Planeación de trayectorias para un robot aéreo AR.Drone 2.0 usando GPS

Daniel Ruiz Restrepo
Pedro Felipe Rizo González

Departamento de Electrónica, facultad de ingeniería
Pontificia Universidad Javeriana

2014

Agradecimientos

Daniel

Le agradezco a mis padres Bernardo y Esperanza por darme la oportunidad de estudiar en tan prestigiosa universidad, por siempre brindarme lo que necesito y asegurarse que nunca me falte nada. A mis hermanas María Angélica y Alejandra por quererme y apoyarme en los momentos de dificultad. Le agradezco a mi novia María Antonia por entregarme su confianza y amor a lo largo de estos años. A nuestro director de tesis Julián Colorado por su colaboración con el proyecto. Y finalmente a mis amigos, especialmente al mejor de ellos, mi compañero de tesis Pedro.

Pedro

Quiero agradecer primero a mis padres, que desde el primer día de este viaje estuvieron siempre acompañándome, apoyándome y brindándome una voz de sabiduría siempre que fue necesario. Les doy gracias por su paciencia y su amor incondicional. A mis hermanos, por ser mi referencia y grandes modelos a seguir, porque con sus propios éxitos me han enseñado el valor del sacrificio y del trabajo. Gracias, porque todos los días me esfuerzo para ser un poco más como ustedes. A mi mejor amigo y compañero Daniel, porque desde el primer minuto recorrimos este complicado camino juntos, ayudándonos en cada paso dado. A Julián Colorado, por brindarnos la oportunidad de realizar este proyecto y confiar en nosotros, ayudándonos siempre. A Andrea, porque su paciencia, cariño y amor, han resultado el pilar del que me he apoyado en los momentos más difíciles, sin ti, me hubiera vuelto loco. A Dios, por esa ayudita extra que nunca sobró y por enseñarme los valores y principios por los que me he regido toda mi vida.

Contenido

Lista de figuras

Lista de tablas

1. INTRODUCCIÓN

Los *quadrotors* son helicópteros de cuatro rotores que pueden realizar maniobras de una manera muy ágil y veloz, debido a su pequeño peso, tamaño y ventajas en términos de dinámica. Esto hace que puedan ingresar en lugares donde los humanos difícilmente podrían, ya sea por restricciones físicas o en términos de seguridad. Estos dispositivos se conocen también como UAV (Vehículo aéreo no tripulado) o MAV (Micro Air Vehicle) y presentan una gran cantidad de aplicaciones en el mundo actual. Cada una de estas aplicaciones requiere trabajo constante por parte de los equipos de desarrollo donde se implementan herramientas que hacen que el robot pueda ejecutar diferentes tareas. Sin embargo, muchas de estas aplicaciones y herramientas aún requieren control humano y no funcionan de manera autónoma. Por esto, se vuelve de vital importancia desarrollar una estrategia que permita que estos UAV's realicen todo tipo de tareas sin necesidad de intervención humana, permitiendo planear rutas con anterioridad, o irlas planeando a medida que van ejecutando las diferentes acciones de vuelo. Los métodos para planeación de trayectorias más conocidos y difundidos pasan por la navegación visual, lo que le ofrece al vehículo un alto conocimiento de su entorno. No obstante, muchas veces son ineficientes y presentan limitaciones, por ejemplo, permiten planear una trayectoria de manera anticipada. En este punto se centra este trabajo de grado.

Haciendo uso del sistema de posicionamiento más utilizado y difundido a nivel mundial, como lo es el GPS, se desarrollará una arquitectura GNC (Guidance, Navigation and control) que se encargue de realizar y controlar la planeación de la trayectoria, el control y la navegación del *quadrotor*. Esto se hará combinando dos conceptos: *GPS Waypoints* y Perfil de velocidad trapezoidal. El primer concepto hace referencia al uso de puntos de ruta definidos previamente en términos de latitud y longitud, recorriendo éstos usando los datos de medición arrojados por el sistema de posicionamiento global (GPS). El segundo concepto se refiere a un método de planeación de trayectorias y ejecución de trayectorias por medio de una línea recta en el espacio, definiendo tres etapas en el recorrido: aceleración, velocidad constante y desaceleración.

Uno de los puntos diferenciadores de este proyecto, es que se busca que esta arquitectura GNC que se planea desarrollar quede como software libre, de tal manera que otros proyectos y trabajos puedan utilizarlo y realizar nuevos avances con él. Para esto, es fundamental el uso de una de las plataformas de desarrollo y programación más conocidas en el campo de la robótica: ROS (Robot Operating System). En este entorno de desarrollo se puede hacer uso de las librerías y paquetes ya desarrollados para diferentes tareas y aplicaciones, lo que disminuye el tiempo de desarrollo y la complejidad del trabajo (algo necesario debido a los requerimientos del mismo) para el desarrollo de un nuevo paquete de este entorno llamado *ardrone_GNC*, el cual integrará todo lo necesario para la navegación autónoma del *quadrotor* por medio de *GPS Waypoints* utilizando el Perfil de Velocidad Trapezoidal. Asimismo, permite que al finalizar, otras instituciones y personas continúen sobre las bases que aquí se construyan.

2. OBJETIVOS

2.1. Objetivo General

- Desarrollar e implementar un algoritmo de planeación y seguimiento de trayectorias espaciales para el AR.*Drone*2.0 mediante GPS *waypoints*.

2.2. Objetivos Específicos

- Desarrollar e implementar en lenguaje C++ un planificador de trayectorias en el espacio con perfil de velocidad trapezoidal.
- Desarrollar e implementar en lenguaje C++ un algoritmo de seguimiento de trayectorias para el AR*Drone*2.0 haciendo uso del GPS y la unidad inercial (IMU) abordo del robot.
- Integrar el planificador y seguidor de trayectoria dentro del entorno ROS (Robot Operating System) en un paquete llamado ar*drone*_GNC.
- Realizar pruebas en simulación con la ayuda del paquete de ROS: Tum_simulator[1]
- Realizar pruebas experimentales con el AR*Drone*2.0 y evaluar el desempeño del sistema en términos de la cercanía a los puntos de referencia de la trayectoria.

[1] http://wiki.ros.org/tum_simulator

3. MARCO TEÓRICO

3.1. UAV's y *quadrotor*s

El control autónomo de los robots se presenta como una dificultad en la actualidad y en el futuro de muchas aplicaciones como, por ejemplo, exploración de áreas de difícil acceso humano o ambientes contaminados, que exigen el desarrollo de plataformas móviles autónomas capaces de maniobrar de una manera ágil y estable en lugares donde no es posible la asistencia humana. Por esta razón, es válido trabajar con helicópteros de cuatro rotores de despegue y aterrizaje vertical, o como se conocen por sus siglas en inglés, "VTOL" (Vertical takeoff and landing). Estos helicópteros, también llamados "*quadrotor*s", son muy prácticos para trabajos donde se requiere gran capacidad de maniobra a bajas velocidades, lo que se puede lograr debido a su reducido peso y tamaño.

Actualmente, se han desarrollado muchas investigaciones sobre la planeación de trayectorias para este tipo de robots. Sin embargo, la mayoría de estas técnicas se basan en la navegación visual [2], por lo que no es posible planear con anterioridad una determinada trayectoria. Algunos proyectos anteriores ya han desarrollado algunos temas que sirven como marco de referencia para este trabajo, como el control de vuelo para este tipo de aeronaves [1], [4], los fundamentos de la navegación por waypoints [6], el seguimiento de trayectorias usando el GPS y la unidad de medición inercial [3], [8] y finalmente en [5] se desarrolla un enfoque al control de posición por medio de GPS waypoints.

3.2. ROS

ROS puede verse como un conjunto de librerías que facilitan el desarrollo de programas para controlar diversos tipos de robots. Es decir, brinda al programador un cierto nivel de abstracción en la gestión de tareas como: comunicación, lectura de sensores (drivers), simulación, interfaz de usuario (UI), entre otros. Es un software gratis de distribución libre, permite la reutilización de paquetes de manera que el código pueda ser mejorado y publicado nuevamente. Actualmente, está soportado únicamente por Ubuntu y se encuentra en etapa experimental para Mac, Windows, Fedora, entre otros.

3.2.1. Cómo funciona

La arquitectura de ROS está dividida en tres secciones o niveles: nivel de sistema de archivos, nivel gráfico y nivel de la comunidad (usuario).

3.2.1.1. Nivel de sistema de archivos

Los conceptos de este nivel se refiere principalmente a los recursos de ROS que se encuentran en el disco como:

- Paquetes: son la estructura y contenido mínima para la creación de un programa dentro de ROS. Contienen los archivos ejecutables (nodos), los servicios, mensajes, etc.
- Manifests: estos archivos contienen la información de los paquetes como las licencias y las dependencias.
- Pilas: son paquetes especiales que sirven para representar un grupo de paquetes relacionados entre sí.
- Mensajes: los mensajes son la información que se envía de un proceso a otro por medio de archivos. Los archivos guardados en *my_package/msg/MyMessageType.msg* definen la estructura para los mensajes en ROS.

Este nivel tiene la siguiente estructura:

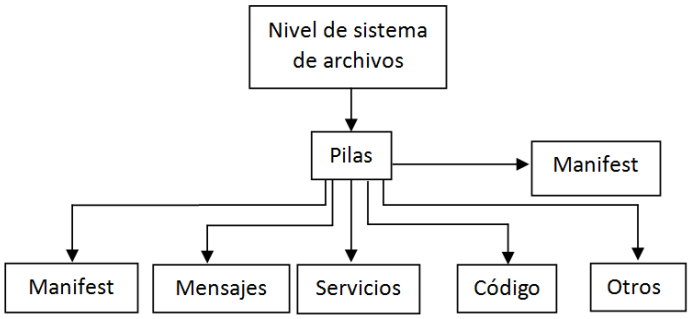

Figura 1. Estructura del sistema de archivos.

3.2.1.2. Nivel Gráfico

Es el nivel donde todos los procesos se conectan. Es una red punto a punto de ROS en la que los nodos pueden interactuar entre ellos transmitiendo datos y permitiendo visualizar estos de manera gráfica.

Su estructura es la siguiente:

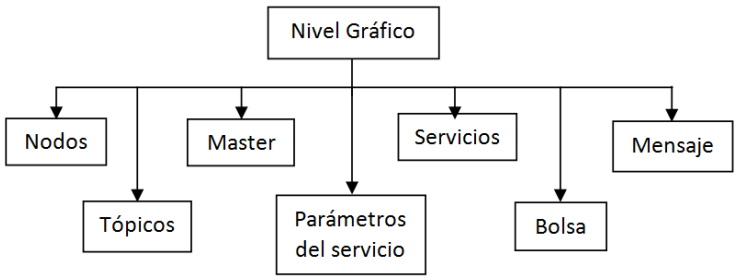

Figura 2. Estructura del nivel gráfico.

- Nodos: es un archivo ejecutable que tiene un propósito específico. Usualmente un programa completo requiere de varios nodos trabajando simultáneamente. Estos nodos usualmente están escritos en lenguajes como C++ o Pyton.
- Master: es el encargado de registrar los nodos y de dar acceso a estos para que se comuniquen. Sin esta se imposibilita la conexión entre los nodos, el envío de mensaje, la publicación a tópicos, etc.
- Tópico: son los mecanismos mediante los cuales los nodos intercambian mensajes por medio del modelo publicado/suscriptor. Los nodos pueden realizar una publicación a un tópico específico permitiéndole a otros nodos suscribirse para obtener datos de este tema.
- Servicios: otro mecanismo de comunicación entre nodos, pero a diferencia del anterior, éste es una comunicación de un nodo a otro que funciona mediante comunicación de petición/respuesta. Es decir, el primer nodo envía una petición a otro y este responde. En este modelo sólo dos nodos interactúan, uno el que envía y otro que lo recibe.
- Mensajes: los nodos se comunican a través mensajes. Estos son simples estructuras de datos de tipos primitivos (enteros, flotantes, booleanos, etc). Los mensajes pueden incluir estructuras anidadas y arreglos.
- Bolsas: es un formato que permite almacenar y posteriormente reproducir datos de ROS.

El siguiente diagrama muestra cómo funcionan los modelos de servicio (petición/respuesta) y el de tópicos (publicador/suscriptor):

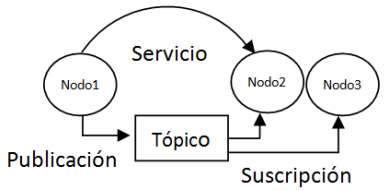

Figura 3. Modelo de comunicación entre nodos.

Muchos nodos pueden suscribirse y publicar en un mismo tópico pero solo un nodo puede recibir mensajes enviados a través de un servicio.

3.2.1.3. Nivel de la comunidad

Los conceptos de este nivel son recursos de ROS que permiten a usuarios de diferentes partes del mundo compartir la información y los desarrollos alcanzados. Estos recursos son:

- Distribución [23]: son colecciones de pilas que se pueden instalar en ROS. Estas distribuciones son similares a las distribuciones de Linux, como por ejemplo Ubuntu.
- ROS Wiki: es el foro principal donde se discuten y preguntan temas de ROS. Cualquier usuario de este sistema operativo puede preguntar y comentar, así como también dar solución a problemas de otros.

Es el canal de comunicaciones más utilizado que trata sobre las nuevas actualizaciones de ROS. También cuenta con un foro en el que se discuten temas de este software.

3.2.2.Paquetes a utilizar

Para el proyecto desarrollado se usaron tres paquetes de ROS: "ar*drone*_autonomy", "tum_simulator" y el "*drone*_GPS". El paquete de "ar*drone*_autonomy" se encarga de realizar todo el puente de comunicación con el AR *Drone* 2.0, además de realizar el control de bajo nivel del mismo, controlando las velocidades de los rotores, el ángulo de inclinación de Roll, Pitch y Yaw, así como obtener las lecturas de los sensores del mismo. Todo esto para poder mandar comandos al *quadrotor* en términos de las velocidades deseadas en cada eje. Esto permite que se puedan concentrar los esfuerzos del proyecto en realizar un buen método de movimiento, mientras que el paquete se encarga de toda la comunicación y el control de bajo nivel. El "tum_simulator" es usado para realizar las simulaciones de los diferentes nodos, ya que este permite tener un entorno virtual donde se puede observar el comportamiento del *drone* y su movimiento usando Gazebo. Por último, el paquete "*drone*_GPS" es usado para dos finalidades, la primera es la de realizar la conversión de las coordenadas recibidas del GPS (latitud, longitud) a coordenadas UTM (X, Y) que brindan muchas facilidades a la hora de realizar los nodos de movimiento. La segunda finalidad es la de realizar el registro de datos en archivos que quedan guardados automáticamente luego de cada prueba, esto con el fin de poder registrar las diferentes trayectorias y pruebas.

3.3. Coordenadas GPS y UTM

3.3.1.¿Qué es un GPS?

El sistema de posicionamiento global GPS (Global Position System) por sus siglas en inglés, es un sistema global de navegación por medio de satélites, el cual permite determinar, en cualquier parte del mundo, la posición de una persona, vehículo u objeto con una precisión de hasta centímetros.

El GPS cuenta con 3 segmentos. El primer segmento formado por los satélites, es llamado "segmento espacial". El segundo segmento, el cual hace referencia a las estaciones de control, se conoce como "segmento de control". El tercero, denominado "segmento del usuario", corresponde a los dispositivos GPS.

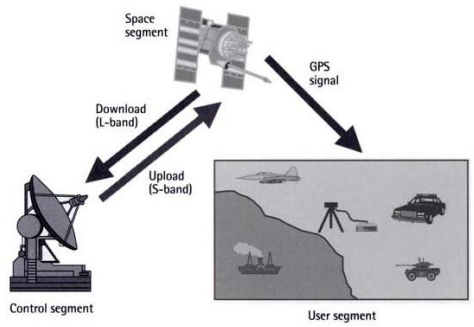

Figura 4. Segmentos del sistema GPS. Tomada de [21].

A continuación se explicarán brevemente estos segmentos.

3.3.1.1. Segmento Espacial (SS)

El segmento espacial está formado por una red de 24 satélites ubicados en la órbita media terrestre (MEO) a una altura de 20200Km, los cuales forman 6 órbitas diferentes con 4 satélites en cada una, con el fin de cubrir toda la superficie de la tierra. La figura 5 muestra cómo es esta distribución, la cual asegura que el usuario pueda recibir información de por lo menos cuatro de ellos sin importar el lugar donde se encuentre. Hay otros satélites que se reservan en caso de emergencia.

Cada órbita es cuasi-circular y consta de una inclinación de 55° con respecto al ecuador. El periodo orbital es de aproximadamente 12 horas (11horas y 58 minutos).

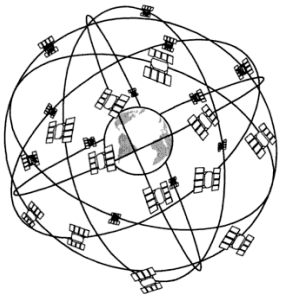

Figura 5. Constelación de satélites. Tomada de [20].

3.3.1.2. Segmento de Control (CS)

Este segmento consiste en una red global de cinco estaciones terrestres de seguimiento y control distribuidas alrededor de todo el planeta. Estas estaciones realizan el seguimiento de los satélites, miden y predicen su posición, y miden y ajustan los relojes internos. Además, se encargan de analizar y monitorear las señales emitidas de manera que puedan evaluar el desempeño, y envían a los satélites los datos que deben transmitir y las correcciones que deben realizar.

3.3.1.3. Segmento del usuario (US)

Está conformado por los equipos de recepción y por la comunidad de usuarios GPS. Los receptores GPS convierten la información recibida en posición, velocidad y tiempo.

3.3.2. Principio de Funcionamiento

Para realizar los cálculos, un receptor GPS debe contar con la información de al menos tres satélites. Sin embargo, esto no representa una dificultad ya que, como mínimo, hay cuatro satélites en cualquier instante de tiempo (esto gracias a la distribución de los satélites previamente mencionada). Cada satélite transmite de manera permanente un mensaje de navegación que contiene su posición orbital y el momento exacto en el que el mensaje fue emitido. La medición de la distancia se basa en la propagación de las ondas electromagnéticas. Los mensajes son enviados mediante ondas de radio que se propagan aproximadamente a la velocidad de la luz (300000Km/s) de manera que cuanto más alejado esté el receptor del satélite, más tiempo se tardará en llegar la señal. Para realizar este cálculo, únicamente es necesario conocer el tiempo de retraso y utilizar la formula Tret x VelLuz = distancia.

El GPS calcula la posición mediante el método de trilateración que sigue los siguientes pasos:

Paso 1: cuando el receptor GPS recibe una señal del primer satélite, se traza una esfera imaginaria centrada en el satélite y de radio igual a la distancia que separa a éste del receptor. Este receptor asume que se encuentra en alguna posición sobre la superficie de la esfera.

Paso 2: al recibir una segunda señal de un segundo satélite, se calcula la distancia y se traza una nueva esfera imaginaria. La superficie de esta esfera tendrá una intersección con la superficie de la primera.

Paso 3: el receptor GPS realiza lo mismo que en los pasos anteriores, pero ahora con un tercer satélite. Esta nueva esfera se intersectará con la intersección creada en paso 2, dando dos posibles puntos como solución: el primero en el espacio y el segundo sobre la superficie de la tierra. El receptor discrimina el punto del espacio y selecciona como punto válido el segundo. Con estos datos es posible conocer la latitud y la longitud en la que se encuentra el dispositivo.

