el motor 3 lo aumenta. El comportamiento es similar hasta llegar al mínimo y se repite la etapa anterior hasta llegar al cero.

La última etapa muestra claramente que las velocidades de los motores se encuentran en pares. Los motores 1 y 3 tienen la misma velocidad y se mantiene constante porque pitch ya se encuentra en el valor deseado. Los motores 2 y 4 comparten la misma característica. El comportamiento en pares es importante debido a que una variación entre los pares de motores implica que se genera un momento respecto del eje vertical (Z o Yaw) y el drone no estaría en *hovering*. De esta forma, se validó que el controlador presenta el comportamiento deseado y fue capaz de responder ante cambios en los ángulos, respetando el modelo dinámico planteado para su funcionamiento.

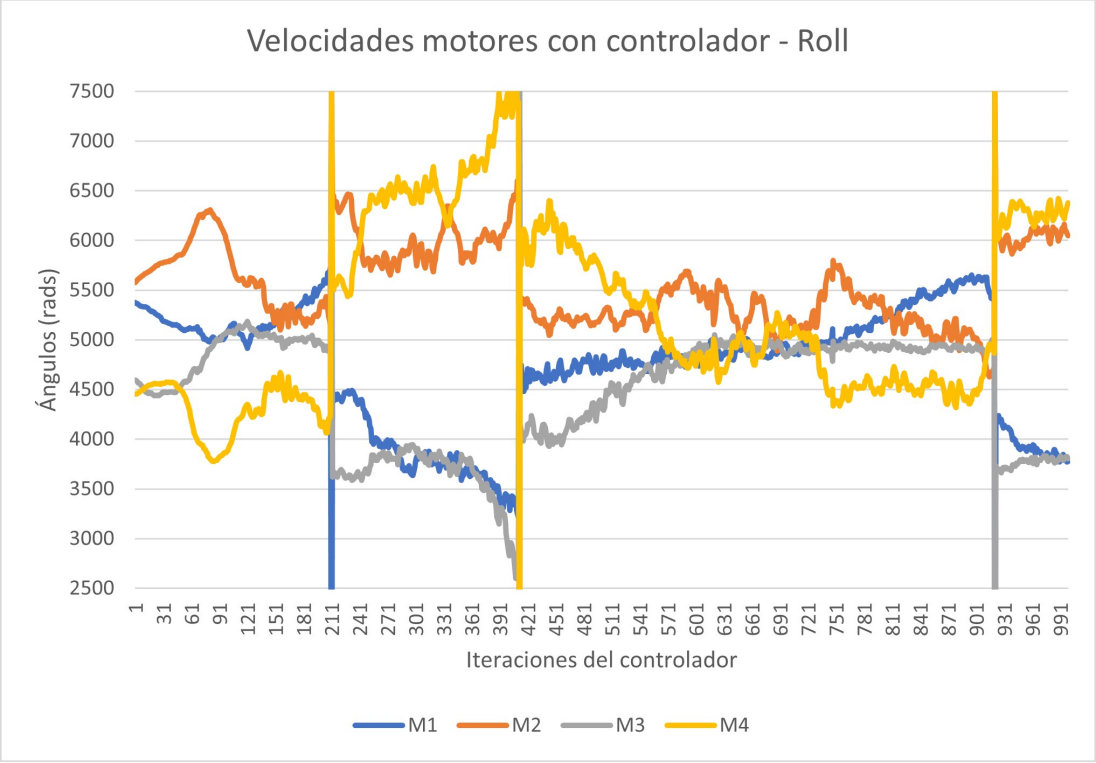


Figura 41: Comparación de velocidad de motores modificando únicamente el valor de roll.

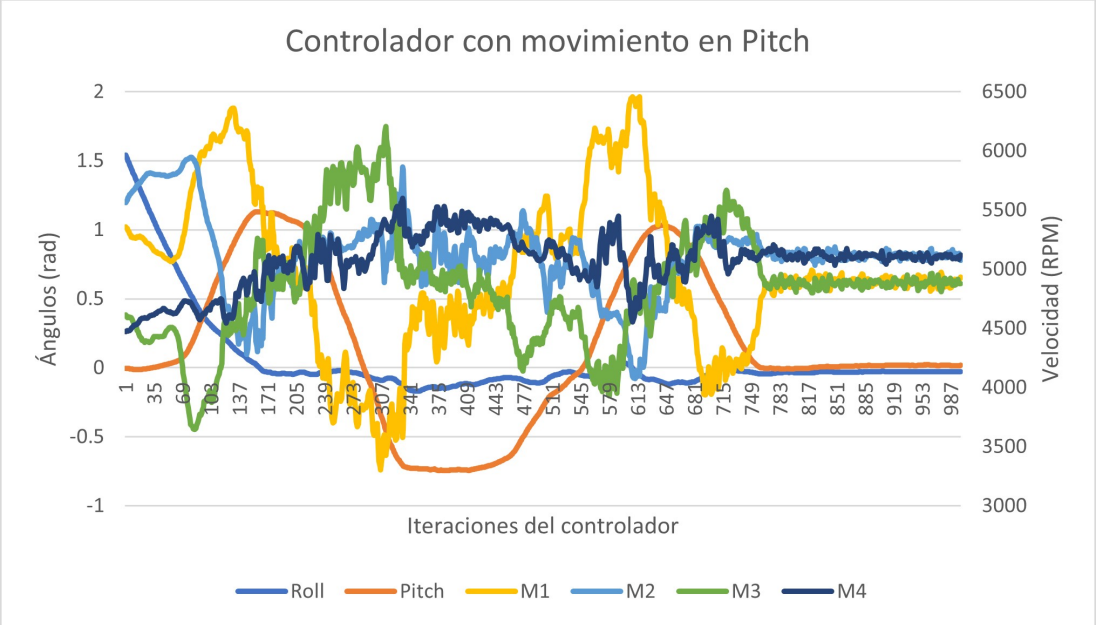


Figura 42: Prueba de controlador modificando únicamente el valor de pitch.

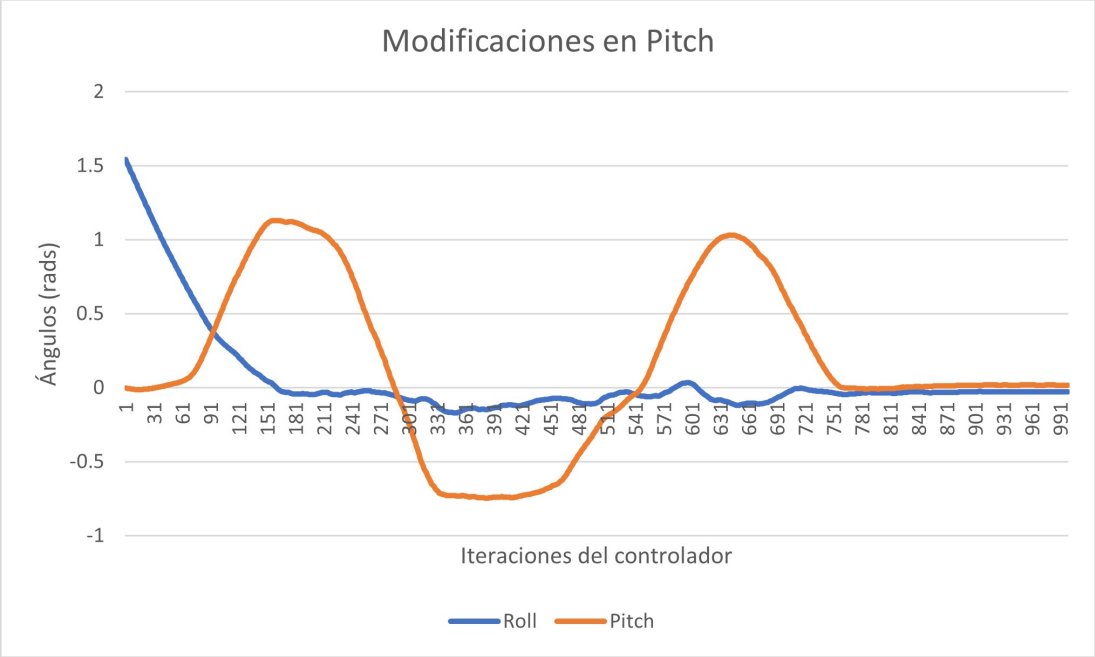


Figura 43: Comparación de ángulos modificando únicamente el valor de pitch.

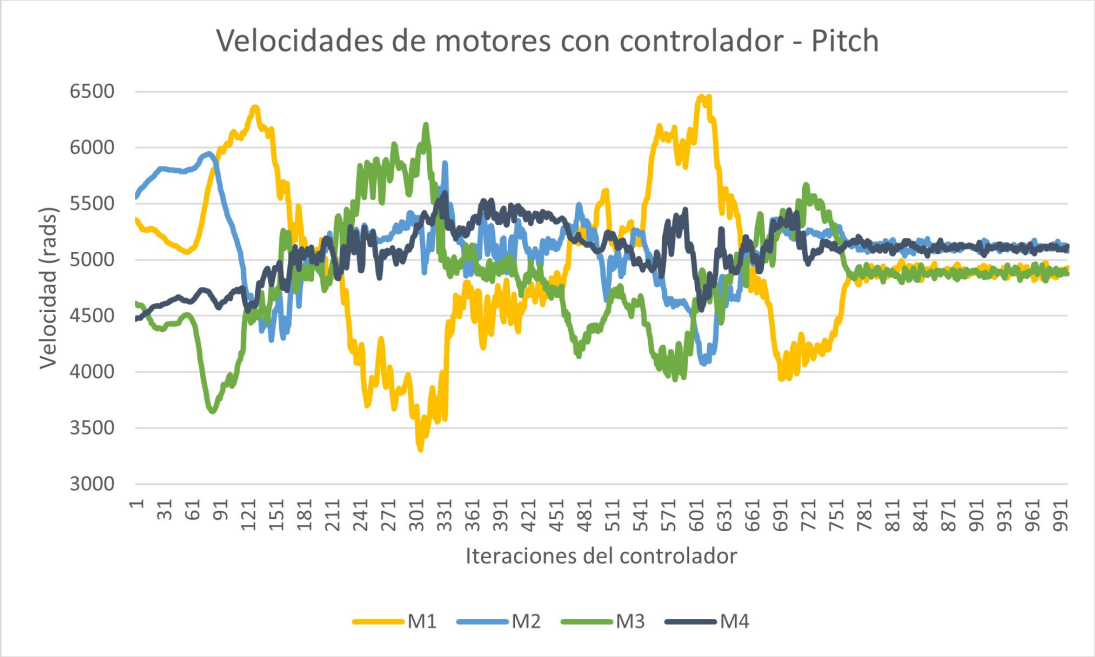


Figura 44: Comparación de velocidad de motores modificando únicamente el valor de pitch.