Para calcular la altura, es necesario recibir una cuarta señal de otro satélite diferente y nuevamente realizar el cálculo de la distancia, trazar la nueva esfera imaginaria y mirar la intersección con esta.

3.3.3. DGPS

El DGPS o GPS diferencial es un sistema de posicionamiento global que además de recibir y procesar la información enviada por los satélites, recibe información de una estación terrestre conocida que le indica el error introducido por cada satélite para así tener una mayor exactitud en los datos del dispositivo. Los errores de estos GPS llegan al orden de centímetros.

3.3.4. Errores

Existen muchas fuentes de error que afectan la precisión del GPS. Algunas de estas se deben a los satélites y a las estaciones terrestres, sin embargo otras son de origen natural. Las fuentes de error más importantes incluyen:

- Errores por reloj: sucede cuando los relojes de los satélites y los receptores no están sincronizados luego el tiempo con el que se realizan los cálculos de posición no son exactos.
- Retrasos atmosféricos y ionosféricos: cuando la señal de radio emitida por el satélite pasa por la ionosfera su velocidad disminuye, esto debido a que tiene que pasar a través de diferentes elementos y componentes naturales. A esto se le suma que la longitud por la que pasa la señal no es constante ya que los receptores no siempre se van a encontrar a la misma distancia de los satélites. Esta disminución de la velocidad causa una recepción con un tiempo mayor creando un error de retraso.
- Disponibilidad selectiva (S/A): es una manipulación de la información emitida por los satélites creando una pequeña alteración en los tiempos, de manera que los datos de posicionamiento global de los dispositivos no sean tan exactos. Esta manipulación es hecha por el departamento de Defensa

de Estados Unidos, con el fin de evitar la excesiva precisión de los receptores comerciales por motivos de seguridad.

- Efecto multitrayectoria: es causado por reflexiones de la señal emitida por el satélite en diferentes superficies u objetos cercanos al receptor. Usualmente este error aparece cuando el dispositivo GPS se encuentra cerca de una superficie reflectora, tal como un edificio o una casa.
- Dilución de la precisión (DOP): es la fuente de error que depende puramente de la geometría, es decir, de la posición de los satélites. A mayor separación entre estos se tiene un menor error.

3.3.5. Coordenadas Geodésicas

Con un receptor GPS, es posible conocer el posicionamiento de un objeto en cualquier instante de tiempo de manera que se puede hacer un seguimiento de la trayectoria o del recorrido de éste; esto es lo que se conoce como navegación GPS. Durante el recorrido, es posible almacenar los datos de la posición del dispositivo. A estos puntos se les conoce más comúnmente como Waypoints. Cada waypoint dispone de un conjunto de datos dados por las coordenadas geodésicas, las cuales utilizan como referencia dos coordenadas angulares: Latitud (Norte-sur) y Longitud (Este y Oeste). La latitud y la longitud miden el ángulo entre cualquier punto y el ecuador y el meridiano de Greenwich, respectivamente.

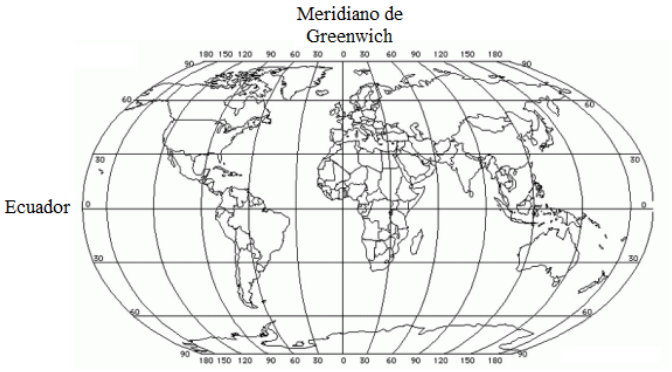

Figura 6. Mapa de coordenadas geodésicas. Tomada de [22].

3.3.6. Coordenadas UTM (Universal Transversal de Mercator)

Es un sistema de coordenadas basado en una proyección cilíndrica conforme[2] de la tierra donde se presenta una red de líneas rectas horizontales y verticales que representa paralelos y meridianos, respectivamente. Este sistema está basado en la proyección cartográfica transversa de Mercator pero en lugar de hacerla tangente al Ecuador, se hace tangente a un meridiano.

Las coordenadas UTM son similares a las geodésicas, pero utilizan como unidad de medida los metros en lugar de grados, minutos y segundos. La cuadrícula UTM fracciona el planeta en 60 zonas diferentes, cada una de 6 grados de longitud (líneas verticales) y en 20 bandas de 8 grados de latitud (líneas horizontales denominadas con letras desde la C hasta la X) como se puede observar en la figura 7.

[2] Conforme: Conserva los ángulos de las líneas de latitud y longitud

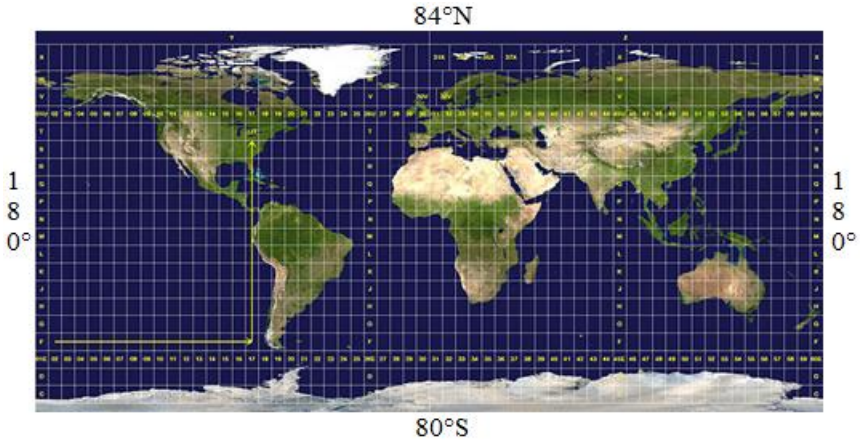

Figura 7. Mapa de coordenadas UTM. Tomada de []

A la línea central de un huso UTM se le denomina meridiano central y siempre se hace coincidir con el sistema geodésico tradicional. Este meridiano central define el origen de la zona UTM y tiene coordenadas 500Km este, 0km norte (cuando se considera el hemisferio norte) o 500Km este, 10.000km norte (cuando se considera el hemisferio sur). Si una banda tiene una letra igual o mayor que N entonces se encuentra en el hemisferio norte.

El valor de una coordenada UTM no corresponde a un punto específico, sino a un área cuadrada cuyos lados dependen de la resolución de la coordenada. Cualquier punto comprendido dentro de este cuadrado tiene el mismo valor de coordenada. Cuanto mayor sea la resolución de ésta, la medida será más exacta ya que el área será menor.

3.3.7. Conversión coordenadas geodésicas a UTM.

En este trabajo de grado se trabajará con coordenadas UTM ya que estas coordenadas están dadas en metros y, para la aplicación práctica, simplifican los cálculos y le facilitan al usuario el ingreso de datos. Se utilizará el paquete *drone*_GPS (desarrollado por estudiantes javerianos) que cuenta con un algoritmo que realiza la conversión automática de coordenadas geodésicas a UTM, de manera que se pueda determinar el desplazamiento del *drone* en unidades métricas. Además, este paquete se encarga de generar una estructura de datos del GPS y de la navegación.

3.4. Arquitectura GNC

En concreto, este trabajo de grado busca el desarrollo de una arquitectura GNC (Guidance, navigation and control) para el *quadrotor* AR.*Drone*2.0. El módulo *Guidance* hace referencia a la planificación de una trayectoria espacial que determina las coordenadas cartesianas del movimiento (X,Y,Z), los vectores de velocidad y aceleración asociados a cada punto cartesiano y a la orientación del robot a lo largo del seguimiento de la trayectoria. El módulo *Navigation* se refiere al sensado/percepción del entorno espacial por parte del robot para poder garantizar el seguimiento de la trayectoria. Se hará uso de tres sensores: GPS para sensado de posición espacial (latitud, longitud), sensor de presión+GPS para la estimación de altura e IMU para el sensado de componentes angulares de movimiento. Finalmente, el módulo *Control* permite el seguimiento de la trayectoria espacial de referencia con el mínimo error posible.

3.5. Perfil de velocidad trapezoidal

Para realizar el planeador de trayectorias y hacer que el *quadrotor* vaya de un punto inicial a un punto final generando un segmento de recta en el espacio cartesiano, se decidió hacer uso del método del *Perfil de velocidad trapezoidal*. Para esto se puede partir de la ecuación de la recta en el espacio y expresar cada una de las coordenadas espaciales en función del tiempo, utilizando las ecuaciones paramétricas de la recta.

$$f(x, y, z) = \lambda(t)(P1 - P0) + P0$$

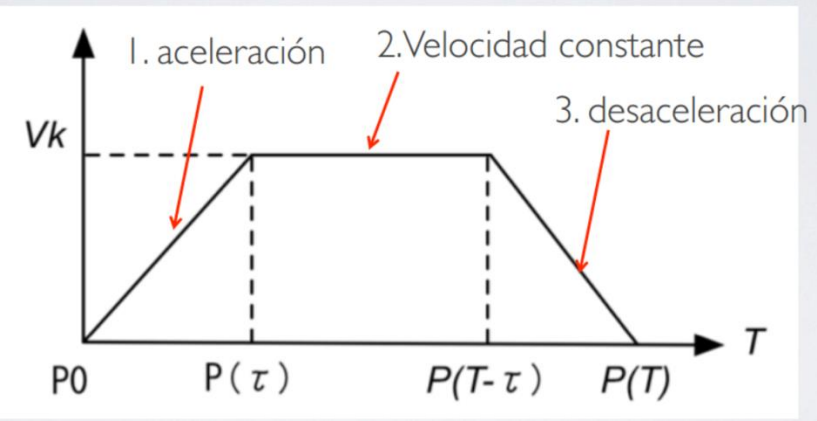

Figura 8. Gráfica del perfil de velocidad trapezoidal.

El concepto básico de este perfil para líneas rectas en el espacio es que se divide la trayectoria en tres tramos del recorrido. Un primer tramo de movimiento acelerado donde se pasa de velocidad cero a la velocidad máxima que se alcanza durante la trayectoria. El segundo tramo corresponde al de velocidad máxima, durante este tramo la velocidad es constante. El último tramo es de desaceleración, donde se pasa de la velocidad máxima a velocidad cero nuevamente. De esta manera, para que el *quadrotor* se desplace entre dos puntos, deberá pasar por estas tres etapas, con determinado tiempo en cada uno. Esto se puede observar en la figura 8.

Se parte de las ecuaciones ya desarrolladas donde los segmentos de recta se pueden representar en la siguiente función a tramos,

$$f(t) = \begin{cases} \dfrac{1}{|P(\tau) - P(0)|}\left(\dfrac{1}{2}a * t^2\right)\big(P(\tau) - P(0)\big) + P(0) & ,0 \le t \le \tau \\ \dfrac{vk * t}{|P(T-\tau) - P(\tau)|}\big(P(T-\tau) - P(\tau)\big) + P(\tau) & ,\tau \le t \le T - \tau \\ \dfrac{1}{|P(T) - P(T-\tau)|}\left(\dfrac{1}{2}a * t^2 + vk * t\right)\big(P(T) - P(T-\tau)\big) + P(T-\tau) & ,T - \tau \le t \le T \end{cases}$$

donde vk es la velocidad máxima dada por,

$$vk = \frac{|P(T) - P(0)|}{T(1 - pt)}$$

definiendo *pt* como el porcentaje de aceleración (entre 0 y 1), este porcentaje indica cuánto tiempo del tiempo total estará destinado para los tramos de aceleración y desaceleración. La aceleración está dada por,

$$\pm a = \frac{vk}{\tau}$$

El segmento con $0 \le t \le \tau$ corresponde al acelerado, el que va de $\tau \le t \le T - \tau$ corresponde al de velocidad constante y el que va de $T - \tau \le t \le T$ corresponde al desacelerado.

Para poder resolver el sistema se deben conocer los puntos intermedios de la trayectoria $P(\tau)$ y $P(T - \tau)$. Estos se pueden hallar con las siguientes ecuaciones:

$$P(\tau) = \left(\frac{1}{2}a * t^2\right)\frac{P(T) - P(0)}{|P(T) - P(0)|} + P(0)$$

$$P(T - \tau) = \frac{vk * (T - 2\tau)}{|P(T) - P(0)|}P(T) - P(0) + P(\tau)$$

Con τ definido como:

$$\tau = T * pt$$

Con estas ecuaciones ya definidas, basta con saber el punto inicial y el final para empezar a despejar los distintos valores. En esta etapa también se requiere o el valor de la velocidad máxima o el tiempo total en el que se va a realizar el recorrido. Sin embargo, debido a que para este proyecto se va a trabajar con distancias variables y recorridos que pueden llegar a ser muy grandes uno comparado con el otro, no se puede definir un tiempo predeterminado. Lo correcto es determinar la velocidad máxima que se desea que alcance el *quadrotor* (se pretende usar un valor cercano a los 1.5 m/s) y el porcentaje de aceleración para que a partir de estos dos se halle el valor de tiempo total. Se piensa utilizar un porcentaje de aceleración de 30%. Con estas condiciones se asegura que el *quadrotor* no va a trabajar forzado y siempre va a realizar cada tramo en un tiempo proporcional a la distancia a recorrer.

De esta manera, el proceder del algoritmo a desarrollar empieza haciendo la resta vectorial $|P(T) - P(0)|$ de los puntos final menos inicial. Se pasa a calcular el tiempo que requiere para realizar el recorrido planteado en términos de la distancia entre ambos puntos y con este tiempo total se calculan los tiempos de aceleración y desaceleración. Luego se calcula la aceleración requerida a partir de los valores hallados. Posteriormente se calculan los puntos intermedios de la trayectoria $P(\tau)$ y $P(T - \tau)$, correspondientes a las intersecciones entre los tramos usando las ecuaciones mostradas anteriormente. Con los puntos se realizan las restas vectoriales correspondientes para conocer la distancia que se debe recorrer en cada tramo y usando la función a trozos definida arriba, se calculan las funciones de la recta para cada tramo del recorrido. Con estos cálculos ya se puede conocer cada punto del recorrido que tiene que realizar el *quadrotor* de manera precisa y se le pueden dar las órdenes correspondientes para realizar la trayectoria.

Para continuar con el proceso y teniendo en cuenta que estas ecuaciones deben ser implementadas a manera de algoritmo en C++ para darle las ordenes al *quadrotor*, se debe definir un número de puntos a usar en toda la trayectoria. Con esto se puede hacer un muestreo y con estos puntos y las ecuaciones halladas para cada tramo de la trayectoria, se puede saber por qué punto debe pasar el *quadrotor* en determinado momento.

4. ESPECIFICACIONES

4.1. Generalidades

Con este trabajo se busca desarrollar un algoritmo qué, por medio del uso del AR *Drone* 2.0, sea capaz de realizar una serie de tareas específicas encaminadas al vuelo autónomo y seguimiento de trayectorias determinadas por medio de puntos de ruta en coordenadas UTM. Por medio de un nodo o paquete software implementado en el entorno de desarrollo para robótica ROS, junto con el uso del *drone* comercial AR *Drone* 2.0 se logrará que el UAV realice una trayectoria definida con anterioridad por el usuario. Es necesario que se tenga la capacidad de planear una trayectoria a partir de una serie de puntos de ruta que serán definidos por el usuario en forma de coordenadas UTM, una variante al sistema típico de GPS, debido a la simplicidad de usar unas coordenadas X, Y, fácilmente aproximables al plano cartesiano en 2D. Se trabaja con estas coordenadas para facilitar el uso del programa. Para introducir estos puntos de ruta, el usuario va a contar con una interfaz de usuario que le permita introducir los datos, cuántos puntos de ruta desea y en qué lugares exactos los desea. La mayoría de los parámetros básicos del *quadrotor* como la velocidad máxima, la altura máxima, la altura de operación y los controles de roll, pitch y yaw no podrán ser definidos por el usuario. Debido a que el tiempo del recorrido depende de la separación entre los distintos puntos de ruta, resulta mucho más seguro si el sistema lo calcula para cada trayectoria en vez de que lo defina el usuario. Una vez se tienen los puntos de ruta requeridos, el paquete debe hacer los cálculos correspondientes y pasarle al paquete de ROS ya desarrollado y funcional, el ar*drone*_autonomy, una serie de comandos y órdenes predeterminadas de velocidad en sus diferentes componentes (x, y, z y rotación) para que éste se los comunique al UAV. Adicionalmente, otro paquete de ROS ya desarrollado llamado *Drone*_GPS (encargado de recibir los datos del GPS a bordo del A.R*Drone*2.0 y convertirlos a coordenadas UTM) proporcionará las entradas de la posición actual para que se pueda realizar el control correspondiente.

Adicionalmente, es importante tener en cuenta el concepto de arquitectura GNC (Guidance, Navigation and Control) que fue mencionado anteriormente para poder entender el desarrollo de software que compone la totalidad del trabajo de grado. Se realizará un control autónomo de trayectoria para un robot AR*Drone*2.0 que se implementará dentro de un nuevo paquete ROS llamado ar*drone*_GNC. Como se observa en la Figura 2, el paquete creado tendrá una interfaz de usuario que tendrá el objetivo de recibir los puntos de ruta y leerlos hacia el programa, un módulo donde se realizará la planificación de la trayectoria y un módulo GNC que se encargará de estar en contacto permanente con el paquete ar*drone*_autonomy (paquete de ROS) para ejecutar la trayectoria.

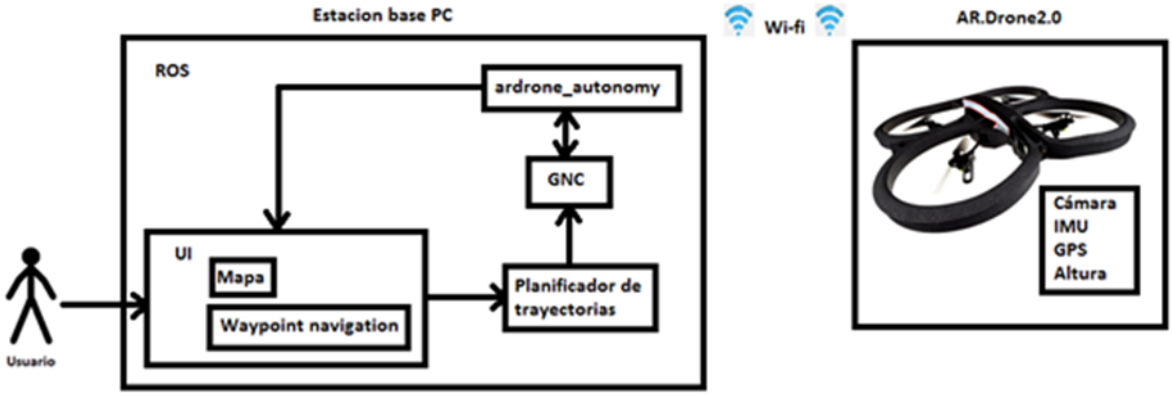

Figura 9. Diagrama general del proyecto.

Como se mencionó anteriormente, se busca trabajar con una arquitectura GNC. En concreto, para la planificación de la trayectoria espacial del robot se trabajará con el método conocido como "Perfil de Velocidad Trapezoidal", en el cual se busca llegar de un punto A a un punto B trazando una línea recta, especificando el tiempo de aceleración, el tiempo de velocidad constante y el tiempo de desaceleración. Adicionalmente, por medio de los GPS waypoints o puntos de ruta, se busca que la planeación de la

trayectoria se vuelva algo tan simple como seleccionar determinados puntos de ruta en un mapa satelital para que el robot los recorra sin necesidad de una interacción adicional.

Para realizar el bloque descrito como Control, debido a la gran dificultad de la dinámica de estos dispositivos y a las restricciones de tiempo, se hará uso de un modelo dinámico y de control ya creado para realizar las simulaciones iniciales, en donde bastará realizar la calibración de las variables Kp, Ki y Kd del sistema para que el *quadrotor* se comporte de la manera deseada. Sin embargo para la implementación real del algoritmo de control, solo se hará uso de un control proporcional que se encargue de controlar el error de posición y que permita hacer las correcciones necesarias con respecto a la velocidad y a la posición.

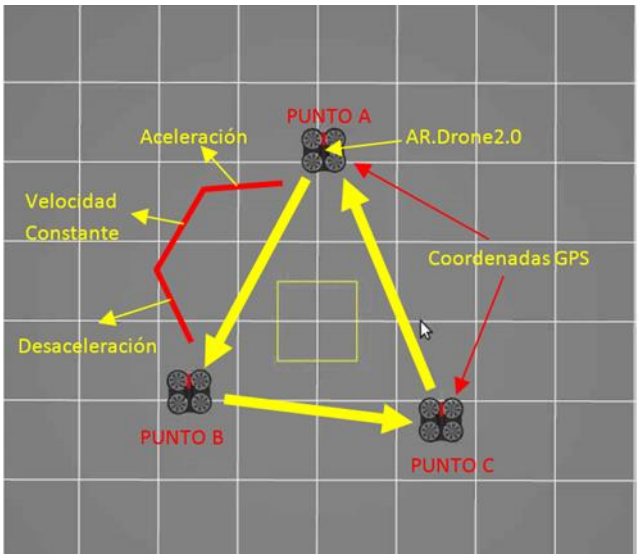

Figura 10. Ejemplo del perfil de velocidad trapezoidal.

Los tres módulos se integrarán en el paquete ar*drone*_GNC mediante programación en lenguaje C++, que permitirá realizar la planeación de la trayectoria y la ejecución de esta, seleccionando los puntos de ruta desde un mapa satelital de determinada área. Esto por medio de una interfaz de usuario en la que el usuario pueda seleccionar los puntos UTM (X y Y) a seguir.

Ya se habló de lo que busca desarrollar el proyecto y de la implementación software que se debe tener, sin embargo, es también importante hablar un poco del hardware que utilizará el proyecto, tanto el UAV como la estación base.

4.2. Estación Base

La estación base del proyecto se trata básicamente de un computador portátil, encargado de establecer la comunicación con el AR *Drone* 2.0 por medio de una conexión Wi-Fi generada por el *drone*. Se utiliza el sistema operativo Ubuntu 12.04 con el entorno de desarrollo para robótica, ROS, instalado en su versión Fuerte. En esta versión de ROS se instalan una serie de paquetes y drivers que permiten acceder a la información del AR *Drone* 2.0 y a su vez, mandarle comandos de vuelo y control. Más especificadamente, el paquete "ar*drone*_autonomy" se encarga de establecer el puente de comunicaciones entre la estación base y el *quadrotor* para la lectura de los sensores (IMU, GPS, etc) y mandar todos los comandos de control y movimiento. El "*drone*_GPS" se encarga se tomar las lecturas del GPS realizadas y almacenar las sesiones de vuelo, así como también, de realizar la conversión de las coordenadas a el sistema UTM, que resulta más sencillo para trabajar. Por último, el paquete creado en este trabajo, el "*drone*_GNC", también se encuentra instalado. El lenguaje usado para la programación de las funciones desarrolladas y el paquete creado es C++ y C. Debido a que para este proyecto, el procesamiento es puramente matemático y no se trabaja con imágenes, la capacidad de procesamiento del computador no es una característica de vital importancia, sin embargo, se recomienda trabajar con un computador similar al usado, Dell XPS 13 Ultrabook con

procesador Intel Core I5. Estas especificaciones son más que suficientes para correr los algoritmos desarrollados.

4.3. AR *Drone* 2.0

El AR *Drone* 2.0 es un *quadrotor* de bajo costo y fácil implementación originalmente pensado como un juguete o una entretención más que como un *drone* para realizar tareas más robustas. Debido a esto sus características resultan un poco limitadas para determinadas tareas, sin embargo, para fines investigativos y para los alcances específicos de este proyecto, presenta una gran ventaja en términos de facilidad de uso y costo. Cuenta con una cámara HD 720p a 30 FPS, un procesador ARM Cortex A8 de 1 GHz con video DSP TMS320DMC64x, memoria RAM DDR2 de 1GB a 200MHz, Wi-Fi b g n y una entrada USB 2.0 entre otras cosas. Adicionalmente cuenta con la unidad de medición inercial (IMU) que está compuesta por un giroscopio de 3 ejes con precisión de 2000°/s, un acelerómetro de 3 ejes con una precisión de +/- 50mg, un magnetómetro de 3 ejes con precisión de 6°, sensor de presión con precisión de +/- 10Pa, sensores de ultrasonido para medir altitud y una cámara vertical QVGA a 60 FPS para medir la velocidad de avance. Para realizar la conexión con la estación base usa la antena WI-FI incorporada, creando su propia red a la que el computador o dispositivo debe conectarse. Por este medio se pueden mandar los comandos de vuelo y se pueden recibir los datos de navegación. Adicionalmente se cuenta con un registrador de vuelo con GPS que añade geolocalización al AR *Drone* 2.0 y le permite grabar aún más parámetros de vuelo. Este dispositivo se conecta al *drone* por medio de su ranura de USB 2.0 y permite que se extraigan los datos de latitud y longitud para poder implementar el resto de funcionalidades que se requieren.

Figura 11. GPS Flight Recorder utilizado en el proyecto.

Debido a que se trata de una plataforma de bajo costo que en muchos casos es usada como un juguete, algunas especificaciones que presenta no son óptimas para la realización del proyecto, y por lo tanto, limitan su alcance. La corta duración de la batería es quizá el mayor limitante debido a que solo aguanta 15 minutos de uso continuo. Adicionalmente, para realizar pruebas y tener un vuelo estable, se requiere usarlo en condiciones limitadas donde haya poco viento, se trabaje a poca velocidad y la altura de vuelo sea baja. Un problema adicional muy grande, es que el UAV no está fabricado para trabajar por encima de los 2000 metros de altura sobre el nivel del mar. Por consiguiente, la fuerza de sustentación resulta bastante baja y limita la altura que se puede alcanzar, el peso que puede cargar y la estabilidad con la que trabaja. Esto limita bastante su uso en determinadas áreas ya que se supone que se debe realizar un vuelo autónomo, pero si no se puede garantizar cierto nivel de sustentación, esto presenta bastantes dificultades. Por último, una de las limitaciones más grandes es el hecho de que la red de conexión que genera el *drone* es de máximo 50 metros, lo que impide que se recorran grandes distancias.

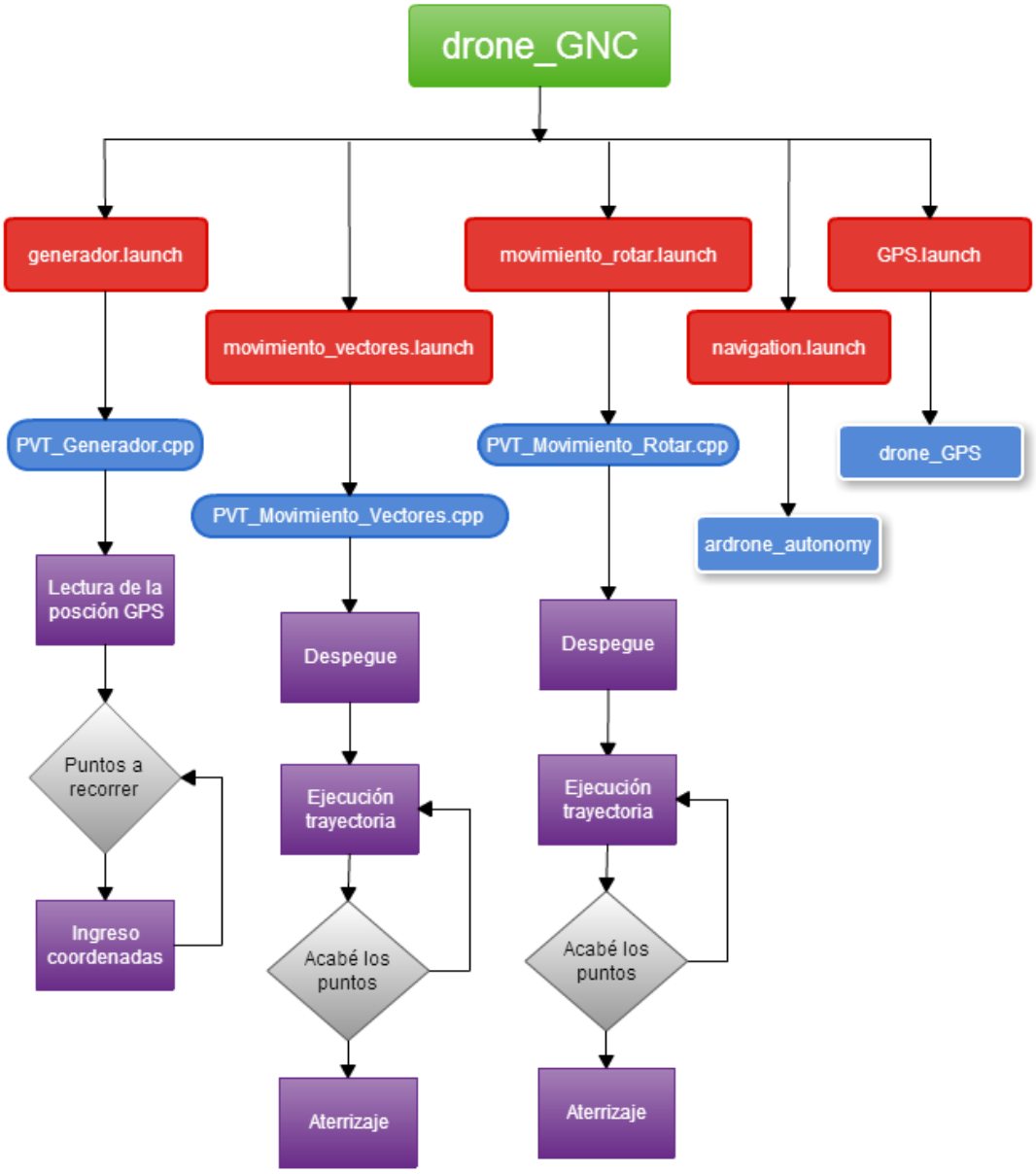

Figura 12. Diagrama de la estructura del paquete de ROS desarrollado "*drone*_GNC".

Basado en las especificaciones de la estación base y del AR *Drone* 2.0, así como en los requerimientos del proyecto, se planteó la siguiente tabla, con determinados parámetros específicos que se recomiendan para que el sistema pueda funcionar de manera óptima.

PARÁMETRO	ESPECIFICACIÓN
Altura de vuelo	Entre 1 y 2 metros
Altura del terreno	Entre 0 y 1600 metros sobre el nivel del mar

Velocidad máxima de vuelo (vk)	2 m/s
Clima	No debe haber viento considerable ni lluvia, la hora del día es irrelevante, aunque se recomienda volar de día.
Distancia entre el *drone* y la estación base	30 metros
Separación máxima entre los puntos de ruta	No importa, teniendo en cuenta que no se puede exceder el parámetro de distancia entre el *drone* y la estación base.
Características del terreno	Debe ser un campo abierto, sin obstáculos o resaltos que pudieran impedir la correcta ejecución de la trayectoria. Adicionalmente no debe haber muchos árboles y edificios para garantizar un correcto funcionamiento del GPS
Precisión del GPS	± 2 metros
Formato de ingreso de los puntos de ruta	Coordenadas UTM
Duración máxima del recorrido	Aproximadamente 15 minutos.

Tabla 1. Especificaciones del proyecto.

5. DESARROLLOS

El desarrollo principal de este proyecto se basa en el diseño y desarrollo de un conjunto de algoritmos en C++ para la navegación autónoma del AR *Drone* 2.0 usando GPS. En esta tarea, se seleccionó ROS como la plataforma de trabajo principal para el trabajo, debido a su gran cantidad de código de acceso libre y debido a que ofrece muchas bondades a la hora de juntar diferentes desarrollos. Para esto, como se explicó anteriormente, se deben crear unidades de organización de sistema básicas llamadas "paquetes", que a su vez contienen procesos específicos o ejecutables llamados "nodos". Para la construcción del paquete desarrollado en este proyecto, el "*drone*_GNC", se requieren 4 archivos fundamentales:

- Manifest: Es un archivo llamado manifest.xml en el que se incluye la licencia, el autor e información relevante del paquete. Aquí también se incluyen y definen las dependencias que tiene el paquete de ciertas librerías o de otros paquetes ROS.
- Mainpage: Es un archivo llamado en donde se incluye toda la documentación relevante sobre el paquete.
- CMakeLists: Es un archivo llamado CMakeLists.txt en el que se proporciona toda la información para la construcción del paquete.
- MakeFile: Es un archivo llamado makefile.m que contiene la instrucción cmake que llama la ejecución del archivo CMakeLists.

Una vez construido el paquete "*drone*_GNC", se desarrollaron los diferentes nodos de ROS necesarios para el correcto funcionamiento del sistema. Dentro de estos nodos, se encapsularon partes diferentes del proceso, de tal manera que resultara más sencillo depurar el trabajo para poder observar lo que se realiza en cada lugar del mismo. Estos diferentes nodos, que son guardados dentro de una carpeta llamada "src", se crean por separado en diferentes archivos con terminación .cpp, indicándole a ROS y al compilador que el lenguaje usado es C++.

Para el trabajo se crearon 3 algoritmos principales. Estos son guardados en los nodos "PVT_Generador", "PVT_Movimiento_Rotar" y "PVT_Movimiento_Vectores". El primero se encarga principalmente de realizar la planeación de la trayectoria y publicar una serie de datos usados posteriormente para realizar el movimiento. El segundo y tercero se encargan de leer estos datos generados y usarlos para transmitirle comandos al *drone*. Sin embargo, debido a que para este proyecto resultaba de vital importancia realizar pruebas en simulación antes de salir al mundo real, estos nodos de generación y movimiento fueron discriminados de acuerdo a su uso, esto debido a que muchas cosas debían ser modificadas a la hora de probar en simulación, tales como el hecho de la inexistencia de GPS en el mundo simulado.

Se procederá a explicar en detalle cada uno de los desarrollos realizados.

5.1. Configuración de los diferentes archivos .launch

Para la ejecución del paquete se crearon 5 archivos de lanzamientos que se encargan de cumplir diferentes tareas. Es necesario lanzar 4 de los 5 para el correcto funcionamiento, donde la opción es cuál de los dos métodos de movimiento se desea ejecutar.

- generador.launch: Se encarga de lanzar el nodo "PVT_movmiento_Generador" que se encarga de leer la posición inicial del *quadrotor* y pedir el número de puntos de ruta deseados y sus respectivas coordenadas.
- movimiento_vectores.launch: Se encarga de lanzar el nodo "PVT_Movimiento_Vectores" que se encarga de ejecutar la trayectoria calculada usando el método de los vectores de velocidad.
- movimiento_rotar.launch: Se encarga de lanzar el nodo "PVT_Movimiento_Rotar" que se encarga de ejecutar la trayectoria calculada usando el método de la rotación.

- navigation.launch: Se encarga de lanzar el nodo del paquete ar*drone*_autonomy para controlar el *drone*, comunicarse con el mismo y enviar los comandos de movimiento.
- GPS.launch: Se encarga de lanzar los nodos del paquete "*drone*_GPC" que se encargar de realizar la conversión de las coordenadas GPS a coordenadas UTM.

5.2. Algoritmo de generación y planeación de trayectorias

El proyecto realizado debe ser capaz de generar una trayectoria con un perfil de velocidad trapezoidal a partir de determinados puntos de ruta ingresados por el usuario. En este algoritmo se hace todo lo relacionado con el cálculo de las variables y los datos de navegación que van a ser necesarios en el momento de generar el movimiento. Luego de realizar los cálculos, el nodo publica los datos a un determinado tópico de ROS para que cualquier otro paquete pueda suscribirse a estos datos y trabajar con ellos.

Cómo se ha mencionado a lo largo del trabajo, el objetivo del paquete desarrollado es trabajar con una arquitectura de tipo GNC (Guidance, navigation and control). Esto implica que se deben realizar determinadas tareas de manera independiente. En concreto, el nodo de "PVT_Generador" se encarga de todo lo relacionado con la parte de Guidance de esta arquitectura. Esto debido a que el algoritmo se encarga de la planificación de una trayectoria espacial que determina las coordenadas cartesianas del movimiento (X, Y), los vectores de velocidad y aceleración asociados a cada punto cartesiano y a la orientación del robot a lo largo del seguimiento de la trayectoria.

La ejecución normal del nodo cuenta con tres partes principales: el ingreso de los puntos de ruta, el cálculo de las variables del perfil de velocidad trapezoidal y la publicación de las mismas. Debido a que el programa requiere una interacción con el usuario para el ingreso de los puntos, no se realiza una ejecución constante del mismo. Además, sería un gasto computacional innecesario debido a que durante cada iteración, el programa calcularía los mismos datos de manera recurrente. En vez de esto, lo que se hace es pasar por las tres etapas del programa una sola vez, y esperar a que el usuario decida realizar una nueva trayectoria.

Para que el programa tenga claridad de donde está ubicado en el momento del despegue, de debe realizar una suscripción al tópico "/drone_GPS/data" de donde se leen las coordenadas UTM que se establecen cómo el punto inicial del recorrido.

Una vez arranca el programa, se imprime en consola un mensaje hacia el usuario indicándole que ingrese la cantidad de puntos que desea recorrer, esto para tener claridad dentro de la ejecución del programa de cuantas iteraciones se deben realizar y mantener siempre un orden de los cálculos. Los puntos de ruta se deben ingresar en coordenadas UTM, que como se explicó anteriormente, entienden la tierra como un plano cartesiano con coordenadas X y Y. Esto decidió hacerse de esta manera para darle una mayor simplicidad al usuario en el uso del programa, ya que las coordenadas GPS pueden ser un poco más confusas a la hora de entenderlas en un plano como el trabajado. Aquí, se realiza una comprobación cada vez que el usuario ingresa una coordenada para prevenir un posible fallo del sistema, esto es, que al ingresar cada par de coordenadas, se calcula por pitágoras, la distancia del punto de origen, que es donde se encuentra la estación base, al punto deseado, si esta distancia es mayor de 30 metros, se tendrá que ingresar el punto nuevamente. Esto se hace debido a que el alcance del AR *Drone* 2.0 es de aproximadamente 30 metros, por lo que si se sale de este rango, podría perderse la comunicación con el mismo y causar algún error.