8.3.4. Resultados prueba modificando valor de *yaw* deseado

Para la segunda prueba de validación del controlador se analizó únicamente el controlador de vuelo para el ángulo *yaw*. Para lograrlo se disminuyó la velocidad de empuje en estado estable del drone "wh" a 3000 RPM, lo cual permitió que el drone tuviese suficiente fuerza

para levantar ligeramente el drone de la plataforma, pero no la suficiente fuerza para hacerlo despegar completamente. En este punto se evitó que el controlador actúe sobre *roll* y *pitch*.

Posteriormente se modificó el valor del ángulo *yaw* utilizando la red de comunicación Wifi a través del protocolo MQTT. Esta conexión y la forma de recibir los datos se explica en el siguiente capítulo.

Para el seguimiento de la trayectoria se utilizó el *software Tracker* el cual es una herramienta de análisis de vídeo la cual permite seleccionar una caracterísitca del vídeo y hacer un rastreo de la misma durante un tiempo dado. Dentro de esta herramienta se estableció un sistema de ejes coordenados los cuales coinciden con los brazos del drone, Figura 45.

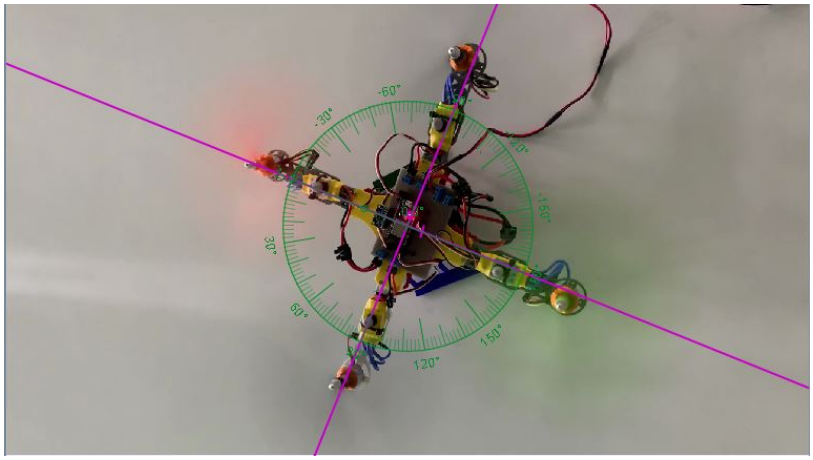


Figura 45: Ejes coordenados y disposición del drone para prueba de *yaw* deseado modificado.

Utilizando una herramienta de transportador se colocó el ángulo cero sobre el Motor1, el cual al tener una hélice de color anaranjado presenta un punto sencillo de destacar para el *software*.

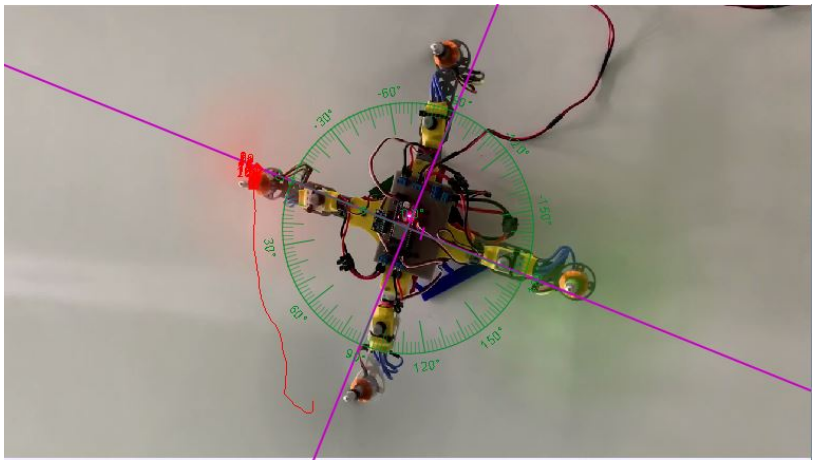


Figura 46: Trayectoria seguida por el controlador de vuelo para *yaw* deseado de 90°.

En la Figura 46 se observa de color rojo la trayectoria que siguió el drone al hacer un giro de 90° en *yaw*. La dirección de giro corresponde al *yaw* positivo según la regla de la mano derecha, por lo que el giro en contra de las agujas del reloj valida el controlador de vuelo

para el ángulo de *yaw*. El ángulo alcanzado fue de 84.8°, lo cual representa una variación de 5.78 % respecto al ángulo deseado. Los cálculos y resultados de esta prueba se encuentran más detallados en el siguiente capítulo.

8.3.5. Resultados prueba *roll* y *pitch* deseados

La tercera prueba realizada para validación del controlador consistió en modificar el valor deseado de los ángulos *roll* y *pitch* de forma que al no ser igual a cero y provocar una inclinación del drone, esta inclinación generaría que el drone avance en la dirección hacia donde la fuerza resultante de empuje apunte.

Con el propósito de observar la dirección en la que se desplaza el drone se colocó el drone en una superficie que dibuja los 4 cuadrantes de ejes coordenados y a cada cuadrante se le asignó el nombre dependiendo de la dirección y signo que provocaría que el drone se mueva en ese sentido.

```
1  //-----------------CONSTANTES CONTROLADOR--------------------------------
2
3  #define SCALE (0.0006f)
4  #define KP_roll (25000*SCALE)
5  #define KD_roll (7000*SCALE)
6
7  #define KP_pitch (90000*SCALE)
8  #define KD_pitch (6000*SCALE)
9
10 #define KP_yaw (150000*SCALE)
11 #define KD_yaw (375000*SCALE)
12
13 #define des_roll (M_PI/12)
14 #define des_p (0)
15 #define des_pitch (0)
16 #define des_q (0)
17 #define des_yaw (0)
18 #define des_r (0)
```

Listing 8.20: Sección de código para modificación de ángulos *roll* y *pitch* deseados (Ángulo *roll* deseado modificado).

El Listado 8.20 despliega la sección de código en el archivo principal `main.c`, "Constantes controlador", que se utilizó para modificar el valor deseado de los ángulos de rotación. En el caso particular del Listado 8.20 se modificó el valor deseado de *roll* con un valor positivo de $\frac{\pi}{12}$ equivalente a 15°. Es importante que el valor ingresado como ángulo deseado esté en radianes para un correcto funcionamiento del controlador.

La Figura 47 muestra la disposición utilizada para la prueba. En donde, el cuadrante I es la dirección que el drone seguiría al modificar el ángulo de *roll* con un valor positivo. El cuadrante II es la dirección que seguiría si se modificar *pitch* con un valor positivo. De igual forma, si se modificacn los ángulos negativos, las direcciones seguidas por el drone son los cuadrantes opuestos a los de su contraparte positiva.

La Figura 48 muestra el comportamiento del drone con el valor de *pitch* modificado con

Figura 47: Colocación de direcciones para prueba de modificación de ángulos de rotación.

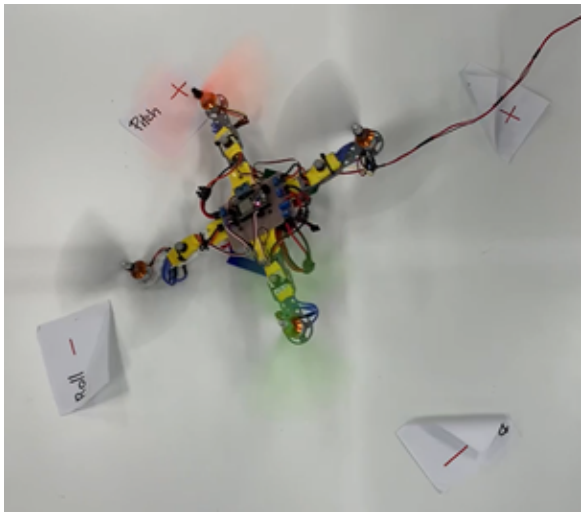


Figura 48: Controlador dirigiendo drone hacia *pitch* positivo.

$\frac{\pi}{12}$ equivalente a 15° positivo. Se observa que el drone se dirige al cuadrante II también nombrado "*Pitch+*". Por lo que presenta el comportamiento esperado para el controlador de vuelo.

En la Figura 49 se observa al drone en dirección "*Pitch-*" o cuadrante IV. El valor utilizado para *pitch* deseado fue de $-\frac{\pi}{12}$ equivalente a $-15°$.

En la Figura 50 se muestra el drone en dirección "*Roll+*" o cuadrante I. De igual forma, el valor utilizado para *roll* deseado fue de $+\frac{\pi}{12}$ equivalente a $+15°$.

Por último, en la Figura 51 se muestra el drone en dirección del cuadrante III o tam-

Figura 49: Controlador dirigiendo drone hacia *pitch* negativo.

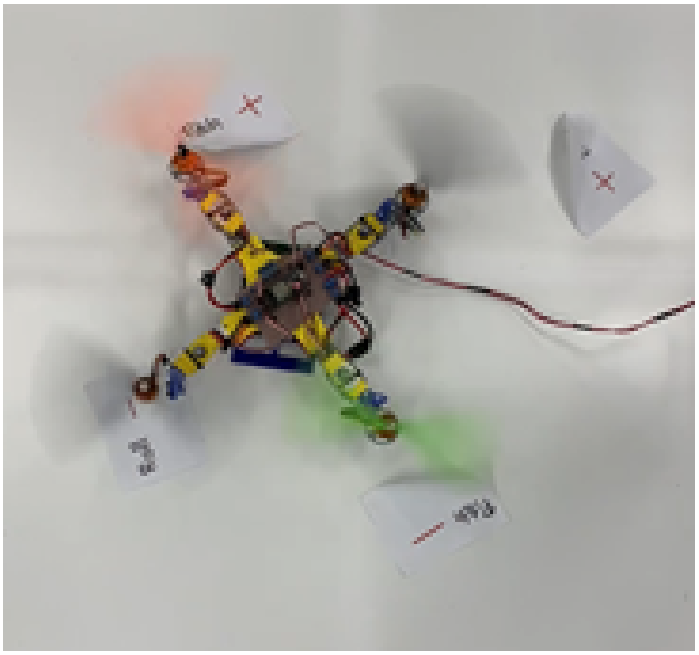


Figura 50: Controlador dirigiendo drone hacia *roll* positivo.

bién nombrado "*Roll-*". De la misma forma, el valor utilizado para *roll* deseado fue de $-\frac{\pi}{12}$ equivalente a $-15°$.