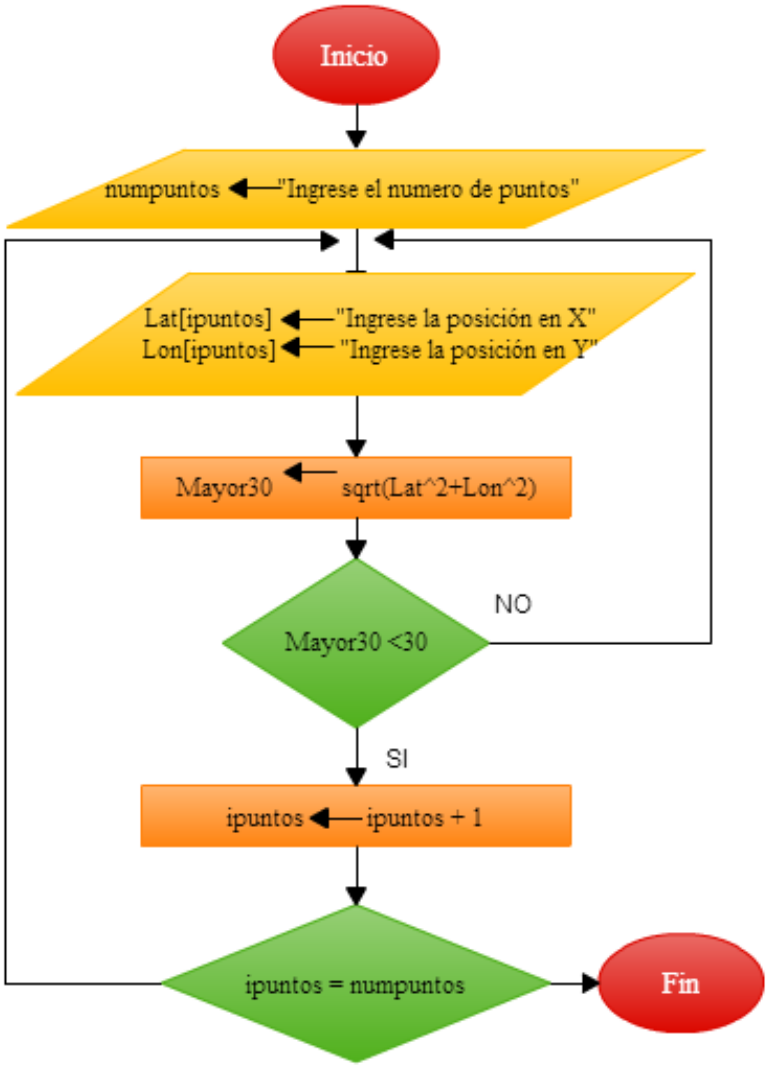

Figura 13. Diagrama de ingreso de puntos de ruta.

Una vez que ya están definidos los puntos válidos, se entra en un ciclo for, que recorre y realiza los cálculos de las variables para cada trayectoria entre punto y punto. Aquí toma mucha importancia todo lo mencionado anteriormente sobre el perfil de velocidad trapezoidal con el que se va a trabajar, debido a que para moverse, es necesario calcular todas las variables que se usarán luego, para fines de control y navegación. Las variables que se deben calcular y publicar son las siguientes: Los puntos iniciales para cada tramo del recorrido se almacenan en xo y yo. Los puntos finales de cada tramo, o el destino, se almacenan en xfinal y yfinal. El ángulo de rotación con respecto al origen que se debe realizar para orientar al *drone* hacia el punto de destino se almacena en theta. El ángulo de rotación con respecto al ángulo anterior que se debe realizar para orientar al *drone* hacia el punto de destino se almacena en thetanuevo. La distancia total del tramo se almacena en distotal. El tiempo total que se debe demorar el *drone* realizando el recorrido se almacena en T. El tiempo asignado a los tramos de aceleración y desaceleración se almacena en t. Los puntos intermedios del recorrido, que corresponden al final del tramo de aceleración y velocidad constante, se almacenan en px1 y py1 y en px2 y py2 respectivamente. Las distancias y tiempos de cada tramo (acelerado, constante, desacelerado) se almacenan en D1, D2, D3 y T1, T2, T3, respectivamente.

Antes de empezar el cálculo de las variables se deben definir tres parámetros fundamentales que le indican al programa cómo se desea que se comporte el *drone*. Estos parámetros son: pt, vk y el número n de puntos de muestreo. “pt” es un valor entre 0 y 1 que corresponde al porcentaje que se desea para la aceleración y

desaceleración de cada tramo. Cómo se explicó en el perfil de velocidad trapezoidal, este porcentaje define el porcentaje de tiempo que se emplea es cada segmento de la trayectoria. Por ejemplo, si se define pt=0.3, el tiempo total será empleado de la siguiente manera: 30% aceleración, 60% velocidad constante y 30% desaceleración. "vk" corresponde al valor de la velocidad máxima que se desea alcanzar en m/s. "muestreo" corresponde al número de puntos n que se van a muestrear en cada trayectoria.

Dentro del for se empieza por definir los valores de los puntos de inicio de cada tramo del recorrido. La posición inicial del *quadrotor* para el primer punto corresponde a la lectura de la posición UTM en la que se encuentra. Para los siguientes puntos basta con realizar la correspondencia de que, el punto inicial para el tramo siguiente, es el punto final del tramo anterior. Luego se hace el cálculo del ángulo de rotación. Para esto se realiza una comprobación para saber si la rotación se debe realizar hacia la izquierda o hacia la derecha. Luego, dependiendo del sentido de la rotación se encuentra el valor con la formula correspondiente.

Lo siguiente que se calcula es thetanuevo. Debido a que la rotación se hace con respecto al ángulo con respecto al origen, pero al llegar a cada punto de ruta, el *drone* queda orientado con respecto al ángulo trazado entre los puntos del tramo anterior, se debe corregir el cálculo del ángulo para que se tenga en cuenta esta desviación y la rotación sea efectiva.

Posteriormente se realiza el cálculo de la distancia total del tramo llamado distotal. Para esto, se realiza la norma del vector resultante a la resta del punto final y el punto inicial. Esta distancia está expresada en metros y debe hallarse para realizar otros cálculos posteriores. El tiempo total del tramo T se halla con la fórmula matemática de la velocidad, teniendo en cuenta los parámetros definidos anteriormente (vk y pt). El tiempo tao t que corresponde al tiempo empleado para el tramo de aceleración, se halla multiplicando el tiempo total T por el parámetro pt.

Para los puntos intermedios en términos de coordenadas cartesianas px1, py1, px2 y py2, se deben usar las fórmulas características del PVT. Estos puntos intermedios cobran mucha importancia para fines de comprobación y exactitud de la trayectoria, debido a que establecen una especie de *checkpoints* en la ruta para así saber la exactitud de la ejecución. De la misma manera, con sus respectivas ecuaciones se calculan las distancias D1, D2 y D3 y los tiempos T1, T2 y T3 que indican la distancia hasta cada punto intermedio y el tiempo que se debe demorar hasta estos puntos respectivamente.

Esta secuencia se realiza el mismo número de veces que puntos de ruta se hayan definido por el usuario. Cada iteración del ciclo, guarda todas las variables explicadas en la posición del vector correspondiente al punto de ruta actual. Esto se hace de esta manera para que la publicación de datos contenga todos los datos necesarios y no sea necesario publicar en muchas ocasiones o una vez por punto de ruta. Cuando ya se llenaron todos los datos dentro de los vectores correspondientes a las variables, se procede a crear y llenar el publicador de datos del nodo.

Para publicar, se usa una estructura de datos creada específicamente para el nodo, de tal manera que contenga todos los datos necesarios y con las longitudes y tipos de variables usadas en el nodo. Esta estructura se crea por medio de un archivo de mensaje personalizado (.msg) dentro de la carpeta "msg". En el archivo "Perfil_trap_data.msg" se definieron todas las variables como de tipo float64 para asegurar que la exactitud del programa pueda ser bastante alta. Una vez que se compila el paquete, se crea de manera automática el archivo "Perfil_trap_data.h" que corresponde al header de la estructura del mensaje, que se debe agregar a los includes de las librerías del nodo generador. Con este encabezado generado, dentro del archivo .cpp se genera un publicador de datos haciendo uso de la sintaxis de ROS. En esta línea de código se define el tipo de mensaje que se desea publicar y el tópico o tema al que se desea hacerlo. En este caso la estructura de datos se publica al tema "/*drone*_GNC/data".

Debido a que se trata de una estructura de datos con vectores, se debe realizar otro for para realizar el llenado de las variables de publicación. Luego de esto se pone la orden de ROS para que el nodo publique al tópico definido.

5.3. Algoritmo de movimiento y seguimiento de trayectoria.

Ya realizada la planeación y cálculo de las variables necesarias para realizar un perfil de velocidad trapezoidal, el siguiente paso es realizar todo lo relacionado con las otras dos partes de la arquitectura GNC. Estas dos partes siguientes son: Navigation, que se refiere al sensado/percepción del entorno espacial por parte del robot para poder garantizar el seguimiento de la trayectoria usando el GPS para el sensado de la posición espacial, sensor de presión+GPS para la estimación de alutra y la IMU para el sensado de componentes angulares de movimiento; y Control que permite el seguimiento de la trayectoria espacial de referencia con el mínimo error posible. Lo anterior significa que se debe realizar la lectura de los datos de los diferentes sensores, así como la ejecución de la trayectoria basado en las variables calculadas anteriormente, implementando un control que permita que la trayectoria se ejecute satisfactoriamente.

Para el trabajo de grado, se desarrollaron dos nodos que realizan la trayectoria calculada usando métodos diferentes para la ejecución de la misma. Cada uno tiene sus ventajas y desventajas dependiendo de lo que se quiera realizar y el tipo de trayectorias requeridas.

5.3.1. Método de rotación.

Este nodo llamado “PVT_Movimiento_Rotar” es algo más complejo que el anterior debido a que se deben realizar muchas más funciones de manera simultánea. Sus partes principales son: la inclusión de librerías y headers, las funciones de adquisición de datos y las etapas de movimiento y control.

La ejecución normal del programa consta de la iteración continua de las etapas de movimiento y de la adquisición de los datos a los que se debe suscribir. Debido a que en este algoritmo se debe ejecutar la trayectoria planeada, incluidos los tiempos de cada comando y orden de movimiento, lo que se hizo fue poner un retardo de T/muestreo por medio de contadores. Esto se realizó para que cada iteración del programa correspondiera al mismo tiempo de muestreo del PVT. Además se tuvo que ordenar una ejecución muy rápida del programa para que este tiempo se vuelva despreciable en comparación al tiempo usado para el muestreo.

Las librerías que se deben incluir en este nodo son básicamente las mismas que las que se usaron en el nodo generador, sin embargo, se deben adicionar un par de encabezados para poder publicar los comandos de movimiento y para realizar las suscripciones necesarias para el funcionamiento del algoritmo. De deben realizar tres suscripciones y tres publicadores diferentes.

La primera suscripción que se debe realizar es a los datos publicados por el nodo generador al tópico “/*drone*_GNC/data”, para poder tener acceso a todas las variables calculadas por el otro nodo. Estos datos son contenidos en la estructura que se mencionó anteriormente y que está incluida en el encabezado “Perfil_trap_data.h”. Luego, se debe crear una suscripción al tópico “/ar*drone*/predictedpose” que proviene del paquete tum_ar*drone* y contiene una serie de cálculos y estimaciones de las variables del *drone* tales como la posición x, y, z o los componentes angulares de roll, pitch, yaw. La estructura de estos datos está contenida en el encabezado “”filter_state.h”. Estos datos son de vital importancia para realizar el control. Finalmente, se realiza la suscripción al tópico “/ar*drone*/navdata” para tener acceso a todas las lecturas de los sensores de la IMU, directamente del paquete ar*drone*_autonomy, cuya estructura se incluye en el encabezado “navdata.h”. Para cada una de estas suscripciones se debe crear una función de tipo void, que se encarga de realizar las equivalencias entre las variables que son publicadas por los otros nodos en los tópicos mencionados, para asignarlas dentro de las variables que se usan en el algoritmo. Por último, también se debe crear la suscripción a los datos del GPS usando el tópico “/drone_GPS/data”.

Debido a que este nodo es el encargado de enviar los comandos de movimiento hacia el paquete ar*drone*_autonomy para que este se los envíe al *drone*, se deben crear tres publicadores diferentes, donde cada uno cumple una única función: despegar, mover y aterrizar. Por la configuración determinada del paquete ar*drone*_autonomy, los tópicos y tipos de mensajes que corresponden a estas tres acciones ya están definidas. Para enviar comandos de movimiento, se debe usar una estructura de tipo "geometry_msgs/Twist.h" y publicarla en el tópico “/cmd_vel”. En esta estructura de datos, se expresan los diferentes componentes de velocidad en sus partes angulares y lineales (x, y, z). Para enviar las órdenes de

despegue y aterrizaje se usa la misma estructura de mensaje, que es vacío y se define como "std_msgs/Empty.h". Para realizar el despegue este mensaje se publica al tópico "/ar*drone*/takeoff" y para el aterrizaje se publica al tópico "/ar*drone*/land". Con estos tres publicadores creados, para generar un movimiento en el UAV, se deben llenar las estructuras con los valores deseados y publicar cada vez que sea necesario.

Adicionalmente, es necesario explicar con un poco más de detalle cómo se llena la estructura de movimiento. Se crea una variable de tipo geometry_msgs::Twist llamada twist. En esta variable se guardaran los comandos a realizar. Como se explicó, esta estructura tiene los componentes de velocidad angular y lineal. Estos se acceden de la siguiente manera: twist.linear.x para la componente x, twist.linear.y para la componente y, twist.linear.z para la componente z y twist.angular.z para la rotación. Las componentes twist.angular.x y twist.angular.y no se usan para nada y deben permanecer en 0.0. Cada uno de los componentes puede tomar un valor entre -1.0 y 1.0, con cualquier valor intermedio deseado, siendo estos dos, los valores máximos posibles. Esto significa que para ir hacia adelante y atrás se establece twist.linear.x en un valor positivo y negativo respectivamente. Para ir a la izquierda y a la derecha se establece twist.linear.y en un valor positivo y negativo respectivamente. Para subir y bajar se establece twist.linear.z en un valor positivo y negativo respectivamente. Para rotar hacia la derecha y hacia la izquierda se establece twist.angular.z en un valor positivo y negativo respectivamente.

Para realizar la ejecución del movimiento del perfil de velocidad trapezoidal, se implementó una estructura que discrimina cada etapa del punto que se está recorriendo. Esto con el fin de organizar el algoritmo y saber en qué punto del recorrido de encuentra el *drone*. Las etapas definidas son las siguientes: 0) Despegue, 1) Rotar, 2) Acelerar, 3) Velocidad constante, 4) Desaceleración, 5) Etapa de vuelo estacionario, 6) Transición entre punto y punto. Cada una de estas etapas será explicada con más detalle a continuación.

La etapa cero, correspondiente al despegue del AR *Drone* 2.0 se encarga de publicar el comando de despegue hacia el paquete ar*drone*_autonomy. Además, se llena la estructura de movimiento con ceros, para que el *drone* se quede en modo de vuelo estacionario mientras realiza el despegue hasta la altura definida.

La etapa de rotación corresponde a la número 1. En esta parte, lo que se hace es mandar un comando de rotación al *drone* dependiendo del valor de theta calculado en el nodo generador. El *drone* debe girar sobre su propio eje y orientarse de tal manera que el frente quede orientado hacia el punto que debe alcanzar. Para esto, se realiza una lectura del dato "rotz" (la rotación en Yaw del *drone*), publicado en el tópico /ar*drone*/navdata. Por medio de esta lectura se verifica el momento en el que ya se entró a un rango admitido en el que el ángulo de la aeronave es muy similar al calculado. Una vez se alcanza este rango admitido se pasa a la siguiente etapa: Acelerar.

Debido a que el *quadrotor* ya se encuentra mirando hacia el punto de destino, el único movimiento que se debe realizar es hacia adelante. Para esto se modifica la variable twist.linear.x en la estructura de publicación de movimiento. Adicionalmente, se hace que para cada iteración del programa, se realice la aceleración en pasos que están determinados por la velocidad máxima vk y por el punto actual de muestreo, al llegar al punto correspondiente a la etapa de velocidad constante se pasa a la siguiente etapa. Esto se observa en la ecuación, debido a que en el momento en el que iacelerado=0, entonces la velocidad en ese instante será vk/k, y en el momento en el que se acaba la etapa de aceleración, iacelerado=k-1, entonces la velocidad será igual a vk, o la velocidad máxima definida para la trayectoria. Con esto se garantiza un aumento progresivo de velocidad.

$$twist.linear.x = \left(\frac{vk}{k} * (i_{acelerado} + 1.0)\right)$$

Para la etapa de velocidad constante, basta con publicar en la variable twist.linear.x, el valor de velocidad correspondiente vk e iterar el programa durante el número de puntos de muestreo definidos para este tramo. La última de etapa de movimiento, correspondiente a la desaceleración del *quadrotor*, realiza un procedimiento similar al de la etapa de aceleración pero de la manera inversa. En esta parte, se pasa de la velocidad máxima, vk, a un valor de velocidad de 0.0. Esto se realiza con la ecuación siguiente, en donde

se puede observar que para la primera iteración, el valor de velocidad que se le ingresa al *drone* es de: vk-vk/k, lo que corresponde al comienzo de la desaceleración, y en el momento en el que la variable ibajada=k-1, la velocidad será igual a cero. Lo que hace que se forme un escalón de disminución de velocidad.

$$twist.linear.x = \left(vk - \left(\frac{vk}{k} * (i_{acelerado} + 1.0)\right)\right)$$

Una vez se termina el perfil de velocidad trapezoidal para el punto actual, se publica nuevamente la estructura de movimiento pero con todos sus valores en cero para que la aeronave realice un vuelo estacionario antes de pasar al siguiente punto. Aquí se reinician todas las variables, se pasa a evaluar el perfil del siguiente punto de ruta, y se pasa a la etapa 1 nuevamente para empezar la nueva trayectoria. Una vez de recorren todos los puntos de ruta definidos por el usuario, el *drone* aterriza y el programa acaba.

Ya se explicó la estructura de las etapas y cómo se realiza la iteración de las mismas en el algoritmo desarrollado, sin embargo es necesario explicar el control que se realiza en cada una de estas etapas. Se implementó un control de ángulo que se realiza en todas las etapas de movimiento (aceleración, velocidad constante y desaceleración). Este control consta de un algoritmo que lee la posición actual del *quadrotor* y con base en esta posición calcula el ángulo que existe entre el punto actual y el punto de destino. De esta manera, si el *quadrotor* se encuentra en la trayectoria ideal, el ángulo entre los puntos debería ser cero. Dependiendo del resultado de este cálculo, se ingresa un comando de rotación en la estructura de publicación de velocidades en twist.angular.z y hace que el *drone* gire hacia izquierda o derecha sobre su propio eje. Al realizar este cálculo de manera constante en cada iteración del programa, el *drone* se mantiene en la ruta ideal, corrigiendo el ángulo a cada momento. Es necesario aclarar que como se requiere que esta rotación se realice al tiempo con el movimiento hacia adelante, entonces el comando de rotación enviado al *drone* es muy bajo, para que la rotación sea suave y no interfiera con el movimiento normal.