El comportamiento obtenido del drone fue el esperado al realizar la modificación ligera de ángulo de rotación deseado. La orientación deseada fuera del estado de equilibrió provocó que el drone se quisiera inclinar y la fuerza neta de empuje de los 4 motores quedara inclinada a su vez, por lo que provocó un desplazamiento en la dirección que apunta dicha fuerza. Por

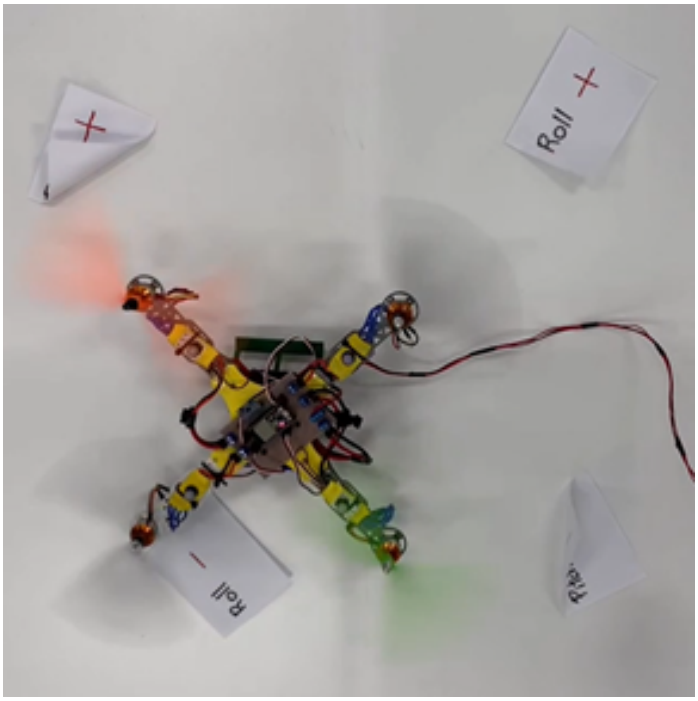


Figura 51: Controlador dirigiendo drone hacia *roll* negativo.

lo tanto, el controlador de vuelo se validó para los ángulos de rotación *roll* y *pitch*.

8.3.6. Resultados prueba con tacómetro láser

Debido a comportamientos de aparente desequilibrio entre las velocidades de los motores se utilizó el sensor *Fluke 810 Vibration Sensor* para realizar la medición de la velocidad angular de dos de los motores. El sensor de vibraciones utilizado cuenta con un tacómetro, el cual utiliza un emisor láser el cual se apuntó hacia el eje de los motores. Al mismo tiempo. fue necesario colocar una tira delgada de cinta reflectiva en el eje. Esta cinta refleja el láser y el sensor es capaz de utilizar la cantidad de veces que detecta la cinta durante un intervalo de tiempo para determinar la velocidad angular del eje, como se muestra en la Figura 52.

Se utilizaron once diferentes velocidades solicitadas de forma manual al controlador de vuelo. Es decir, en lugar de que el controlador de vuelo determinara la velocidad de giro de cada uno de los motores, se seleccionó un valor constante de giro tanto para el motor 1 como para el motor 2. De esta forma se aseguró que no hubiese variación por factores como inclinación de la *IMU*. Las velocidades utilizadas comenzaron en 2500 RPM, realizando saltos de 200 RPM con la siguiente velocidad hasta alcanzar los 4500 RPM. 2500 RPM fue la velocidad determinada experimentalmente como velocidad fuera de la zona muerta. Velocidades debajo de esta provocan comportamientos erráticos del motor o simplemente no gira.

En la Figura 53 se despliegan los valores obtenidos de velocidades angulares reales con el tacómetro láser y su comparación con el valor solicitado de forma manual para el motor 1. Se observó que existe una diferencia significativa entre las velocidades, teniendo una diferencia

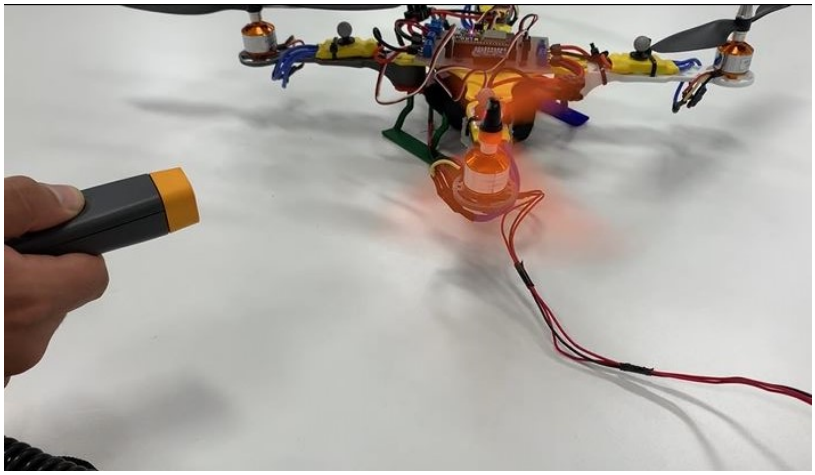


Figura 52: Disposición de tacómetro láser para medición de velocidad angular en RPM.

Motor 1		
Iteración	RPM código	RPM real
1	2500	2791
2	2700	3064.25
3	2900	3114.33
4	3100	3337.66
5	3300	3478.33
6	3500	3669.5
7	3700	3703.25
8	3900	3876.667
9	4100	3984.25
10	4300	4066.5
11	4500	4133.5

Figura 53: Comparación de velocidades angulares experimentales obtenidas contra velocidades angulares solicitadas por controlador de vuelo, motor 1.

máxima de $366.5 \approx 366$ RPM en la iteración 11. Esta variación es equivalente al 8.14 % de error. Sin embargo, una variación de más de 100 RPM en la velocidad de la hélice, la fuerza de empuje generada ya es significativa, por lo que su efecto ya es contraproducente para el vuelo del drone.

El gráfico de la Figura 54 permite observar que existe una variación entre la velocidad solicitada por el controlador de vuelo y la que se obtiene de los motores de forma experimental. La media de la variación es de 199.8 RPM, por lo que su efecto es perceptible a simple vista. En dicho gráfico se observa también que antes de la iteración 7 o 3700 RPM, las velocidades reales son más altas que las solicitadas en código. Sin embargo, después de este punto, las velocidades reales se encuentran por debajo de las solicitadas. Por lo tanto, no se puede componer la "descompensación" simplemente sumando o restando un valor que disminuya la diferencia, pues esta depende de la velocidad a la que se encuentre girando el motor.

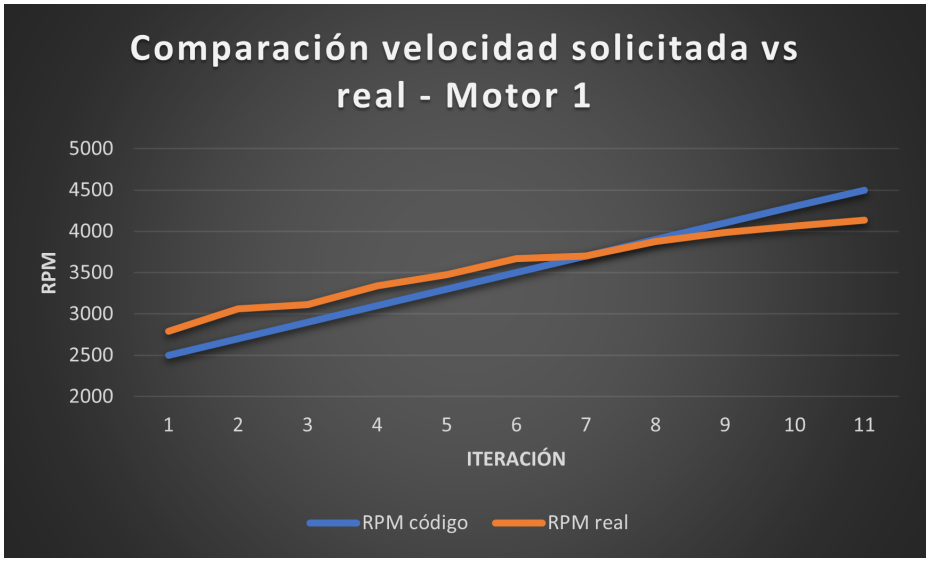


Figura 54: Gráfico de comparación de velocidades angulares experimentales obtenidas contra velocidades angulares solicitadas por controlador de vuelo, motor 1.

Se realizó la misma prueba para el motor 2, cuyas velocidades reales y solicitadas se encuentran en la Figura 55. La diferencia máxima fue de 523.5 RPM, en la iteración 1. Esta variación equivale a 20.94 % de error entre el valor solicitado y el valor obtenido experimentalmente.

En la Figura 56 se observa nuevamente que existe una variación entre la velocidad angular real (línea anaranjada) y el valor solicitado por el controlador (línea azul). A diferencia del motor 1, la varación del motor 2 es bastante más maracada. Esto se comprueba al tener una diferencia promedio de 247.1 RPM durante la prueba. El detalle que sí se comparte de forma similar con el otro motor es que existe un punto en el cual las velocidades menores a ese punto son mayores experimentalmente que las solicitadas. Sin embargo, al sobrepasar ese punto, las velocidades reales son menores que las solicitadas en código. A pesar de que el comportamiento es similar, el punto en donde ocurre no es el mismo que para el motor 1, ya que para este motor el punto se encuentra después de la iteración 8 o 3900 RPM.

Gracias a estos resultados se determinó que la razón por la que el drone no es capaz de estabilizarse en un espacio pequeño es por la descompensación que existe entre los motores, por lo que el controlador de vuelo al alcanzar el punto de equilibrio y enviar la misma velocidad a los cuatro motores percibe nuevamente que el dispositivo se inclina y debe compensar nuevamente. Este ciclo infinito de compensar y descompensar produjo que el cuadricóptero tenga un comportamiento de estabilización temporal. Es decir, el controlador de vuelo estabiliza al drone en diferentes puntos durante el tiempo que esté funcionando.