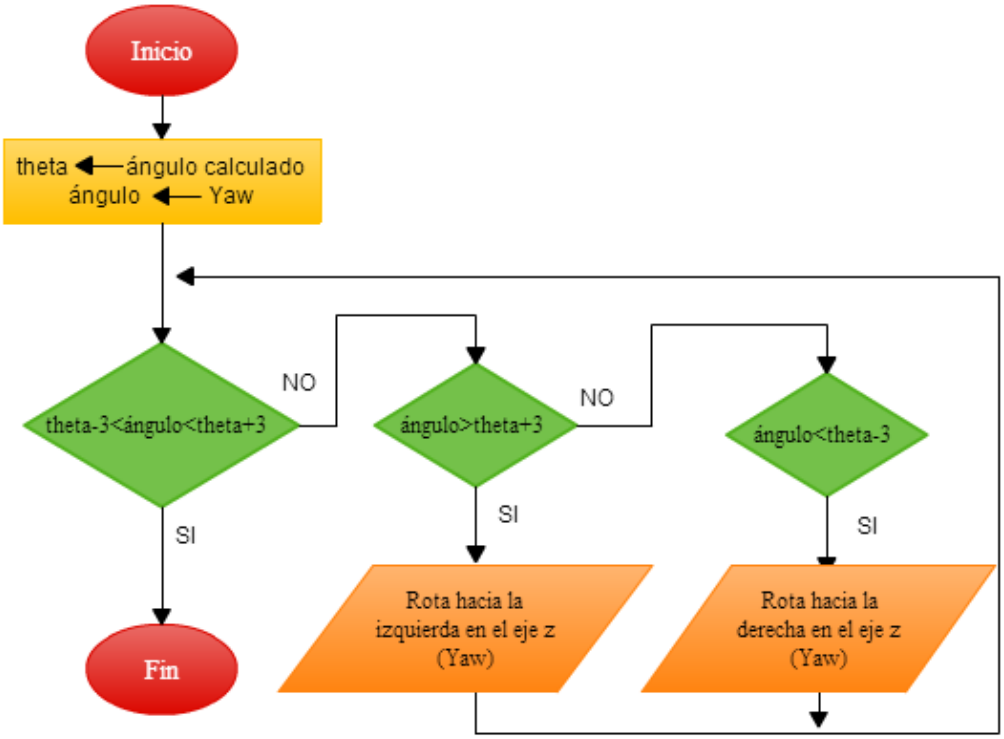

Figura 14. Diagrama del control de ángulo.

5.3.2. Método de vectores de velocidad.

Este nodo es llamado "PVT_Movimiento_Vectores" y se encarga de realizar la trayectoria planeada en el nodo generador por medio del cálculo del vector de la trayectoria que será explicado a continuación.

La principal diferencia entre este método y el anterior, es que en el nodo de "PVT_Movimiento_Rotar", se ejecuta la trayectoria realizando un giro que haga que el *drone* quede orientado hacia el punto de destino, con el fin de moverse en una sola dirección (adelante). Para el nodo de vectores, no se realiza ninguna rotación, el *drone* permanece orientado con un ángulo de cero grados en toda la ejecución de la trayectoria. Esto implica que el movimiento se debe realizar en ambos ejes (x, y) y que el norte coordenado del *drone* va a coincidir en todo momento con el norte del plano UTM.

La estructura general de este algoritmo es bastante similar a la del nodo de movimiento ya explicado, debido a que se ejecuta por etapas y tiene las mismas suscripciones y los mismos publicadores de datos.

El cambio en la forma de ejecutar el movimiento en este nodo se debe a que en este caso, se creó una función llamada "vectores_de_movimiento" que usa los datos del perfil de velocidad trapezoidal para calcular un arreglo de dos dimensiones que contiene los cálculos de posición en x, posición en y, velocidad en x, velocidad en y, aceleración en x y aceleración en y, para cada punto de muestreo de la trayectoria. Adicionalmente se incluye el vector de tiempo estimado. Los valores de entrada corresponden a las variables que se publicaron en el tópico /*drone*_GNC/data y a los cuales se crea un suscriptor en el nodo. Con estas variables y usando las ecuaciones descritas en la documentación acerca del Perfil de velocidad trapezoidal, se puede llenar el arreglo de dos dimensiones en donde se almacena toda la información relevante del movimiento. Es importante aclarar que debido a que las ecuaciones de posición, velocidad y aceleración para cada tramo de la trayectoria (acelerado, constante y desacelerado) son distintos, se debe realizar un ciclo para cada tramo por separado, con sus respectivas ecuaciones, sin embargo, siempre se llena la variable de trayectoria, continuando donde acabó el tramo anterior.

En la variable "traj" que tiene un número de filas igual al número de puntos de muestreo que se hayan definido para cada trayectoria, que normalmente es de 30 puntos por trayectoria, tiene un número de columnas igual a 7, donde cada columna corresponde a uno de los parámetros calculados. Estos parámetros almacenados en cada columna son: traj[muestreo][0] corresponde al vector de posición en x, traj[muestreo][1] corresponde al vector de posición en y, traj[muestreo][2] corresponde al vector de velocidad en x, traj[muestreo][3] corresponde al vector de velocidad en y, traj[muestreo][4] corresponde al vector de aceleración en x, traj[muestreo][5] corresponde al vector de aceleración en y. Finalmente, traj[muestreo][6] corresponde al vector de tiempos para cada instante.

Como se dijo que este nodo tiene una estructura por etapas al igual que el nodo de movimiento por rotación. La iteración del nodo es constante y se realiza a un lazo de repetición de 40 ciclos por segundo de tal manera que las publicaciones tengan un tiempo mucho más exacto y el movimiento sea más estable. Lo que se hace dentro del algoritmo es que se implementan contadores auxiliares que se incrementan en cada iteración del programa, de tal manera que se realiza el cálculo de a qué valor deben llegar estos contadores para realizar una espera de determinado tiempo. Luego se ingresa a las diferentes etapas que son básicamente las mismas y se organizan de la siguiente manera: 0) Despegue, 1) Orientación, 2) Movimiento, 3) Hover, 4) Cambio de punto.

Las etapas usadas en este algoritmo son similares a las implementadas en el algoritmo de rotación, sin embargo hay algunas diferencias con respecto a la ejecución del programa. La etapa 0, corresponde al despegue de la aeronave. En esta parte del programa se publica el mensaje de despegue junto con un comando de estabilización de tal manera que al despegar, se mantenga en el mismo punto. La etapa 1 es usada para la rotación y estabilización del *drone*. Para el método usado, debido a que los comandos de velocidad que se envían, son en los componentes x, y, se debe mantener el vehículo orientado hacia el norte, o con un ángulo de cero grados para que el movimiento se realice de manera efectiva. Teniendo esto en

cuenta, en la etapa actual se realiza una rotación hasta que se logra llegar a un valor en el ángulo de rotación con respecto a z que sea muy cercano a cero y se pasa a la siguiente.

La etapa 2 cubre todo lo relacionado con la ejecución del movimiento. En esta se hace el llamado a la función "vectores_de_movimiento" que realiza el cálculo de las velocidades y las posiciones que serán usadas en la trayectoria actual. El objetivo de estos vectores es que se puedan mandar los comandos de velocidad, simplemente recorriendo el arreglo, donde cada posición corresponde a la velocidad o posición que debería tener el *drone* en determinado instante para realizar la trayectoria ordenada de manera satisfactoria. Esto se hace usando dos contadores, el primero se encarga de iterar de manera constante hasta que se alcanza el tiempo determinado por el muestreo. Este conteo corresponde al tiempo durante el que debería permanecer la publicación de cada comando de velocidad. Al alcanzar el valor correspondiente, se incrementa la posición del vector de velocidad y se reinicia el contador de tiempo. Este proceso se realiza hasta recorrer el vector completo, cuya longitud está dada por el número de puntos de muestreo determinados. Con este proceso se asegura que el *drone* reciba el comando correspondiente, durante el tiempo correspondiente, de tal manera que la trayectoria planeada pueda ser ejecutada.

De manera paralela a este proceso de movimiento, se está realizando el control implementado. Este consta de un simple control proporcional. Lo que se hace es recorrer el vector de posiciones calculado (al igual que se recorre el vector de velocidad) y compararlo con la posición actual del *drone*. Haciendo esto en cada iteración del programa, se va a obtener el error de posición actual para cada instante de la trayectoria. Luego se multiplica este error por la constante Kp, que se encarga de escalarlo en términos de los comandos de velocidad que se deben mandar. Esta constante debe ser de un valor muy pequeño debido a que la velocidad del *quadrotor* es bastante elevada. De esta manera, se obtiene el error, se multiplica por la constante Kp y este valor se le suma a la velocidad que debería tener el *drone* en ese instante. Realizando este proceso en cada iteración del programa, se puede corregir progresivamente la ubicación, logrando reducir el error y llegando al punto deseado.

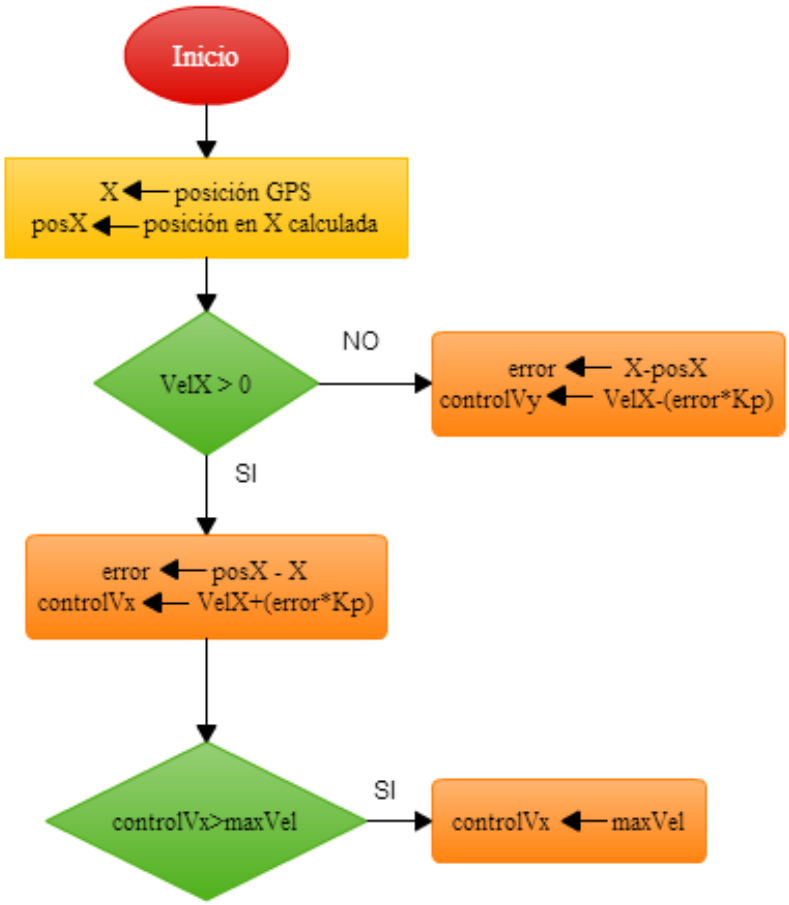

Figura 15. Diagrama del control proporcional.

Luego de recorrer todo el vector de velocidades, para cada punto de muestreo, se finaliza el movimiento y se pasa a una etapa de vuelo estacionario en la que se espera unos pocos segundos y se pasa a la etapa final. Aquí, se realiza la inicialización de las variables y se cambia el punto de trayectoria actual, para pasar a recorrer el siguiente punto de ruta determinado por el usuario y se pasa a la etapa de movimiento nuevamente. Al finalizar el recorrido de todos los puntos de ruta determinados por el usuario, se realiza una comprobación especial y se envía el comando de aterrizaje al *drone*.

5.4. Diferencias entre los archivos de simulación y pruebas reales

Debido a que el proyecto desarrollado se basa en el uso del GPS para la navegación y la adquisición de los puntos de ruta y la posición actual del *quadrotor*, se deben discriminar los archivos para las simulaciones y los archivos finales del paquete desarrollado. En el simulador utilizado, no hay manera de usar el GPS para realizar estas pruebas, por lo que para poder verificar el funcionamiento de los diferentes nodos en el paquete "tum_simulator", lo que se hizo fue usar los datos del nodo "*drone*_statestimation" incluido en el paquete "tum_ar*drone*", que realiza una estimación de la posición actual del *quadrotor* y permite simular un comportamiento de seguimiento de puntos de ruta. Esto es muy válido para las simulaciones y para realizar las pruebas preliminares, sin embargo a la hora de probar el código en el mundo real, se debieron realizar unas modificaciones en los algoritmos muy pequeñas.

Para empezar, se debe trabajar con el paquete "*drone*_GPS" que se encarga de obtener las lecturas del GPS del AR *Drone* 2.0 y además realiza la conversión de estas coordenadas GPS (latitud, longitud) a las

coordenadas UTM (X, Y) que se usan para realizar la navegación por medio de punto de ruta. Para esto, se debe incluir en los nodos usados, el encabezado de la estructura del mensaje publicado por este paquete, adicionando la siguiente línea de código: #include "*drone*_GPS/GPS_data.h". Además, se debe realizar la suscripción al tópico "/*drone*_GPS/data" para realizar la lectura de los datos enviados.

Una vez se añadieron el encabezado y la suscripción a los nodos utilizados, es necesario cambiar unas líneas en el nodo generador y de movimiento. En el nodo "PVT_Generador" se debe cambiar el punto inicial de la trayectoria, se tal manera que no sea cero, sino en cambio, la posición actual del *quadrotor* antes de despegar. En los nodos de movimiento "PVT_Movimiento_Rotar" y "PVT_Movimiento_Vectores" se debe cambiar la forma de encontrar el error actual de la trayectoria. Se debe realizar el cálculo no con las variables del estimador de estado sino con las lecturas de X y Y de las coordenadas UTM.

6. ANÁLISIS DE RESULTADOS

En este capítulo se presentan todos los resultados obtenidos con los algoritmos desarrollados, incluyendo las simulaciones realizadas, los experimentos en terreno y los diferentes protocolos de prueba y criterios de evaluación usados para verificar la validez del paquete.

6.1. Simulaciones

6.1.1. Simulación de la planta del *quadrotor* usando Simulink

Se realizó la simulación del control de bajo nivel del modelo dinámico del quadrotor con el propósito específico de aprender sobre el funcionamiento de este. Las simulaciones fueron realizadas en Simulink, haciendo uso de un modelo previamente desarrollado, donde se incluye toda la planta del quadrotor. Este modelo, permite calcular las diferentes variables que se encargan de controlar el movimiento y la estabilidad del robot.

El sistema dinámico y cinemática del quadrotor cuenta con seis grados de libertad (tres de translación y tres de rotación) y con únicamente cuatro entradas independientes (las fuerzas de los rotores). Las ecuaciones iníciales del modelo competen una complejidad mayor a la que este trabajo de grado abarca, por lo que las ecuaciones que se presentarán a continuación representan el modelo simplificado, con el que se realizará el modelo. Para llegar al modelo simplificado se hacen una serie de aproximaciones, como lo es despreciar los términos de Coriolis[26] y suponer que los ángulos φ y θ son muy pequeños.

La implementación de este modelo se muestra en la figura 17 y está basado en las ecuaciones diferenciales simplificadas del modelo inercial del quadrotor [17] y son:

$$\ddot{x} = -\cos\varphi \sin\theta \frac{U_1}{m} \quad (1) \qquad \ddot{\varphi} = \frac{l}{J_x} U_2 \quad (4)$$

$$\ddot{y} = \sin\varphi \frac{U_1}{m} \quad (2) \qquad \ddot{\theta} = \frac{l}{J_y} U_3 \quad (5)$$

$$\ddot{z} = g - \cos\varphi \cos\theta \frac{U_1}{m} \quad (3) \qquad \ddot{\psi} = \frac{l}{J_z} U_4 \quad (6)$$

Estas ecuaciones están dadas para el sistema inercial. En la tabla 2 se describe lo que representa cada una de las variables en las ecuaciones diferenciales y en la tabla 3 se presenta la descripción y los valores asignados para esta aeronave en específico.

Variable	Descripción
$x, \dot{x}, \ddot{x}$	Posición, velocidad, aceleración en x.
$y, \dot{y}, \ddot{y}$	Posición, velocidad, aceleración en y.
$z, \dot{z}, \ddot{z}$	Posición, velocidad, aceleración en z.
$\varphi, \dot{\varphi}, \ddot{\varphi}$	Ángulo de rotación, velocidad y aceleración angular con respecto a x (Pitch).
$\theta, \dot{\theta}, \ddot{\theta}$	Ángulo de rotación, velocidad y aceleración angular con respecto a y (Roll).
$\psi, \dot{\psi}, \ddot{\psi}$	Ángulo de rotación, velocidad y aceleración angular con respecto a z (Yaw).
U_1	Empuje total de los rotores.
U_2, U_3, U_4	Fuerza para realizar movimiento roll, pitch y yaw.

Tabla 2. Variables del modelo dinámico.

Descripción	Notación	Valor
Constante gravitacional	g	$9{,}81\left[{}^{m}/_{s^2}\right]$
Masa de la aeronave	m	$0.420[kg]$
Momento de inercia rotacional en el eje x.	J_x	$0.0075\ [Kg.m2]$
Momento de inercia rotacional en el eje y.	J_y	$0.0075\ [Kg.m2]$
Momento de inercia rotacional en el eje z.	J_z	$0.0075\ [Kg.m2]$
Longitud del centro del motor al eje del rotor.	l	$0.2[m]$

Tabla 3. Constantes del modelo dinámico para el AR Drone 2.0

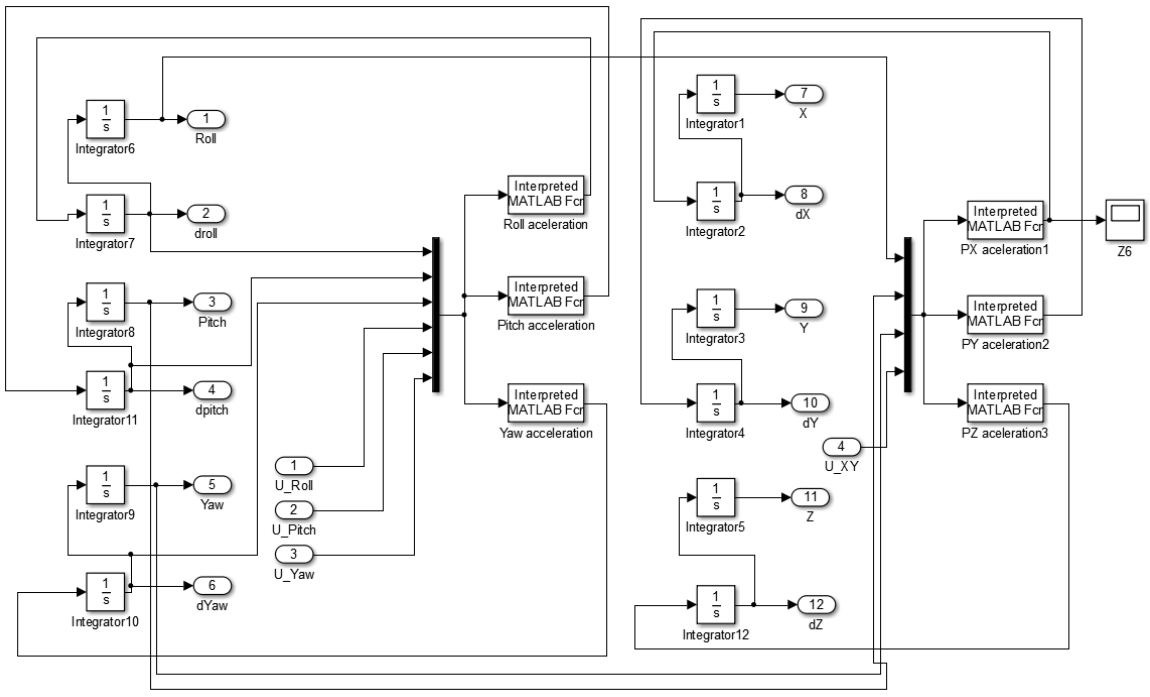

Figura 16. Modelo de la planta del *quadrotor* en Simulink.