La variación de las velocidades se atribuyó a factores físicos de la construcción de los motores, los cuales al ser de bajo costo, no tienen una exactitud adecuada para el vuelo del cuadricóptero. Además, el controlador electrónico de velocidad controla la velocidad del motor en lazo abierto, por lo que una vez generada la señal no puede detectar de forma exacta la velocidad de salida del eje. Los motores utilizados fueron modelos provistos por la universidad y por lo tanto, se encontraron las limitantes de su funcionamiento al momento

Motor 2		
Iteración	RPM código	RPM real
1	2500	3023.5
2	2700	3166
3	2900	3242.667
4	3100	3404
5	3300	3524
6	3500	3672.1
7	3700	3851.25
8	3900	3891
9	4100	4086.25
10	4300	4133.667
11	4500	4154

Figura 55: Comparación de velocidades angulares experimentales obtenidas contra velocidades angulares solicitadas por controlador de vuelo, motor 2.

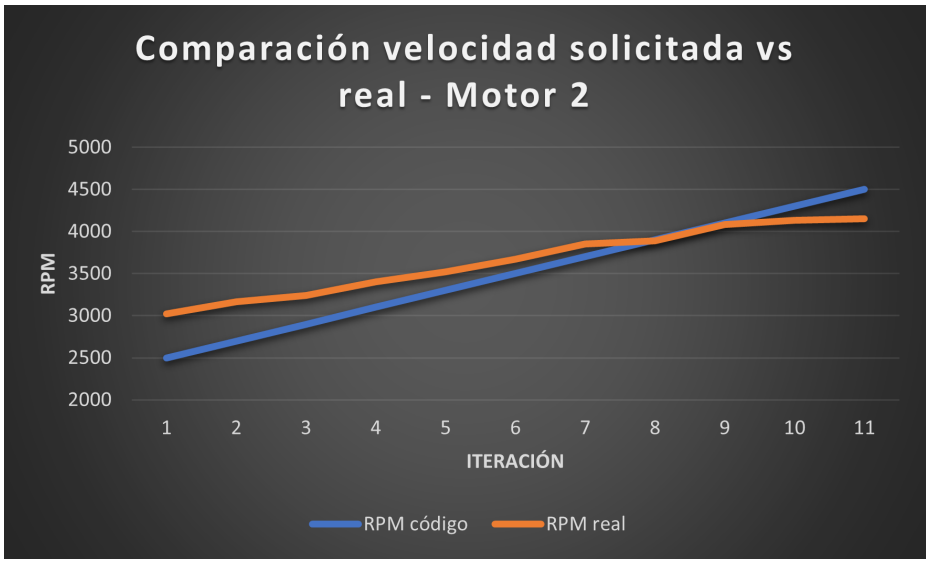


Figura 56: Gráfico de comparación de velocidades angulares experimentales obtenidas contra velocidades angulares solicitadas por controlador de vuelo, motor 2.

de las pruebas.

El diseño del marco estructural fue pensado para admitir la colocación de motores sin escobillas con entradas convencionales, por lo que existe una gran variedad de motores que pueden ser utilizados y conectados con los elementos de sujeción y en su posición sin necesidad de realizar modificaciones al marco.

CAPÍTULO 9

Integración de drone a ecosistema Robotat

9.1. Metodología

El sistema de captura de movimiento OptiTrack permitió obtener información del drone respecto al marco de referencia global. Es decir, la posición (coordenadas X, Y y Z) respecto a un origen determinado y los ángulos de rotación *roll*, *pitch* y *yaw*. Esta información se utilizó para dejar a dispoción del usuario la capacidad de aplicar un control de posición para el dispositivo.

El sistema OptiTrack fue encendido, calibrado y programado de forma independiente al dispositivo trabajado en esta investigación. Para más información referirse al trabajo de graduación de Camilo Perafán [22]. Para realizar la interconexión entre el sistema de captura de movimiento y el drone fueron necesarias dos etapas. La primera, consistió en la instalación de al menos tres (para mayor precisión se utilizaron 4) marcadores reflectivos en el marco estructural del cuadricóptero, los cuales forman un polígono que fue rastreado por las cámaras de alta velocidad. La segunda etapa consistió en la utilización de la librería "`Robotat.c`" en el archivo principal "`main.c`", el cual se implementó en el ESP32. Utilizando comandos integrados en la librería se llevó a cabo la conexión vía Wifi y la solicitud de datos por medio del protocolo MQTT.

Debido a que el controlador desarrollado en este trabajo tiene como objetivo únicamente el control de orientación o *hovering* y con el propósito de verificar la conexión e interacción del drone con el sistema de captura de movimiento se utilizó envió un dato de ángulo $yaw_{deseado}$ desde el sistema OptiTrack. El valor leído por el ESP32 modificó la referencia que se desea alcanzar por el drone. Al modificarse este valor en tiempo real y observar el cambio de orientación del drone con la dirección y desplazamiento angular deseado se verificó el funcionamiento del controlador como de la integración del dispositivo al Robotat. Se utilizó el *software Tracker* para el seguimiento de la trayectoria del drone y analizar el

desplazamiento angular obtenido con el controlador. Se validó tanto el controlador de vuelo como la integración al ecosistema Robotat utilizando como parámetro el porcentaje de error entre el desplazamiento angular deseado y el experimental y que este se encuentre debajo del 10 % de error.

9.2. Instalación de reflectores y lectura de datos desde interfaz *OptiTrack*

La primera etapa para la interconexión entre el sistema de captura de movimiento y el drone a través del ESP32 consistió en la instalación de los marcadores reflectores sobre los brazos del cuadricóptero para que al momento de ser introducido en la plataforma del Robotat, las 6 cámaras de alta velocidad puedan detectar los marcadores y a través de la interfaz del OptiTrack generar un objeto como conjunto.

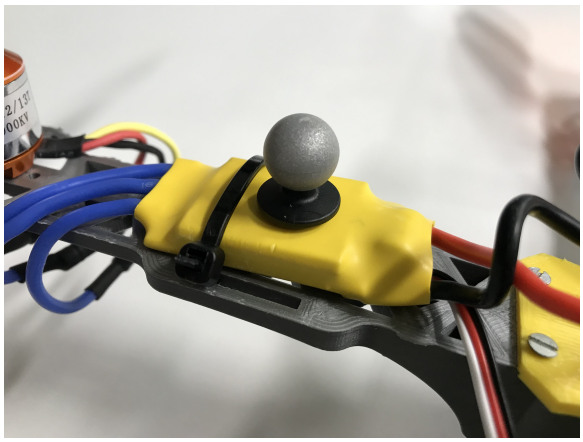


Figura 57: Marcador reflectivo colocado sobre controlador electrónico de velocidad.

En la Figura 57 se demuestra que la posición elegida para una lectura sencilla y sin obstáculos del marcador es sobre el controlador electrónico de velocidad. Esta ubicación permitió que los cuatro marcadores utilizados estén a la misma distancia (los ESCs están en la misma posición) y que sobresalgan de los demás componentes del drone para evitar un bloqueo parcial o completo de la línea de visión de las cámaras. Se verificó que las aspas de las hélices girando no bloqueasen la visualización de los marcadores, Figura 58, y el resultado en esta posición fue exitoso.

En la Figura 59 se observa la forma en que las cámaras detectan los marcadores reflectivos. Como se refleja en dicha imagen, la calidad de la imagen del drone no es alta, sin embargo, lo que es importante para que el sistema funcione correctamente es que los puntos blancos sí estén claros y destaquen del resto, tal como lo es para este drone. En la interfaz del OptiTrack se indica que la tasa de refresco de las cámaras es de 120 Hz con un tiempo de exposición de 250 μs.

A pesar de que se puede visualizar el drone de forma realista, la manera más importante en la que el sistema de captura de movimiento provee información es al crear un objeto con los marcadores detectados y seleccionados, Figura 60.

Figura 58: Marcador reflectivo colocado sobre brazo del drone.

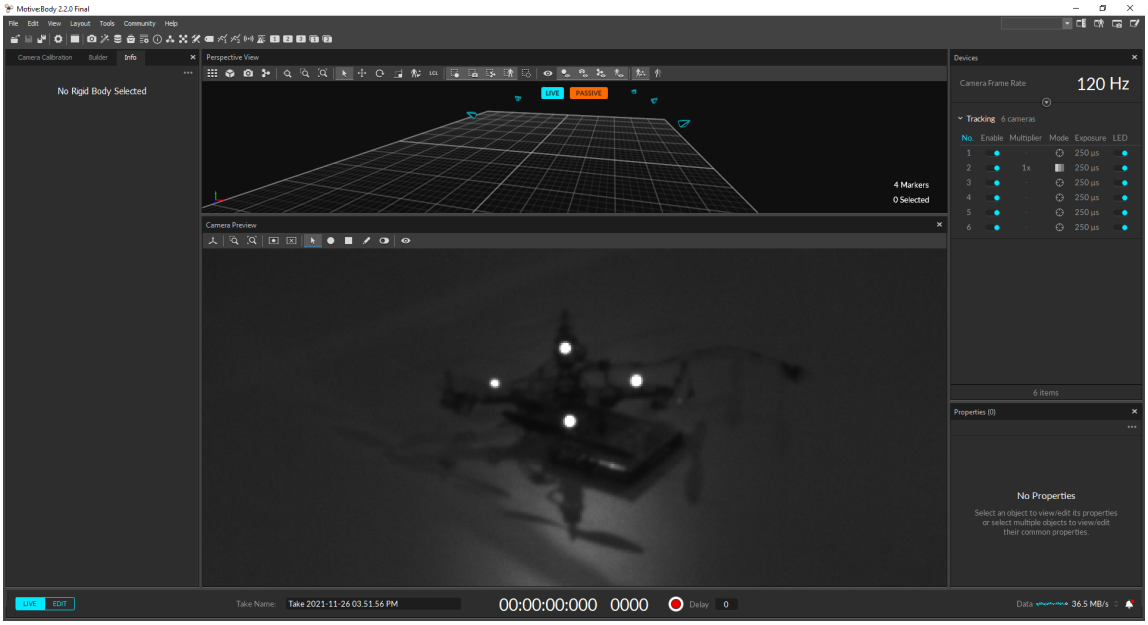


Figura 59: Visualización de marcadores del drone realista desde el OptiTrack.