Este sistema dinámico se divide en dos subsistemas que están interconectados, como se puede observar en la figura 18. El primer subsistema es el de rotación, cuyas entradas son los momentos de fuerza de los rotores y sus salidas son los tres ángulos de viraje en los que se puede mover el *drone* (Roll, Pitch, Yaw). El segundo subsistema es el de translación, el cual tiene como entradas las salidas del subsistema anterior además de la fuerza de empuje de los rotores y como salida la posición en X, Y y Z.

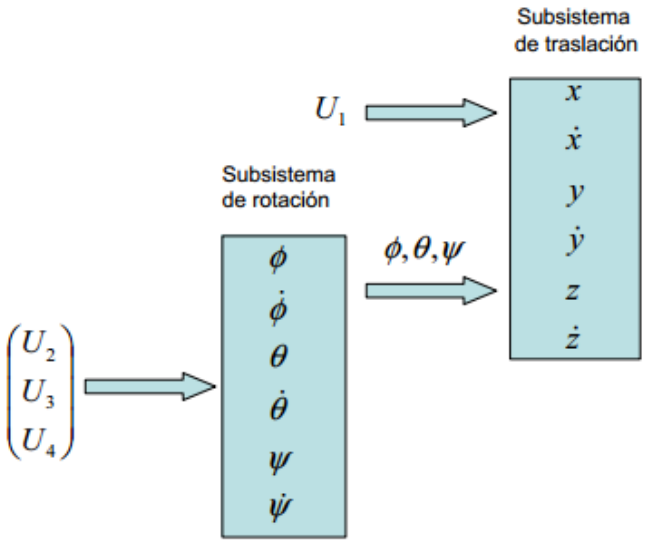

Figura 17. Sistema dinámico dividido en dos subsistemas interconectados Tomada de [30].

Para controlar el modelo mostrado en Simulink, se usó un controlador PID donde las contantes proporcional, integral y derivativa, fueron encontradas por medio del método de despeje matemático por medio del polinomio característico. Primero se hallaron las funciones de transferencia de las ecuaciones 4, 5 y 6 ya que estas representan los tres ángulos que controlan el movimiento. Además es necesario encontrar una cuarta función de transferencia, la cual representa la fuerza de empuje mostrada como U1 en la figura 18. Luego de realizar esto, se encontraron las diferentes constante que se muestran en la tabla 3.

El procedimiento de la sintonización del PID se debe realizar para cada una de las cuatro variables de entrada. A continuación se muestra el procedimiento para la entrada U_2. El procedimiento para calcular las variables restantes se realiza de forma análoga.

La función de transferencia en lazo cerrado para el ángulo planteado es:

$$M(s) = \frac{G(s) * H(s)}{1 + G(s) * H(s)}$$

Donde la planta $\big(G(s)\big)$ y el control $\big(H(s)\big)$ son:

$$G(s) = \frac{\varphi(s)}{U_2(s)} = \frac{l}{I_{yy} * s^2} = \frac{b_1}{s^2} \qquad H(s) = \frac{K_p * s + K_i + K_d * s^2}{s}$$

Por lo tanto

$$M(s) = \frac{b_1 * K_p * s + b_1 * K_i + b_1 * K_d * s^2}{s^3 + b_1 * K_p * s + b_1 * K_i + b_1 * K_d * s^2}$$

Para la sintonización del controlador PID se define un tiempo de estabilización de 3 segundos y un sobre pico del 10%, es decir $t_s = 3\ s$ y $SO = 0.1$. Con estos datos y con la ayuda de Matlab se halla el coeficiente de amortiguamiento y la frecuencia $\xi = 0.63$, $\omega_n = {}^{4}/_{(\xi * \mathrm{t_s})} = 2.13$. El polinomio característico para la sintonización es el siguiente:

$$(s^2 + 2 * \xi * \omega_\mathrm{n} * s + {\omega_\mathrm{n}}^2)(s + \alpha)=0$$

$$(s^2 + 2.68 * s + 4.54)(s + \alpha)=0$$

Se selecciona un nuevo polo α que se encuentra diez veces antes que los polos del sistema, es decir $\alpha = -13.33$. Utilizando los datos de las tabla 3 se hallan los valores de las constantes proporcionales, integrales y derivativas para cada una de las 4 fuerzas de entrada del sistema.

	Kp	Kd	Ki
Roll (U1)	1.50	0.60	2.26
Pitch (U2)	1.50	0.60	2.26
Yaw (U3)	0.52	0.21	0.78
Empuje (U4)	-16.11	-6.43	-24.31

Tabla 4. Valores de las constantes encontradas para los controlados PID.

Con el modelo del *quadrotor* mostrado y con las constantes halladas, se procedió a desarrollar un lazo de control en Simulink para las entradas mostradas. El resultado de esto se puede observar en la figura 19.

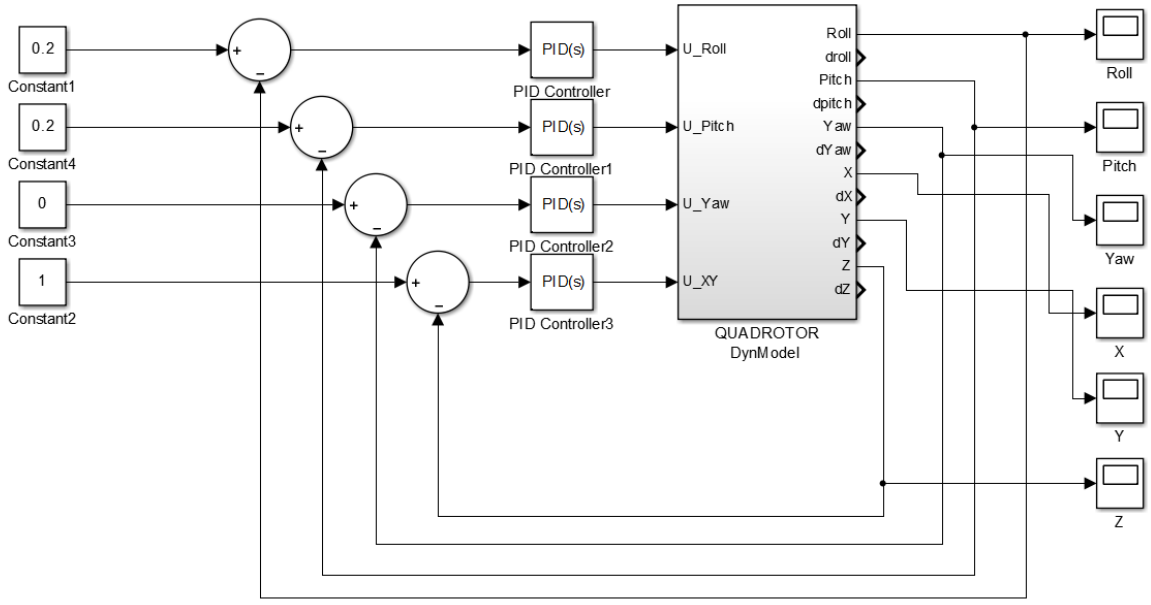

Figura 18. Modelo de control de lazo cerrado en Simulink.

Para realizar la simulación se pusieron valores en las variables a controlar, donde se quería que Roll llegara a 0.1 radianes, Pitch a 0.2 radianes y Yaw a 0.3 radianes. Esto significa que se simula un movimiento en los tres ejes de rotación del *quadrotor*.

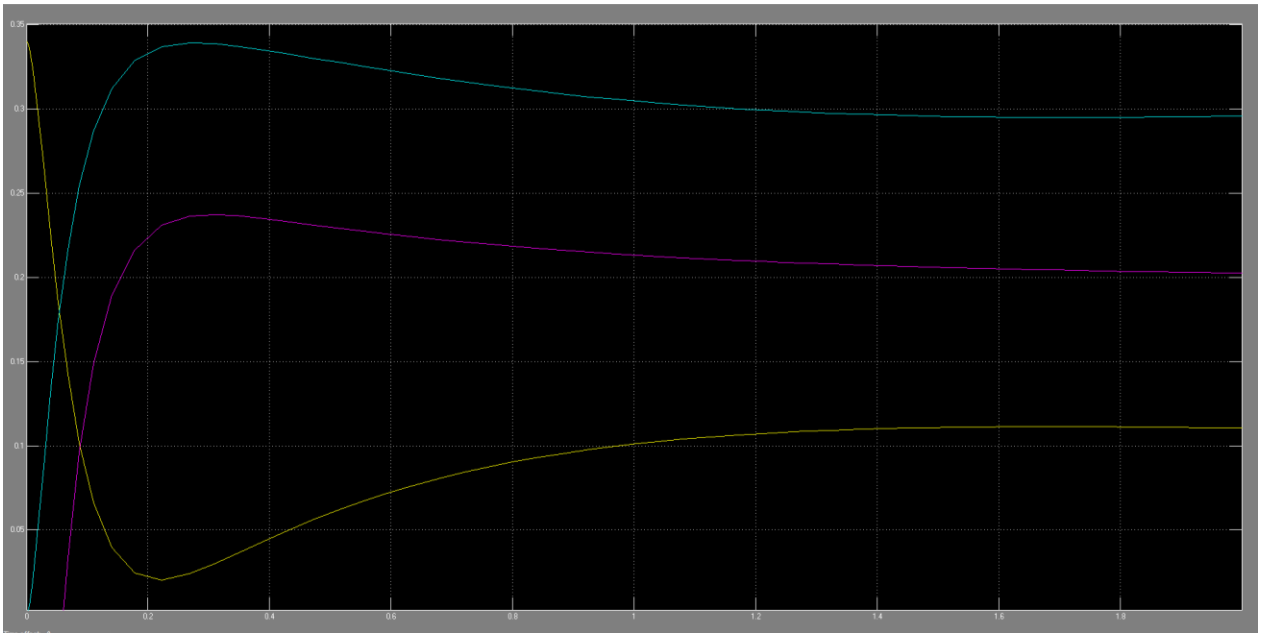

Figura 19. Simulación del modelo del *quadrotor* usando Simulink. Las curvas amarilla, morada y cian, corresponden a las variables Roll, Pitch, Yaw, respectivamente.

El objetivo de esta sección es sólo para mostrar leyes teóricas de control PID y algunas pruebas de simulación inicial de este enfoque. Si se quiere indagar más en este campo de control lineal y no lineal (Backstepping) se pueden revisar las bibliografías.

Debido a que el paquete usado en ROS llamado "ar*drone*_autonomy" ya cuenta con el control de bajo nivel del *quadrotor*, que permite controlar el robot con el envío de comandos de velocidad en los 3 ejes de movimiento (X, Y, Z) y en el eje angular (Z), este trabajo se enfocó en el desarrollo de un control proporcional controlando las velocidades de movimiento del *drone*. Este control se realizó por medio de la comparación de la trayectoria de referencia calculada y la posición real del *drone* (obtenida por medio del sensor GPS) para cada instante de tiempo. Posteriormente, se multiplica el error por la constante proporcional, para luego ser sumada a la velocidad calculada y así realizar una corrección del movimiento. La constante kp utilizada, se determinó usando calibración manual basado en el error promedio entregado por el GPS. Estos valores de kp oscilan entre 0.01 y 0.03. Para observar los resultados, remítase a la sección de pruebas con el método de "PVT_Movimiento_Vectores". En la figura 21 se muestra un esquema del lazo de control desarrollado en los nodos de movimiento.

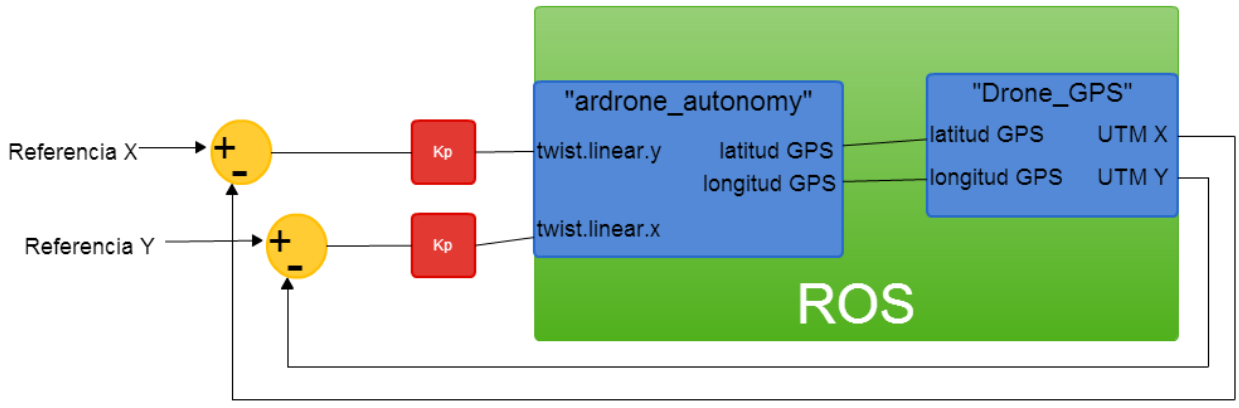

Figura 20. Esquema del lazo de control implementado en los nodos de movimiento.

6.1.2. Algoritmo de cálculo del Perfil de Velocidad Trapezoidal

Para realizar una simulación precisa de los datos que son calculados por el nodo generador o de cálculo, se utilizó Matlab. De esta manera, lo que se hizo fue darle un punto inicial y un punto final al algoritmo para que este realizara los cálculos. Las variables que calcula el algoritmo, son las mismas que se encuentran en el nodo "PVT_Generador" y además se incluye el cálculo de los vectores de movimiento incluidos dentro de la función "vectores_de_movimiento". En estas se incluyen: el tiempo total de la trayectoria, los tiempos y distancias de los puntos intermedios, la distancia total, y los vectores de posición, velocidad y aceleración en cada instante. Debido a que el proyecto actual solo debe moverse en dos dimensiones (x, y), la simulación se realizó de la misma manera.

En la simulación realizada, se fijó como punto inicial la posición [1, 2, 0] y se fijó el destino en [6, 10, 0]. En la figura 22 se observa la trayectoria realizada en cada uno de los puntos de muestreo, y se debe resaltar cómo en el tramo de velocidad constante, los puntos de muestreo se encuentran más separados debido a la mayor velocidad del movimiento, mientras que en los tramos de aceleración y desaceleración, los puntos se encuentran mucho más juntos. Esto se explica mucho más fácil al observar las gráficas de los distintos componentes de velocidad mostrados en la figura 23. En estos se observa claramente cómo la velocidad en cada uno de sus componentes forma un perfil de velocidad trapezoidal, donde el tiempo de aceleración y desaceleración está definido por una constante pt. En este caso, se define que pt sea igual a 0.3, lo que hace que el tramo acelerado corresponda a un 30% del tiempo total, el tramo constante a un 40% y el tramo desacelerado a otro 30%.

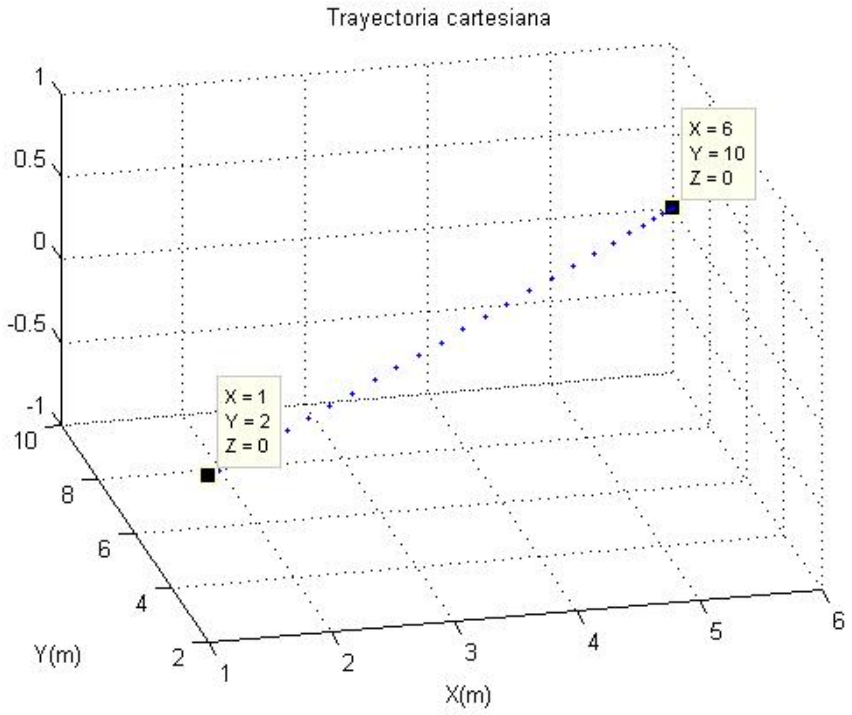

Figura 21. Trayectoria cartesiana de vuelo entre los puntos [1, 2, 0] y [6, 10, 0].

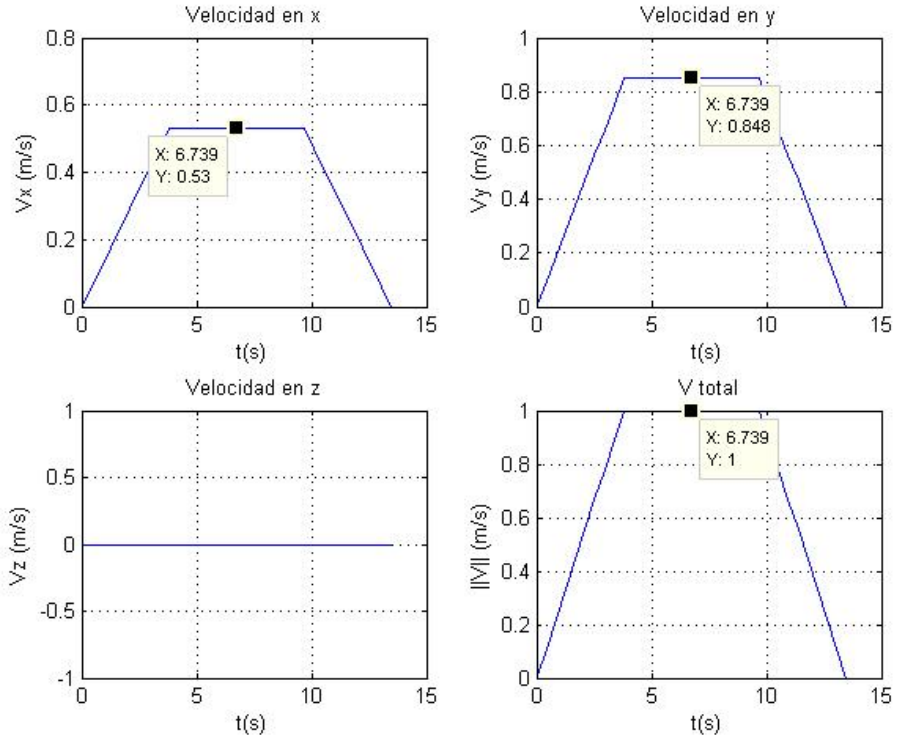

Figura 22. Gráficas de velocidad para la trayectoria realizada.

Adicionalmente, cabe resaltar que debido a que en la coordenada x se debe recorrer una distancia (5) menor que en la coordenada y (8), la velocidad alcanzada en esta última es más grande, ya que en el mismo tiempo debe recorrer una distancia mayor. Sin embargo, al realizar la norma de ambos componentes de la velocidad, el resultado en el tramo constante es vk (en este caso ajustado a un valor de 1 m/s).

En la tabla 4 se pueden observar los datos que se calculan y posteriormente se publican en el tópico /*drone*_GNC/data. De aquí se extraen todos los datos para ser usados en el nodo de movimiento. En este caso, los datos calculados corresponden a la trayectoria mostrada en la figura 22.

Xo	1.0
Yo	2.0
Xfinal	6.0
Yfinal	10.0
Theta	32.0°
Distancia Total	9.43 m
Tiempo Total	13.47 s
Tiempo Tao	4.04 s
Punto intermedio 1	[2.07, 3.71]
Punto intermedio 2	[4.92, 8.28]

Tiempo tramo 1	4.04 s
Tiempo tramo 2	5.39 s
Tiempo tramo 3	4.04 s
Distancia tramo 1	2.02 m
Distancia tramo 2	5.39 m
Distancia tramo 3	2.02 m

Tabla 5. Valores de publicación del nodo generador.

6.1.3. Simulación de los algoritmos de movimiento

Para realizar una comprobación de los dos métodos de movimiento empleados, antes de probarlos en un entorno real, se utilizó el paquete de ROS llamado "tum_simulator" que permite la simulación del AR *Drone* 2.0 con sus respectivos parámetros y comandos de vuelo de una manera precisa. Para cada uno de estos, se realizaron un set de 10 pruebas en donde se determinó el error general de la posición alcanzada vs la deseada. Estas simulaciones se hicieron con la ayuda también del paquete "tum_ar*drone*" que realiza una estimación de los diferentes parámetros del *drone*, entre estos, su posición x, y. Esto se hizo debido a que en el entorno de simulación no se tenía manera de probar con el GPS y por lo tanto la lectura de la posición para el control se relizó por este medio.

6.1.3.1. Simulación del nodo "PVT_Movimiento_Vectores"

Para las simulaciones realizadas, se buscó que la trayectoria realizada, fuera muy similar a la que se hiciera en la realidad, esto para poder realizar comparaciones entre los métodos y los posibles errores que se presentaron. La figura 24 muestra el resultado de una de las simulaciones a la trayectoria propuesta. Para esta prueba el parámetro vk fue de 0.4 y la constante kp fue de 0.3. Al igual que esta, se realizaron muchas pruebas que se pueden encontrar en los anexos del proyecto, que por motivos de brevedad y pertinencia no se incluyeron en el documento.

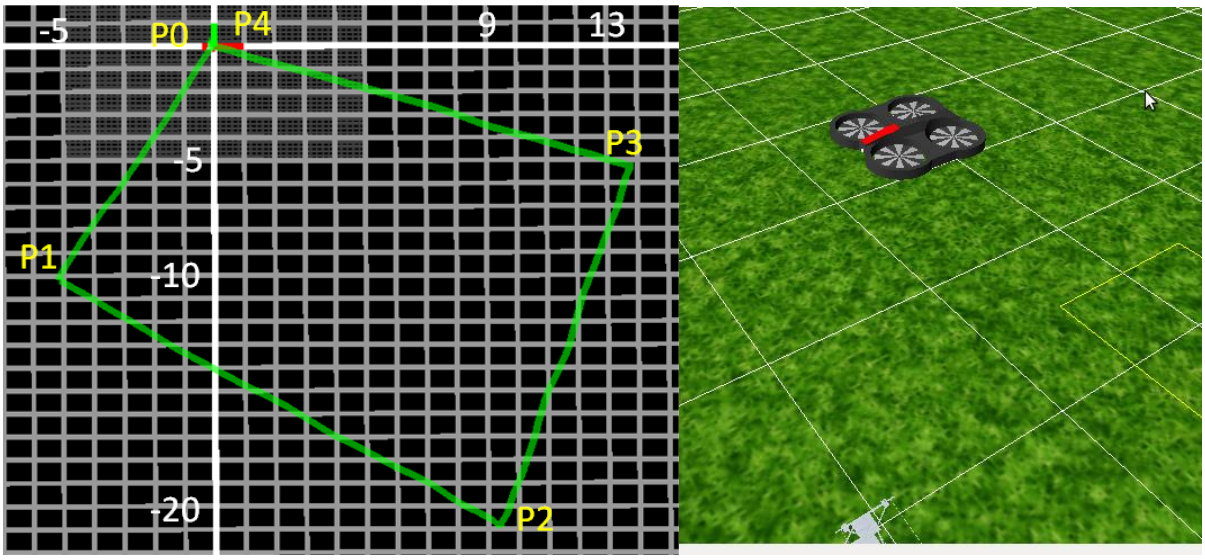

Figura 23. Resultado de la simulación para el nodo "PVT_Movimiento_Vectores".

La tabla 5 muestra las coordenadas alcanzadas, así como las calculadas y el error en cada momento. Se puede observar que la trayectoria en todo momento resulta muy precisa y que los mayores errores se dan al finalizar la etapa de velocidad constante, donde luego, en la etapa de desaceleración, los errores se corrigen casi hasta llegar a cero. Esto sucede debido a que en las etapas de aceleración y desaceleración, la velocidad es de pequeña magnitud, entonces la multiplicación de la constante kp por el error va a ser comparativamente mayor y la corrección es bastante más agresiva. Adicionalmente, cabe resaltar que el mayor error encontrado en la trayectoria fue de 0.342 metros, lo que es algo bastante tolerable, teniendo en cuenta la trayectoria recorrida y su magnitud. En este último aspecto, radica la mayor debilidad de este método y del control implementado en este, y es que en el tramo donde la velocidad alcanza su valor máximo vk, la corrección de la velocidad no va a ser muy significativa y por consiguiente el error no va a disminuir mucho. De aquí se explica, que si se llega al primer punto intermedio con algo de error, este no va a disminuir mucho al llegar al segundo punto, ya que en el tramo de velocidad constante el valor de la corrección realizada es muy inferior con respecto a vk. Sin embargo, en el tramo de desaceleración, donde viene todo el error acumulado, el *drone* tiene mayor margen de maniobra y alcanza a corregir el error que traía la trayectoria para así acabar los puntos de una manera correcta. Una manera de corregir este problema, sería discriminando la constante kp para cada tramo de la trayectoria.

Punto	Coordenada alcanzada	Coordenada calculada	Error
Punto de inicio	[0.000, 0.000]	[0.000, 0.000]	[0.000, 0.000]
Punto intermedio 1	[-1.201, -2.375]	[-1.071, -2.142]	[0.130, 0.233]
Punto intermedio 2	[-4.145, -8.008]	[-3.928, -7.857]	[0.217, 0.151]
Primer punto de ruta	[-5.011, -9.973]	[-5.000, -10.000]	[0.011, -0.027]
Punto intermedio 1	[-1.765, -12.351]	[-2.000, -12.142]	[-0.235, 0.209]
Punto intermedio 2	[6.342, -18.082]	[6.000, -17.857]	[-0.342, 0.223]
Segundo punto de ruta	[9.039, -20.047]	[9.000, -20.000]	[-0.039, 0.047]
Punto intermedio 1	[9.998, -16.398]	[9.857, -16.785]	[-0.141, -0.387]
Punto intermedio 2	[12.197, -7.842]	[12.142, -8.214]	[-0.055, -0.372]
Tercer punto de ruta	[13.096, -5.032]	[13.000, -5.000]	[-0.096, 0.032]
Punto intermedio 1	[10.029, -3.849]	[10.214, -3.928]	[0.185, -0.079]
Punto intermedio 2	[3.079, -1.161]	[2.785, -1.071]	[-0.294, 0.090]
Fin de la trayectoria	[-0.087, 0.000]	[0.000, 0.000]	[0.087, 0.000]

Tabla 6. Coordenadas de la trayectoria simulada usando el nodo "PVT_Movimiento_Vectores".

6.1.3.2. Simulación del nodo "PVT_Movimiento_Rotar"

Para las simulaciones del nodo "PVT_Movimiento_Rotar", se realizó la misma trayectoria que la usada en el otro método de movimiento, esto para poder comparar la validez de ambos métodos y sus ventajas y

desventajas. La mayor ventaja de la rotación es que en los casos en los que se desee asistir la navegación del *quadrotor* usando la cámara principal, este método se encarga de que siempre se avance mirando en la dirección del punto de ruta siguiente. En este trabajo solo se desarrolló la navegación de un punto de ruta a otro, sin embargo con la posibilidad de avanzar con la cámara mirando hacia el destino, se podrían agregar distintas características tales como la evasión de obstáculos, u otros elementos que aportarían a la navegación o a la tarea de detección.

Sin embargo, el hecho de realizar la rotación complica las cosas a la hora de realizar el control y al realizar la trayectoria. Esto debido a que las correcciones y la trayectoria de referencia se deben hacer sobre el eje coordenado del mundo, sin embargo, al rotar el *drone*, no se puede corregir el movimiento con cambios sencillos de la velocidad en x, y, esto porque los comandos se realizan con respecto al eje del *drone* y no al eje mundo. Sin embargo, el control que se incorpora en este método contrarresta esos problemas girando sobre su eje en cada momento del recorrido, para que siempre se encuentre en una trayectoria justo hacia el punto de ruta de destino. Se presenta también un problema adicional y es que la rotación no se realiza exactamente sobre el eje y normalmente, cuando se realiza la rotación, el *drone* se ha desplazado un poco fuera de la trayectoria. Por este motivo se observa que al comienzo de cada trayectoria entre los diferentes puntos de ruta, se realiza una corrección evidente del ángulo para meterse de nuevo en la trayectoria trazada.

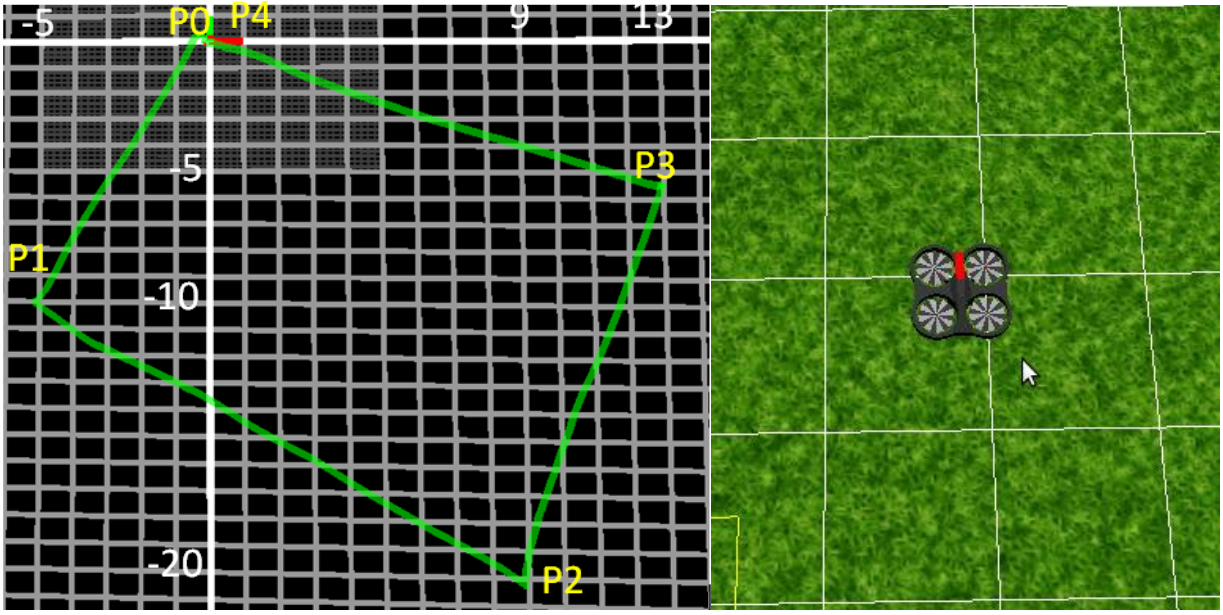

Figura 24. Resultado de la simulación para el nodo "PVT_Movimiento_Rotar".

Punto	Coordenada alcanzada	Coordenada calculada	Error
Punto de inicio	[0.000, 0.000]	[0.000, 0.000]	[0.000, 0.000]
Punto intermedio 1	[-1.367, -2.155]	[-1.071, -2.142]	[0.296, 0.013]
Punto intermedio 2	[-4.161, -7.997]	[-3.928, -7.857]	[0.233, 0.140]
Primer punto de ruta	[-5.076, -9.946]	[-5.000, -10.000]	[0.076, -0.054]
Punto intermedio 1	[-2.090, -12.182]	[-2.000, -12.142]	[-0.090, 0.130]

Punto intermedio 2	[5.894, -17.848]	[6.000, -17.857]	[-0.106, 0.009]
Segundo punto de ruta	[8.825, -19.983]	[9.000, -20.000]	[0.175, -0.017]
Punto intermedio 1	[9.513, -16.552]	[9.857, -16.785]	[0.344, -0.233]
Punto intermedio 2	[12.009, -8.169]	[12.142, -8.214]	[0.133, -0.045]
Tercer punto de ruta	[12.839, -5.321]	[13.000, -5.000]	[0.161, 0.321]
Punto intermedio 1	[9.988, -4.396]	[10.214, -3.928]	[0.136, 0.468]
Punto intermedio 2	[2.633, -1.300]	[2.785, -1.071]	[0.152, 0.229]
Fin de la trayectoria	[0.079, -0.085]	[0.000, 0.000]	[-0.079, 0.085]

Tabla 7. Coordenadas de la trayectoria simulada usando el nodo "PVT_Movimiento_Rotar".

Por estos problemas mencionados, el error con este método es un poco más grande, esto a expensas de ubicar la cámara mirando hacia el siguiente punto de ruta. Sin embargo, los valores de error siguen siendo bastante tolerables y no presentan ningún problema serio a la hora de realizar la trayectoria.

Al igual que con el otro nodo implementado, las simulaciones realizadas, de esta misma trayectoria y de otras trayectorias con características diferentes, se encuentran en los anexos.

6.2. Criterios y protocolos de prueba

Para realizar las pruebas reales de los algoritmos desarrollados, se realizó un viaje a la ciudad de Villavicencio, que se encuentra a una altura de 500 metros sobre el nivel del mar, lo que garantiza que el AR *Drone* 2.0 funcione de manera correcta. Allí se realizaron pruebas en un terreno óptimo para estas gracias a que este era plano y con el pasto corto, haciendo que no hubiera mayores obstáculos para realizar la trayectoria con un cierto margen de error. Sin embargo, el tamaño del terreno (15 x 30 metros) se presentó como un impedimento bastante grande debido a que las trayectorias que se debieron realizar fueron bastante cortas, haciendo que la variación de los parámetros (X, Y) no fuera mucha.

Es necesario resaltar que, debido a que los nodos desarrollados en ROS fueron probados en una plataforma física como el AR *Drone* 2.0, que si bien presenta muchas ventajas sobre todo en términos de economía, el bajo costo de esta, impide que los resultados alcanzados tengan una precisión mucho mayor. A esto se suma la baja fidelidad del GPS utilizado, que introduce un error de medición de 2 a 3 metros. Esto se evidencia cuando se deja el *drone* quieto en el suelo en el terreno de prueba y sin embargo, las mediciones del GPS no resultan muy estables. Este ruido o error hace que el recorrido de la trayectoria se realice de manera eficaz, pero con algunos segmentos donde el error se dispara, posiblemente debido a estos fallos.

Como se explicó anteriormente, este proyecto se enmarca dentro de un macro proyecto para la detección de minas antipersona. Por este motivo, el proyecto debe poder funcionar junto con el paquete creado anteriormente en otro trabajo de grado llamado "*drone*_detection", que se encarga de realizar la detección de las minas visibles en el terreno, utilizando la cámara inferior del AR *Drone* 2.0, además de otro proyecto que busca realizar la detección de minas enterradas usando un método que también requiere que el *drone* vuele a una altura baja (entre 0.3 m y 1m) para que se presente un correcto funcionamiento. Debido a que el control básico de altura, que es el que se necesita, está integrado dentro del paquete base para la comunicación con el *drone* (ar*drone*_autonomy), en los algoritmos desarrollados en este trabajo no se realiza ningún control de altura, ya que por lo expuesto anteriormente, resultaría innecesario y redundante.

Teniendo en cuenta las limitaciones descritas anteriormente se diseñó una trayectoria que aprovechara las posibilidades del terreno. Esta trayectoria consta de 4 puntos de ruta básicos que forman un rombo similar al que se observa en las figuras 24 y 25 de las simulaciones. Las coordenadas UTM de estos puntos se muestran en la tabla 7.

Punto	X	Y
Origen	650783	460433
Punto 1	650778	460423
Punto 2	650792	460413
Punto 3	650796	460428
Punto 4 y destino	650783	460433

Tabla 8. Coordenadas UTM de la trayectoria de pruebas.

Este trazado de la trayectoria se presenta ideal debido a que se recorren los cuatro posibles cuadrantes de movimiento o combinaciones de comandos de velocidad para el *drone*. Esto significa que en ambos ejes (X, Y) se realizaran desplazamientos negativos y positivos. En la figura 26. Se muestra el trazado de la trayectoria ideal en el terreno, usando Google Earth.

Figura 25. Trazado ideal de la trayectoria de pruebas.

En esta trayectoria, se probaron los dos métodos de movimiento implementados (rotación y vectores de velocidad). Los resultados demuestran que los diferentes métodos resultan efectivos, sin embargo, la limitante del GPS a la hora de realizar el movimiento, esto debido a que los datos que este arroja son bastante inexactos y lentos. Esto impide que la precisión pueda ser mayor. A continuación se presentan las pruebas realizadas para cada conjunto de parámetros.

Finalmente es necesario tener en cuenta al revisar las curvas de velocidad del *drone* que para realizar el movimiento, los ejes cambian con respecto a lo que se está acostumbrado normalmente. Para los movimiento de adelante y atrás se publica una velocidad en x, y para realizar un movimiento hacia izquierda o derecha, se publica una velocidad en y.

Para las pruebas en campo, se encontró que no se debe trabajar con una velocidad mayor a 1.5 m/s para que no se vea afectada la estabilidad del *quadrotor*. Si se aumenta este valor, se puede apreciar cómo la altura no permanece constante y la inclinación alcanzada se vuelve peligrosa de tal manera que se podrían ocasionar colisiones. Debido a que el AR *Drone* 2.0 es bastante frágil e inestable, se prefirió trabajar a menores velocidades para poder mantener una mayor estabilidad. Adicionalmente, debido a que la recepción de datos desde el GPS es muy lenta en comparación con la ejecución de los nodos de ROS, trabajar con altas velocidades sería contraproducente ya que no se podría ejecutar un correcto control del mismo basado en la posición. Para todas las pruebas realizadas, se estableció la velocidad vk en 1.5 m/s.

6.3. Experimentos en campo

En la trayectoria descrita se probaron los dos métodos de movimiento implementados (rotación y vectores de velocidad). Los resultados demuestran que los diferentes métodos resultan efectivos, sin embargo, la limitante del GPS a la hora de realizar el movimiento, esto debido a que los datos que este arroja son bastante inexactos y lentos. Esto impide que la precisión pueda ser mayor. A continuación se presentan las pruebas realizadas para cada conjunto de parámetros.

6.3.1. Nodo "PVT_Movimiento_Rotar"

El nodo de rotación se encarga de realizar la trayectoria planeada con la cámara mirando hacía el punto de destino y debido a las limitantes de control que presenta, su error es algo mayor comparado con el otro método. La figura 27 muestra algunas pruebas realizas, discriminándolas por colores para poder realizar una comparación efectiva con respecto a la trayectoria de referencia (color verde oscuro).