En la Figura 61 se presentan los 4 marcadores seleccionados y generados como un cuerpo rígido. De esta forma, el programa automáticamente genera un centro geométrico, el cual aparece como un punto más grande que los marcadores y es donde se coloca el marco de referencia de la base. Se destaca que para la creación de un objeto o polígono en el programa, son necesarios únicamente 3 marcadores. Sin embargo, al tener 4 se pudo mantener la forma simplificada del drone y lograr que el centro geométrico del polígono en el OptiTrack coincida con el centro del drone, lo cual permite que los ejes de rotación, y por tanto *roll*, *pitch* y *yaw* coincidan. En la misma Figura 61 se observa el origen del marco de referencia global indicado como una "x"sobre el suelo de la interfaz. Este origen se indica durante la calibración del sistema.

En la Figura 62, en la pestaña ”Info” (lado izquierdo) se despliega la información que se utilizó para el cuadricóptero. Se encuentra la posición en milímetros por cada uno de

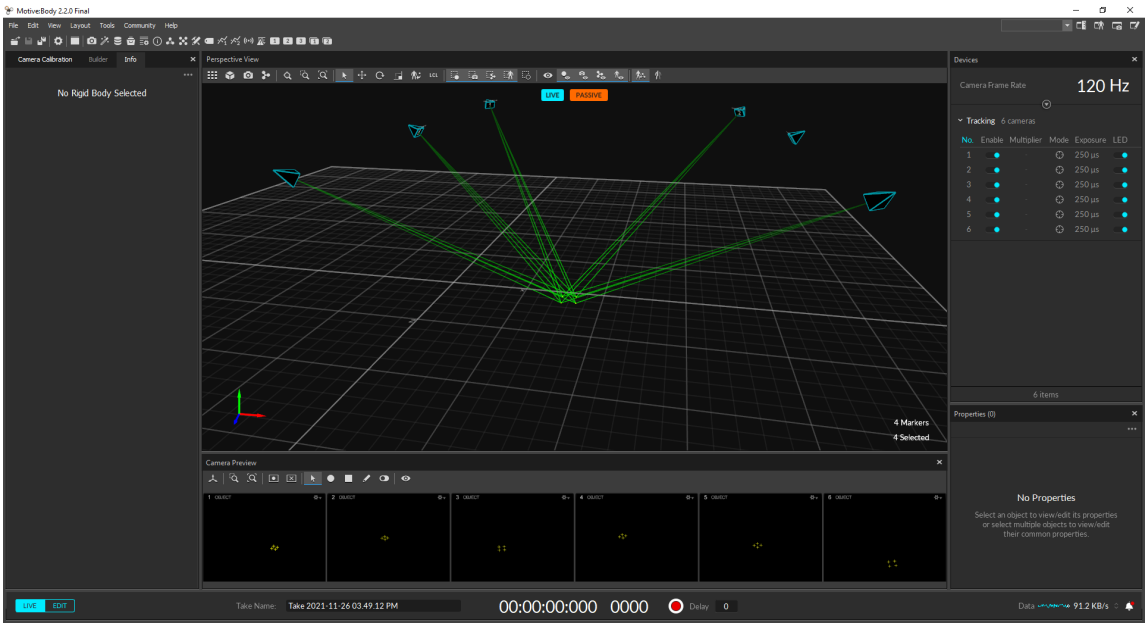


Figura 60: Señales de las cámaras para detección de marcadores reflectivos.

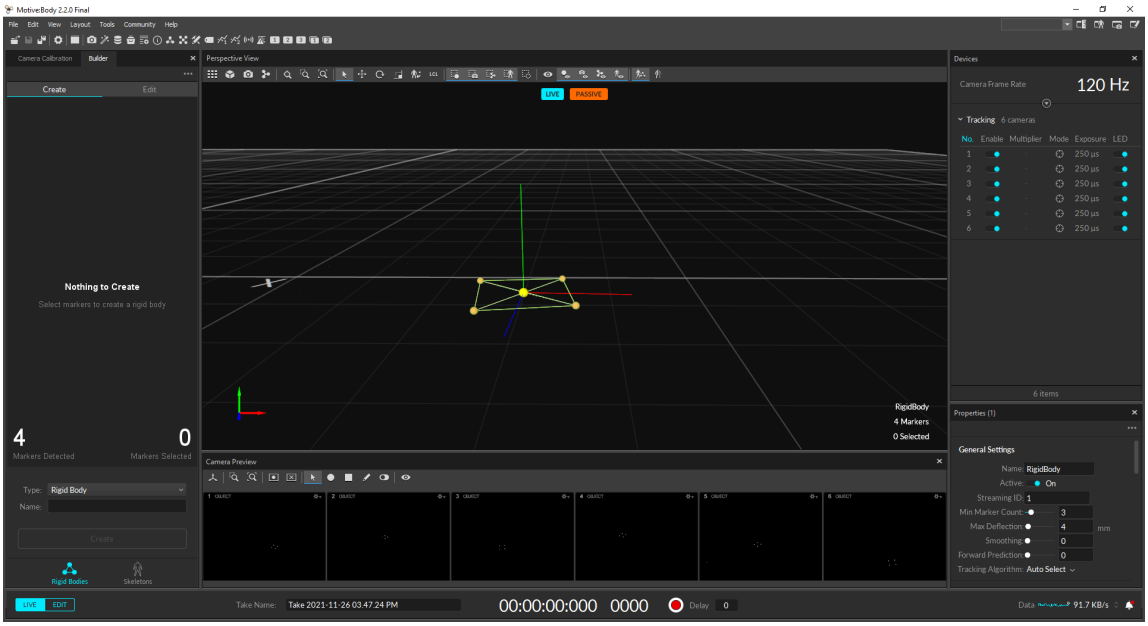


Figura 61: Visualización de marcadores del drone desde el OptiTrack como objeto.

los ejes coordenados y la orientación en grados para *roll*, *pitch* y *yaw*. También se marca la tolerancia del sistema, la cual se indió de error medio 0.096 mm/marcador.

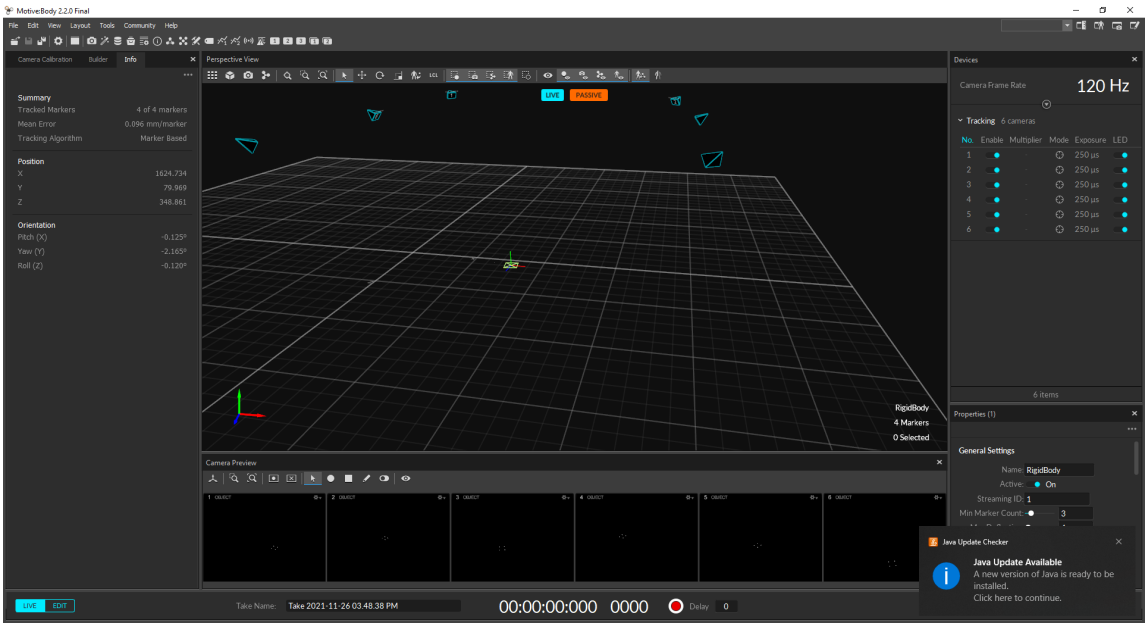


Figura 62: Obtención de información del drone desde interfaz del OptiTrack.

9.3. Solicitud de datos a través de MQTT desde ESP32

En la Figura 63 se despliega la forma en la que se trabajó con la librería desarrollada por Camilo Perafán para realizar la conexión del ESP32 por medio de Wifi al sistema de captura de movimiento OptiTrack.

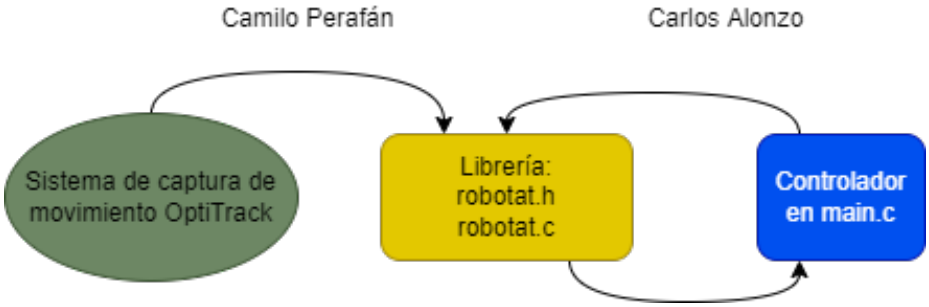


Figura 63: Interacción con libería desarrollada por Camilo Perafán para obtención de datos del OptiTrack.

Las funciones utilizadas para la conexión y solicitud de datos a través del protocolo MQTT se presentan en el Cuadro 7.

Función	Parámetros	Descripción
`robotat_connect`	Ninguno	Esta función realiza las configuraciones necesarias para preparar y conectar el ESP32 a la red de comunicación por medio de Wifi y el protocolo MQTT. Para más información referirse al trabajo de graduación de Camilo Perafán [22].
`robotat_get_status`	Ninguno	Al llamar esta función y almacenar su resultado en un variable se obtienen los siguientes posibles valores de salida: 1. MQTT conectado 2. MQTT desconectado 3. MQTT suscrito a un tópico 4. MQTT desuscrito a un tópico 5. Mensaje publicado 6. Mensaje recibido 7. Error
`robotat_get_data`	Ninguno	Función que permite obtener el paquete de datos enviado a través del tópico al que se está suscrito. El paquete de datos se trabaja como un conjunto de información para la cantidad de objetos suscritos, por lo que es necesario separar o seleccionar únicamente los que se desean utilizando los índices dentro del *array* de datos.

Cuadro 7: Funciones utilizadas para conexión del ESP32 con sistema de captura de movimiento OptiTrack.

Tal como se mencionó previamente, estas funciones pertenecen a la librería desarrollada para el ecosistema Robotat completo, por lo que fue necesario hacer una pequeña modificación para adaptarla al drone. En el archivo "`robotat.c`" se comentó la línea de código 156, desplegada en el Listado 9.1.

```
1 //Linea 156:
2
3 //utilizada para seleccionar pin 2 del ESP32 como salida para el LED RGB.
4
5 //gpio_set_level(CONNECTED_LED, 1);
```

Listing 9.1: Línea de código 156 de librería robotat.c

De igual forma se comentaron las líneas de código 247 y 248 del mismo archivo, desplegadas en el Listado 9.2:

```
1 //Lineas 247 y 248 utilizadas para configurar y el pin 2 del ESP32 como ...
      salida para el LED RGB.
2 // gpio_pad_select_gpio(CONNECTED_LED);
3 // gpio_set_direction(CONNECTED_LED, GPIO_MODE_OUTPUT);
```

Listing 9.2: Línea de código 156 de librería robotat.c

Estos cambios son necesarios debido a que el pin 2 utilizado como señalización de que la conexión Wifi fue exitosa para el ESP32 por la librería `robotat.c` utiliza el mismo pin que la salida del PWM para el motor 4 del drone.

Después de recibidos los datos, fue necesario realizar una conversión para los ángulos de rotación. Esto se debe a que el sistema de captura de movimiento entrega los valores como un cuaternión. Por lo tanto, fue necesario realizar la conversión a ángulos de Euler *roll*, *pitch* y *yaw* utilizando la convención XYZ.

Las funciones utilizadas se presentan en el cuadro 8.

Función	Parámetros	Descripción
`OptiTrack GetEuler Angles`	quat0 quat1 quat2 quat3	Función que convierte un cuaternión a ángulos de Euler con convención XYZ. Los parámetros quat seguido por un número representan cada componente del cuaternión.
`norm_angle_0_2pi_OT`	ángulo	Función utilizada para normalizar un ángulo de 0 a 2π. Es llamada dentro de la función `OptiTrackGetEulerAngles`.

Cuadro 8: Funciones utilizadas para conversión de cuaternión a ángulos de Euler.

El código utilizado para las funciones mencionadas previamente se encuentra en los Listados 9.3 y 9.4, respectivamente.

```
1 void OptiTrackGetEulerAngles(float quat0,  float quat1, float quat2, ...
      float quat3){
2   float wwOT = quat0 * quat0; float xxOT = quat1 * quat1;
3   float yyOT = quat2 * quat2; float zzOT = quat3 * quat3;
4   YAW = norm_angle_0_2pi_OT(atan2f(2.0 * (quat1 * quat2 + quat3 * quat0), ...
        xxOT - yyOT - zzOT + wwOT));
5   PITCH = asinf(-2.0 * (quat1 * quat3 - quat2 * quat0));
6   ROLL = atan2(2.0 * (quat2 * quat3 + quat1 * quat0), -xxOT - yyOT + zzOT ...
      + wwOT);}
```

Listing 9.3: Función para conversión de cuaternión a ángulos de Euler

```
float norm_angle_0_2pi_OT(float a){
  a = fmod(a, M_PI * 2.0);
  if (a < 0){
    a += M_PI * 2.0;}
  return a;}
```

Listing 9.4: Función para normalizar un ángulo de 0 a 2π

Utilizando las funciones previas se presenta la sección de código del archivo `main.c` donde se seccionó únicamente la información deseada del paquete de datos recibido y su asignación a las variables que utiliza el controlador. Esta sección de código se presenta en el Listado 9.5.

```
        //Obtencion de angulos OptiTrack
        estado_wifi = robotat_get_status();
        if(estado_wifi == 8){
          data = robotat_get_data();
          quat0 = data[4];
          quat1 = data[5];
          quat2 = data[6];
          quat3 = data[7];
          yaw_OT_deseado = data[8];
          yaw_OT_deseado = yaw_OT_deseado*M_PI/180.0;
          OptiTrackGetEulerAngles(quat0, quat1, quat2, quat3);
          ROLL = ROLL*(180.0/M_PI);
          PITCH = PITCH*(180.0/M_PI);
          YAW = YAW*(180.0/M_PI);
          printf("R_OT: %f \n P_OT: %f \n Y_OT: %f\n Yaw_des: %f\n \n", ...
              ROLL, PITCH, YAW, yaw_OT_deseado);
        }
```

Listing 9.5: Sección de código donde se separan los datos obtenidos del paquete recibido

Para modificar el valor de *yaw* desde el sistema de captura de movimiento se utilizó un *script* trabajado por Camilo Perafán [22] en Python donde se modifica el contenido del paquete de datos y se actualiza el valor deseado para este ángulo. En la Figura 64 se presenta la línea de código en donde se modifica el valor de la variable `yaw_direction`.

9.4. Resultados de recepción de datos

Se utilizó la consola de Spyder (Anaconda3) para la impresión de datos de la red de comunicación obteniendo el paquete de datos y el estado actual del estado de conexión entre el ESP32 y el OptiTrack.

La Figura 65 muestra que el paquete de datos despliega un ”0” en la segunda posición, lo cual indica que el ESP32 aún no está conecatado a la red de comunicación a través del protocolo MQTT ("0"desconectado, ”1” conectado). La información se envía constantemente desde el OptiTrack, sin importar si hay un dispositivo conectado y recibiéndola. Al tener encendido el sistema de comunicación y paquete de datos se procedió a encender el ESP32 del drone y esperar a que se conecte de forma automática.

```
#New data for drone
yaw_direction = 0
yaw_direction_str = str(yaw_direction)

# Payload to send via MQTT
data_str = str_len + " " + id_use + " " + id_str + " " + posx_str + \
           " " + posy_str + " " + posz_str + " " + eta_str + " " + \
          eps1_str + " " + eps2_str + " " + eps3_str + " " + \
          yaw_direction_str + " " +  \
          traj_points_str + " " + mode_str + " " + speed_str + " " + \
          run_mode_str + " " + grip_stat_str + " " + conf_str + " " + \
          end_effector_str + " " + trj_pos_table_str + " " + "0"
```

Figura 64: Modificación de paquete de datos para actualizar el ángulo deseado de *yaw*.

```
Received data:
47 0 1 0.8548 0.0796 1.1068 0.992 -0.0014 0.126 0.0026 0 7 1 10000 1 0 0.59 -59 50 -27 0 0 0 0 450 0 0 750 0 0 550
100 200 550 0 450 400 -100 100 350 0 550 300 -100 100 350 0
Received data:
47 0 1 0.8547 0.0796 1.1069 0.992 -0.0016 0.1258 0.0025 0 7 1 10000 1 0 0.59 -59 50 -27 0 0 0 0 450 0 0 750 0 0 550
100 200 550 0 450 400 -100 100 350 0 550 300 -100 100 350 0
Received data:
47 0 1 0.8548 0.0796 1.1068 0.992 -0.0014 0.126 0.0027 0 7 1 10000 1 0 0.59 -59 50 -27 0 0 0 0 450 0 0 750 0 0 550
100 200 550 0 450 400 -100 100 350 0 550 300 -100 100 350 0
```

Figura 65: Impresión en consola de estado desconectado entre ESP32 y OptiTrack.

```
--------------------------------
Topic: Used_IDs
Message: b'1 ESP32_Drone'
QOS: 2
{'1': 'ESP32_Drone'}
47 1 1 0.8548 0.0796 1.1068 0.992 -0.0014 0.1262 0.0026 0 7 1 10000 1 0 0.59 -59 50 -27 0 0 0 0 450 0 0 750 0 0 550
100 200 550 0 450 400 -100 100 350 0 550 300 -100 100 350 0
Received data:
47 1 1 0.8548 0.0796 1.1068 0.992 -0.0014 0.126 0.0026 0 7 1 10000 1 0 0.59 -59 50 -27 0 0 0 0 450 0 0 750 0 0 550
100 200 550 0 450 400 -100 100 350 0 550 300 -100 100 350 0
Received data:
47 1 1 0.8547 0.0796 1.1068 0.992 -0.0016 0.1259 0.0025 0 7 1 10000 1 0 0.59 -59 50 -27 0 0 0 0 450 0 0 750 0 0 550
100 200 550 0 450 400 -100 100 350 0 550 300 -100 100 350 0
```

Figura 66: Impresión en consola de estado conectado entre ESP32 y OptiTrack.

Una vez que se realiza exitosamente la conexión entre los dispositivos la consola muestra el mensaje desplegado en la Figura 66. Después del mensaje de conexión al tópico, el valor de la segunda posición del paquete de datos cambia a "1", lo cual indicó que la conexión fue exitosa.

El controlador de *hovering* que contiene el ESP32 del drone modifica las velocidades de los motores para alcanzar los valores de *roll*, *pitch* y *yaw* deseados. Se modificó el valor de velocidad en estado estable ”wh” a 3000 RPM, lo cual es una velocidad suficientemente alta para que el drone gire alrededor de su eje vertical, pero no lo suficientemente alta para que se levante completamente del suelo. Esta forma de vuelo permitió que se comprobara que el controlador es capaz de alcanzar el valor de *yaw* deseado y que este valor al ser modificado en tiempo real también cambiase la orientación del drone. Se utilizó el programa de análisis de video Tracker [19] para el seguimiento del drone y el cambio de orientación que se obtuvo. El primer paso fue el establecimiento de ejes coordenados que coincidieran con los brazos del drone. Se esperó a que el drone alcanzara su estado de equilibrio (*yaw* deseado igual

a cero) y en ese punto se estableció como el estado inicial del drone. Se escogió el Motor1 como motor a analizar debido a que es el más sencillo de observar tanto a simple vista en el video como para ser detectado automáticamente por Tracker. La Figura 67 muestra la disposición seleccionada para el punto inicial de la prueba. El transportador (círculo de color verde) muestra el Motor1 (hélice color anaranjado) como el cero del transportador. También se comprueba que la separación entre los brazos es de 90°.

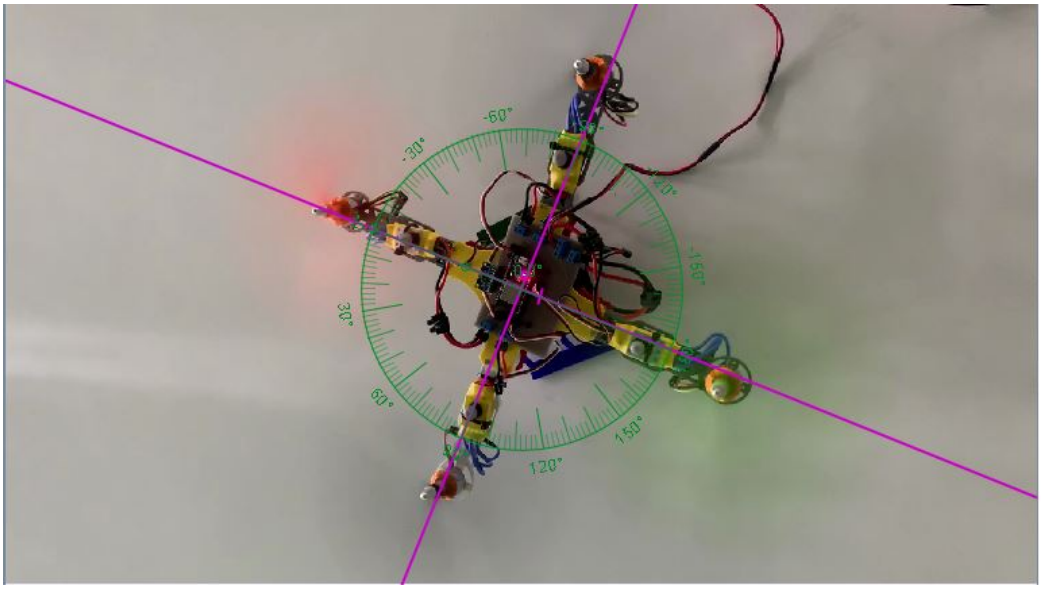


Figura 67: Estado inicial del drone y colocación de ejes coordenados en *Tracker*.

Debido a que la conexión entre el ESP32 del drone y el OptiTrack se encuentraba en línea se realizó la modificación al *script* de Python para enviar el nuevo valor deseado de *yaw*, el cual al principio es de "0" para no interferir con el deseado por el controlador en estado de equilibrio. Como se muestra en la Figura 68, en la línea 129 del código se modifica la variable "yaw_direction" de 0 a 90.

```
128     #New data for drone
129     yaw_direction = 90
130     yaw_direction_str = str(yaw_direction)
131
132     # Payload to send via MQTT
133     data_str = str_len + " " + id_use + " " + id_str + " " + posx_str + \
134                " " + posy_str + " " + posz_str + " " + eta_str + " " + \
135                eps1_str + " " + eps2_str + " " + eps3_str + " " + \
136                yaw_direction_str + " " +  \
137                traj_points_str + " " + mode_str + " " + speed_str + " " + \
138                run_mode_str + " " + grip_stat_str + " " + conf_str + " " + \
139                end_effector_str + " " + trj_pos_table_str + " " + "0"
```

Figura 68: Modificación del *yaw* deseado desde *script* de Python.

Se utilizó la funcionalidad de *Tracker* para seguimiento automático de una partícula dentro del vídeo. Se remarca que mientras más destaque la característica dentro del vídeo, más sencillo será para el programa seguirla durante el tiempo. La Figura 69 muestra la trayectoria que siguió el Motor1 durante los 5 segundos que duró el análisis del vídeo. Dentro

de este intervalo de tiempo el drone alcanzó el nuevo ángulo *yaw*_deseado. El sentido de giro concuerda con el valor positivo de *yaw* respecto al eje vertical. Utilizando la regla de la mano derecha se determina que el giro positivo de 90° debe ser en contra de las agujas del reloj, tal como se obtuvo en la Figura 69. A simple vista se observó que el drone realizó un giro de 90° en *yaw*, pero para saber qué tan cerca se llegó se utilizaron los datos provistos por *Tracker*.

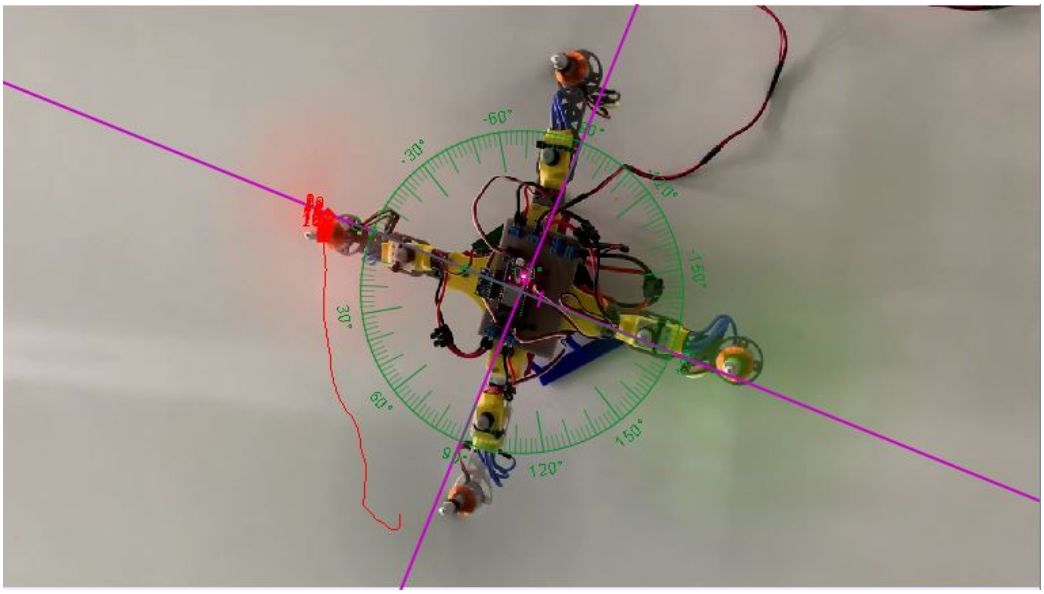


Figura 69: Trayectoria del Motor 1 analizado desde *Tracker* utilizando el controlador de vuelo para nuevo *yaw* deseado.

La Figura 70 muestra un gráfico del ángulo del motor 1 respecto a los ejes coordenados establecidos en un inicio en términos del tiempo. El tiempo de muestreo utilizado fue de 0.033s. Se observó que el comportamiento es bastante lineal y las variaciones fueron provocadas por el giro del drone el cual por un instante se movió levemente del origen en el que se encontraba, sin embargo el controlador lo colocó nuevamente en su posición original. El eje vertical muestra los ángulos absolutos respecto el eje x positivo (cuadrante I) de los ejes coordenados, por lo que el valor inicial es 179.6° en lugar de 0°, Figura 71. El valor final del ángulo *yaw* es de 264.4° en el tiempo 4.805s, Figura 72.

$$desp_{angular} = 264.4° - 179.6° = 84.8° \tag{25}$$

El desplazamiento angular experimenal fue de 84.8°, obtenido en la ecuación 25.

Desplazamiento angular deseado	Desplazamiento angular experimental	% de error
90°	84.8°	5.78 %.

Cuadro 9: Porcentaje de error entre desplazamiento angular deseado y desplazamiento angular experimental utilizando controlador de vuelo.

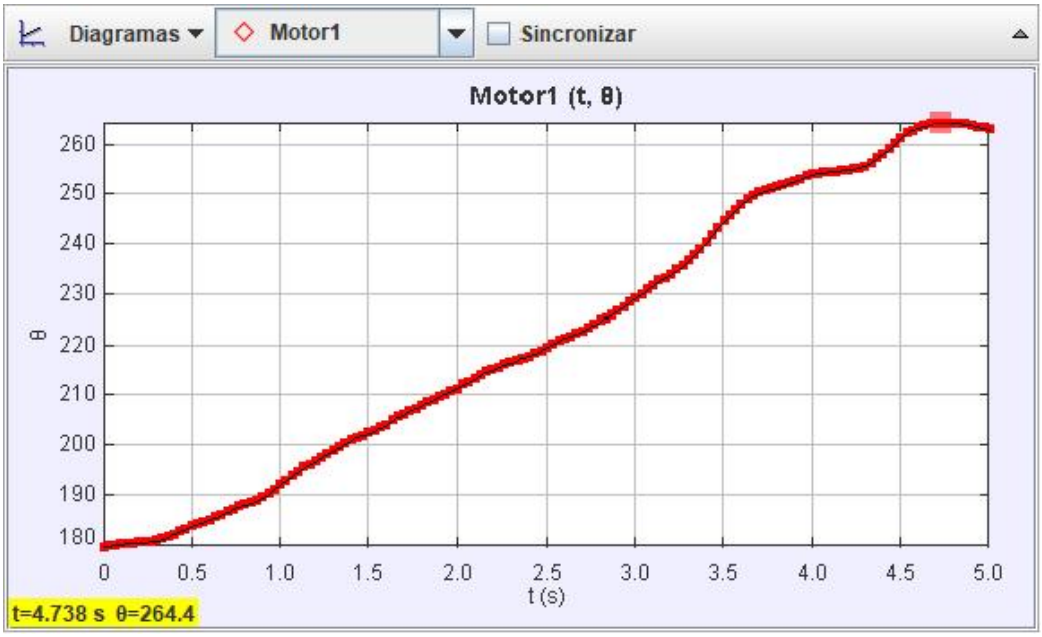


Figura 70: Comportamiento del ángulo *yaw* utilizando sistema de captura de movimiento para prueba de controlador.

t (s)	θ
0.000	179.6°
0.033	179.7°
0.067	179.9°
0.100	180.0°
0.133	180.2°

Figura 71: Primeros 5 datos de prueba para *yaw* deseado.

4.705	264.3°
4.738	264.4°
4.772	264.4°
4.805	264.4°
4.838	264.3°
4.872	264.1°
4.905	263.9°
4.938	263.7°
4.972	263.4°
5.005	263.1°

Figura 72: Últimos 10 datos de prueba para *yaw* deseado.

La prueba del controlador de vuelo modificando únicamente el valor de *yaw* deseado resultó exitosa con un porcentaje de error de 5.78 % como se muestra en el Cuadro 9. El porcentaje de error se adjudicó a dos factores principales. El primero, factores físicos como la descompensación de las velocidades de los motores, mostrada en el capítulo anterior, la cual no afectó de forma significativa en esta prueba debido a que el drone se encontraba a nivel del suelo. El segundo factor de error fue el análisis de vídeo ya que a pesar de contar con una gran herramienta de *software* como *Tracker*, el seguimiento de los píxeles puede

variar ligeramente. Además, el drone al no estar anclado sobre un eje físico, los pequeños desplazamientos en este eje virtual provocan variaciones en las lecturas del ángulo *yaw*.

El resultado de esta prueba permitió validar que el controlador de vuelo permite alcanzar un valor específico de *yaw* con un porcentaje de error significativamente bajo (5.78 % para este caso), lo cual comprueba que el controlador presenta el comportamiento deseado. Ademáss, se validó que el cuadricóptero se integró al ecosistema Robotat y sistema de captura de movimiento Robotat por medio de comunicación Wifi utilizando el protocolo MQTT, y que se pueden realizar modificaciones en tiempo real para que el controlador del drone las ejecute.

CAPÍTULO 10

Conclusiones

- Se diseñó un marco estructural capaz de sostener los componentes del drone con una deformación de 0.543 % del total de su longitud, lo cual está debajo del límite superior establecido del 1 %.
- Se diseñó un controlador proporcional-derivativo con un acercamiento combinado entre control clásico y moderno, control LTI en espacio de estados, con el que se utilizó el modelo dinámico del drone, sensores a bordo y motores sin escobillas para mantener el dispositivo en estado de *hovering*. Las modificaciones de las condiciones del estado deseado permitieron observar y validar el comportamiento del controlador de vuelo.
- Se desarrolló un entorno de simulación para un cuadricóptero utilizando la herramienta de Matlab, el cual permite la programación, configuración y evaluación de un controlador de vuelo previo a su implementación en el dispositivo físico. El entorno está limitado a simular el control como solución a la ecuación de movimiento del drone, por lo que no toma en cuenta la conversión de fuerzas y momentos en velocidades de rotación para los motores sin escobillas.
- La integración del drone con el OptiTrack permitió el envío y recepeción de información a través de wifi y el protocolo MQTT por lo que se pudo modificar la orientación del cuadricóptero en tiempo real con un 94.22 % de exactitud cambiando los parámetros deseados de forma inalámbrica.
- La descompensación de los motores, alcanzando hasta un 20.78 % de error, dependiente de su elementos físicos fue la provocante de la dificultad del cuadricóptero para estabilizarse en espacios pequeños.

CAPÍTULO 11

Recomendaciones

1. Realizar una comparación de deformación entre PLA y *nylon* relleno con micro fibra de carbono (*Onyx*), para determinar si disminuye el peso y deformación del drone lo suficiente para contrarrestar el aumento del costo económico.

2. Utilizar un promedio de datos o suavizador de curvas para las velocidades de los motores obtendias en tiempo real para que las gráficas resultantes sean curvas más suaves y el comportamiento del controlador de vuelo se observe de forma más clara.

3. Verificar y utilizar motores sin escobillas que tengan una velocidad de salida con una variación o porcentaje de error bajo respecto a la velocidad solicitada por el controlador electrónico de velocidad. Esto permitirá que el controlador de vuelo tenga una ejecucción más eficiente y el cuadricóptero se estabilice en espacios más pequeños.

4. Asegurar que el marco estructural del drone no se mantenga o almacene en espacios calientes. Debido al material plástico y el peso de los motores, los brazos tendrán una deformación por el calor que provocará que los motores no se apunten completamente verticales.

5. Las patas diseñadas para el despegue del drone no forman parte de este trabajo, por lo que se recomienda diseñarlas dependiendo del tipo de trabajo para el que se desee emplear el dispositivo. Tomar en cuenta que deben resistir una gran cantidad de impactos durante las pruebas de vuelo.

6. Utilizar un valor de constante proporcional de *yaw* alto, alrededor de 120,000 y 150,000. Y, utilizar un valor de constante derivativa de al menos el doble de la proporcional.

7. Para mayor comodidad de programación, implementar la programación *Over-The-Air* que permite el ESP32. Esto evitará la necesidad de conectar el cuadricóptero a una computadora cada vez que se desee hacer una modificación al controlador.

CAPÍTULO 12

Bibliografía

[1] Goldman Sachs, *Goldman Sachs Research*, https://www.goldmansachs.com/insights/technology-driving-innovation/drones/, Accessed: 2021-03-20, 2021.

[2] M. Búrbano, J. Morales, M. Sandoval, R. Saravia y D. Suazo, "Megaproyecto SEEQ Robohelicóptero," Tesis de mtría., Universidad del Valle de Guatemala, Guatemala, GTM, nov. de 2011.

[3] J. Castañeda, "Diseño e implementación de una red de comunicación wifi e interfaz gráfica para una mesa de pruebas de robótica de enjambre," Tesis de mtría., Universidad del Valle de Guatemala, Guatemala, GTM, oct. de 2020.

[4] G. García, "Diseño de Estructura en Configuración Y4 para Dron," Tesis de mtría., Universidad del Valle de Guatemala, Guatemala, GTM, dic. de 2019.

[5] D. W. Mellinger, "Trajectory Generation and Control for Quadrotors," Tesis doct., University of Pennsylvania, Pennsylvania, PA, 2012.

[6] R. Budynas y J. Nisbett, *Diseño en ingeniería mecánica de Shigley (9a. ed.)* 2012, ISBN: 9781456245238. dirección: https://books.google.com.gt/books?id=knWJDAAAQBAJ.

[7] L. Ríos y E. Roncancio, "Análisis y desarrollo de un programa de selección rápida de factores de seguridad, para diseño de elementos mecánicos," *Scientia Et Technica*, vol. XIII, n.º 35, págs. 255-260, 2007.

[8] E. Rayón y M. Arrieta, "Metodología docente para explicar el concepto de Resistencia de Materiales. Estudio del Apple Watch," Universitat Politècnica de València, Valencia, ESP, inf. téc. s/n 03801, 2015.

[9] B. Szabó e I. Babuška, *Finite element analysis*. John Wiley & Sons, 1991.

[10] I. Domínguez, L. Romero, M. Espinosa y M. Domínguez, "Impresión 3D de maquetas y prototipos en arquitectura y construcción," es, *Revista de la construcción*, vol. 12, págs. 39-53, nov. de 2013, ISSN: 0718-915X. dirección: http://www.scielo.cl/scielo.php?script=sci_arttext&pid=S0718-915X2013000200004&nrm=iso.

[11] P. Corke, *Robotics, vision and control: fundamental algorithms in MATLAB® second, completely revised*. Springer, 2017, vol. 118.

[12] B. Siciliano, L. Sciavicco, L. Villani y G. Oriolo, *Robotics: Modelling, Planning and Control*, 1st. Springer Publishing Company, Incorporated, 2008, ISBN: 1846286417.

[13] K. Ogata, *Ingeniería de control moderna*. Pearson Educación, 2003, ISBN: 9788420536781. dirección: https://books.google.com.gt/books?id=QK148EPC%5C_m0C.

[14] ESPRESSIF, *ESP32 Series of Modules*, https://www.espressif.com/en/products/modules/esp32, Accessed: 2021-05-09, 2021.

[15] La Electrónica, *MÓDULO WI-FI + BLUETOOTH ESP32*, https://laelectronica.com.gt/modulo-wi-fi--bluetooth-esp32, Accessed: 2021-05-09, 2021.

[16] OptiTrack, *OptiTrack - Motion Capture for Robotics*, https://optitrack.com/applications/robotics/, Accessed: 2021-05-14, 2021.

[17] Digey-Key Electronics, *A2212 Brushless Motor*, https://components101.com/motors/2212-brushless-motor, Accessed: 2021-08-28, 2018.

[18] S. Yadav, M. Sharma y A. Borad, "Thrust Efficiency of Drones (Quad Copter) with Different Propellers and there Payload Capacity," *International Journal of Aerospace and Mechanical Engineering*, vol. 4, n.º 2, págs. 18-23, 2017.

[19] Tracker, *Tracker Video Analysis and Mdeling Tool*, https://physlets.org/tracker/, Accessed: 2021-08-10, 2021.

[20] University of Pennsylvania, *Robotics: Aerial Robotics*, https://www.coursera.org/learn/robotics-flight/home/info, Accessed: 2020-12-22, 2016.

[21] MathWorks, *MathWorks ode45*, https://www.mathworks.com/help/matlab/ref/ode45.html, Accessed: 2021-08-20, 2021.

[22] C. P. Montoya, "Robotat: un ecosistema robótico de captura de movimiento y comunicación inalámbrica," Tesis de mtría., Universidad del Valle de Guatemala, Guatemala, GTM, 2021.

[23] Renesas Electronics Corporation, *What are Brushless DC Motors*, https://www.renesas.com/us/en/support/engineer-school/brushless-dc-motor-01-overview, Accessed: 2021-09-30, 2021.

CAPÍTULO 13

Anexos

13.1. Repositorio de Github y Drive

13.1.1. Planos de construcción del drone

En el siguiente enlace se encuentran los planos para la manufactura del drone por medio de impresión 3D.

https://drive.google.com/file/d/10IFMl5_AlkvUf-uVIS1fDu96TJ-5uQ8O/view?usp=sharing

13.1.2. Archivos de controlador simulado

En el siguiente enlace se encuentran los archivos principales y secundarios utilizados para ejecutar entorno de simulación en Matlab.

https://drive.google.com/drive/folders/1NDdLrZ1aTkS9YlUpyzSm2tGWb9Sgpc_x?usp=sharing

13.1.3. Archivos de controlador en ESP32

En el siguiente enlace se encuentran los archivos que se cargaron al microcontrolador ESP32 para el controlador de vuelo.

https://github.com/CarlosAlonzoUVG/Controlador_Final.git

CAPÍTULO 14

Glosario

Brushless: Es un motor síncrono de corriente directa, también conocido como motor electrónicamente conmutado [23]. 29