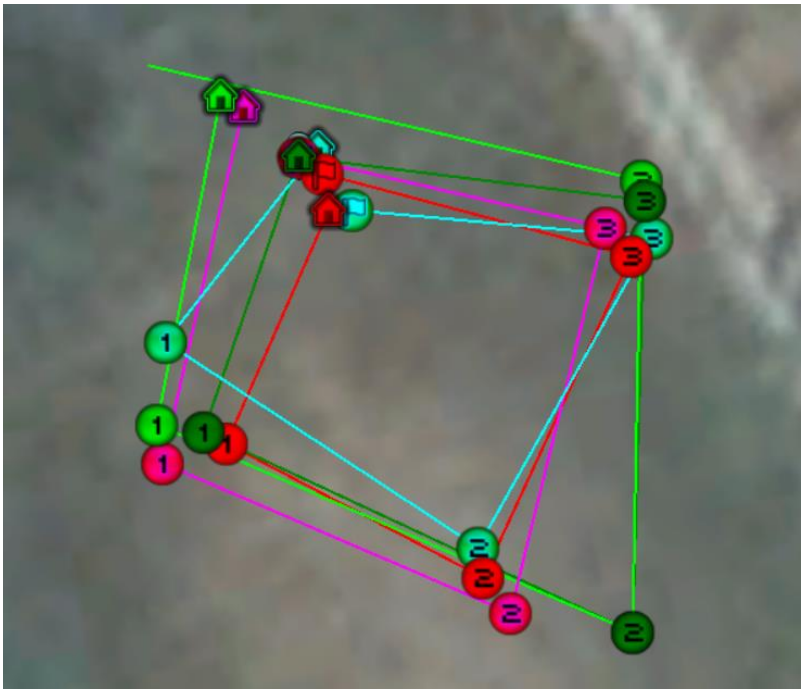

Figura 26. Pruebas realizadas con el nodo "PVT_Movimiento_Rotar".

Se puede observar que la trayectoria se realiza de manera satisfactoria con algunos errores. Especialmente en el punto 2, 3 de las 4 pruebas mostradas se quedan cortas, lo que puede deberse a que el primer punto acaba algo corrido, ocasionando un error que se propaga hacia este punto en el eje X. Ahora, si se observan las curvas obtenidas luego de procesar los datos para una de las pruebas realizadas (verde lima), se puede ver como en la trayectoria se realiza siempre una especia de giro hacia el centro o un desfase en la trayectoria en los vértices, lugares donde se ejecuta la rotación. Esto se debe principalmente a que el *drone* al realizar

la rotación tiene cierto error y se desplaza de su punto inicial, cosa que no debería suceder, al desplazarse se genera un error que hace que la trayectoria siguiente empiece con cierto error, que posteriormente se corrige con la reubicación hacia el punto de destino que se va realizando a medida que transcurre el perfil de velocidad trapezoidal, hasta que se llega al punto, con un error bastante reducido, lo que indica que el método de control por corrección de ángulo es bastante útil y acertado para este método. Al revisar las curvas del error se observa el mismo comportamiento, donde hay puntos donde este se ve bastante alto, debido a que la trayectoria se inicia con un corrimiento con respecto al punto inicial, sin embargo, en los puntos donde se realiza el cambio de punto, donde la velocidad es aproximadamente cero, el error es muy reducido, lo que lleva a la misma conclusión de que el mayor error se genera al moverse del punto actual cuando se realiza la rotación.

Adicionalmente, si se observan las curvas velocidad, la velocidad en x (según el eje del *drone*), que es la encargada de mover el *drone* hacia adelante, traza un claro comportamiento trapezoidal, pasando por etapas de aceleración, movimiento constante y desaceleración para el recorrido de cada punto de ruta.

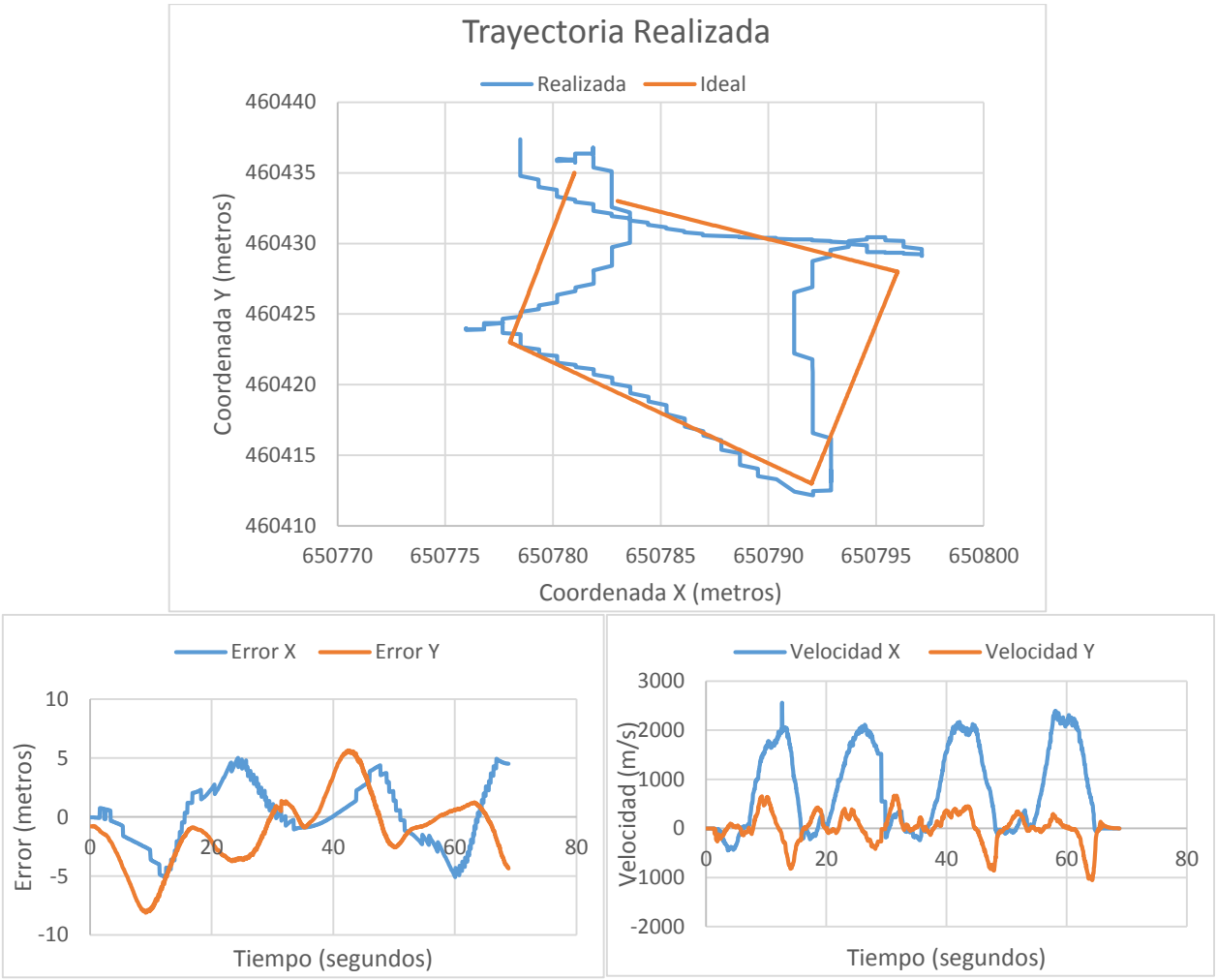

Figura 27. Gráficas de trayectoria, error y velocidad para una de las pruebas realizadas con el nodo de rotación.

En la tabla 8 se observa con mayor detalle el error que se tiene al momento de alcanzar cada uno de los diferentes puntos de ruta. La trayectoria verde lima fue la que tuvo el mayor error en el final de la trayectoria y sin embargo fue la más acertada en el resto de puntos. Puede analizarse también que generalmente los errores alcanzados son del orden de los 2-4 metros, lo que supone un valor bastante correcto si se tienen en

cuenta las características de la plataforma de prueba y el error del GPS utilizado, que de por sí es de 2 a 3 metros.

Prueba	Error punto 1 (metros)	Error punto 2 (metros)	Error punto 3 (metros)	Error punto final (metros)
Verde lima	[1.192, -0.927]	[-0.908, -0.568]	[-0.269, -2.071]	[4.521, -4.372]
Rojo	[-1.358, 0.179]	[4.181, -3.388]	[0.567, 0.775]	[-1.413, 0.846]
Magenta	[1.189, 0.548]	[3.324, -1.992]	[1.422, -0.436]	[-0.569, 0.055]
Cian	[0.343, -3.933]	[4.182, -4.442]	[-0.273, 0.195]	[-2.257, 2.323]

Tabla 9. Error en metros en cada punto de ruta alcanzado para cada una de las pruebas mostradas con el nodo de rotación.

6.3.2.Nodo "PVT_Movimiento_Vectores"

El nodo que utiliza el método de movimiento por medio del cálculo de los vectores de velocidad y posición se considera el principal y el más robusto en el trabajo desarrollado. Esto debido a que se comprobó, primero en simulaciones y luego en las pruebas experimentales que el método de control que este posee y la forma de ejecutar el movimiento son las más correctas y acertadas en términos de la cercanía a los puntos alcanzados.

Debido a que este nodo presenta un mejor funcionamiento al de la rotación y además tiene un control más robusto y confiable, con este algoritmo se realizaron diferentes pruebas, trazando diferentes trayectorias y probando diferentes constantes kp. A continuación se muestran las pruebas más importante y relevantes. Debe tenerse en cuenta que los valores de kp usados debían ser muy pequeños, debido a que la publicación de la velocidad que se realiza hacia el paquete ar*drone*_autonomy tiene un rango entre -1 y 1. Además al error estar dado en metros con respecto a la trayectoria de referencia, no se podía suponer que el seguimiento de la trayectoria fuera tan exacto y si se adiciona el error del GPS que es de 2 o 3 metros, entonces la modificación de la velocidad debería cambiarse más no sobrescribir totalmente la calculada. Por esto se trabajó con constantes kp con una magnitud aproximadamente 10 veces menor que el valor de la publicación de velocidad realizada.

6.3.2.1. Pruebas con kp=0.01;

Para este conjunto de pruebas, se trabajó con una constante kp=0.01. Esto significa que con un error de dos metros, se realizaría una corrección de 0.02 en la publicación de la velocidad del *drone*, lo que corresponde a un 20% de la velocidad máxima alcanzada o con que se está trabajando. En la figura 29 se pueden observar las diferentes pruebas realizadas, donde la trayectoria verde oscura corresponde a la de referencia.

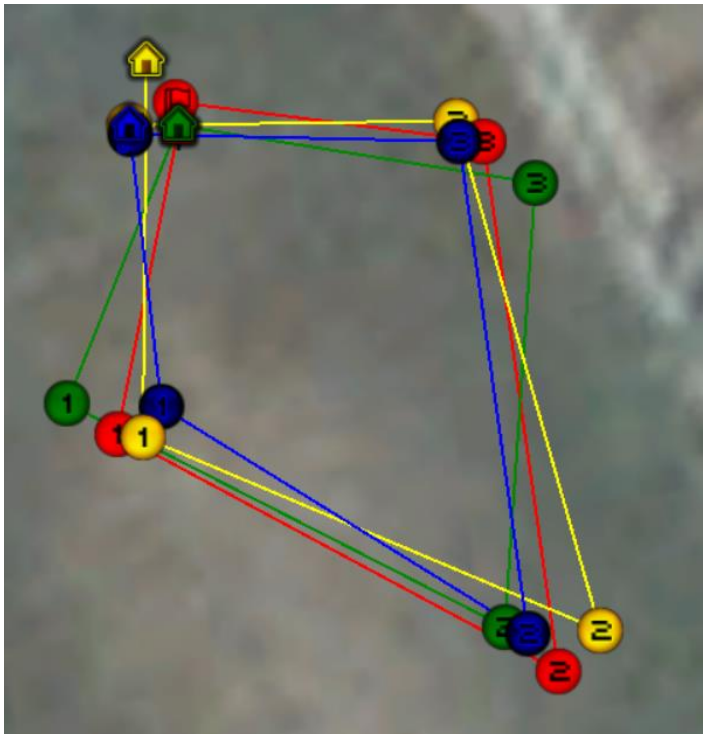

Figura 28. Pruebas realizadas con el nodo "PVT_Movimiento_Vectores" con kp=0.01.

Es claro que estas trayectorias son mucho más acertadas en términos de precisión con respecto a las que se realizaron con el método de rotación. El hecho de que la constante kp se encargue de corregir la velocidad en términos de si está más atrás o más adelante a la trayectoria de referencia, hace que los puntos se acerquen mucho a los ideales. Si se observan las gráficas especificas a la trayectoria roja, se puede entender mucho mejor el funcionamiento del algoritmo y los posibles errores. Lo primero que es necesario resaltar es el hecho de que la trayectoria realizada difiere durante la mayor parte del recorrido bastante con respecto a la ideal y sin embargo al final de los puntos se realiza una corrección más evidente para acabar con poco error. Esto se debe a que la corrección realizada es proporcional con respecto al error pero también al comando de velocidad para el instante específico de la trayectoria. Como la constante utilizada es de un valor tan pequeño, la multiplicación del error por la constante en el tramo de velocidad constante no es mucho, por lo tanto no se realiza mayor corrección porque no se puede anular totalmente la velocidad que lleva el *drone*, sin embargo, en el tramo de desaceleración, a donde se llega con un error acumulado, se puede realizar una mayor corrección porque las velocidades son más pequeñas y por tanto la multiplicación del error por la constante kp es mayor con respecto a la velocidad del instante y así la corrección que se puede hacer es mucho mayor.

En la tabla 9 se evidencia que en términos generales, el seguimiento de las trayectorias es mucho más preciso con este método ya que el mayor error alcanzado fue de -4.299 metros, sin embargo en términos generales, el error osciló en los valores deseables de hasta 3 metros.

Prueba	Error punto 1 (metros)	Error punto 2 (metros)	Error punto 3 (metros)	Error punto final (metros)
Amarillo	[-3.054, 1.181]	[-4.299, 0.063]	[2.269, -3.863]	[1.124, -0.207]
Rojo	[-2.201, 0.970]	[-2.606, 1.276]	[1.426, -2.967]	[-0.569, -0.893]
Azul	[-3.894, 0.232]	[-1.762, -0.252]	[2.268, -3.125]	[1.125, 0.108]

Tabla 10. Error en metros en cada punto de ruta alcanzado para cada una de las pruebas mostradas con el nodo de vectores y kp=0.01.

Observando también las gráficas de error y velocidad se puede encontrar este comportamiento. En los tramos donde la velocidad es grande, el error tiende a aumentar, sin embargo en los momentos donde esta empieza a disminuir, el error también lo hace. En las curvas de velocidad debe observarse algo muy particular con respecto a las observadas para el método de rotación. Primero que todo, en este método se realiza movimiento en ambos ejes del *drone*, tanto x como y, lo que se queda expuesto en la figura, sin embargo, los perfiles de velocidad trapezoidal no son tan claros y es debido a que el control realizado modificaba directamente la velocidad del *quadrotor* en todo momento, lo que hace que se observen picos y sobresaltos en las curvas, o que en general no se puedan distinguir fácilmente las etapas de movimiento.

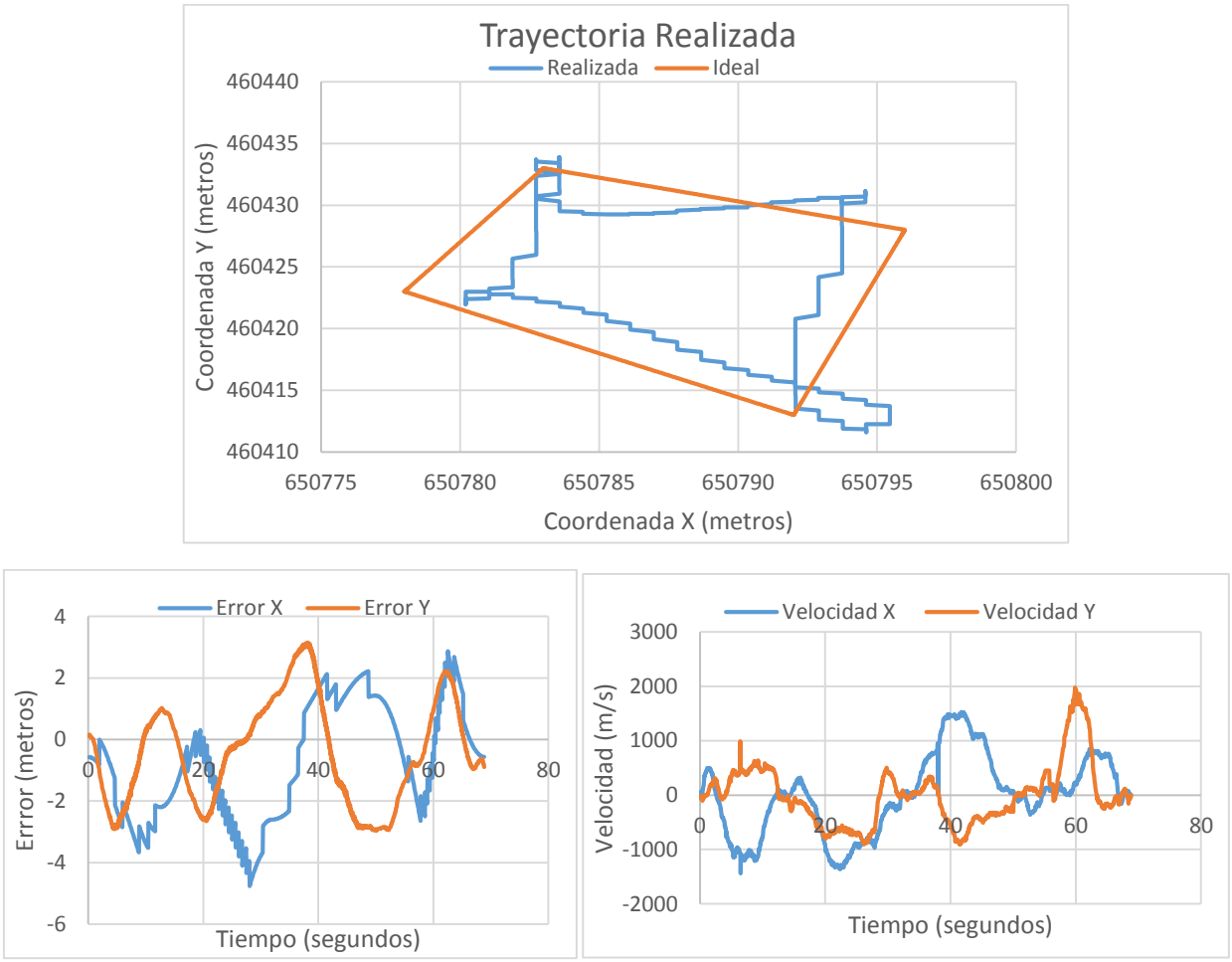

Figura 29. Gráficas de trayectoria, error y velocidad para una de las pruebas realizadas con el nodo de vectores.

6.3.2.2. Pruebas con kp=0.02

Debido al fenómeno explicado anteriormente, que hace que la constante proporcional utilizada resulte insuficiente para realizar la corrección en el tramo de velocidad constante, se decidió probar con el valor de kp=0.02. Esto hace que las correcciones a la velocidad sean mucho más agresivas. En la figura 31 se observan las trayectorias realizadas, donde la trayectoria verde oscura corresponde a la de referencia.

Al observar las trayectorias, se puede saber que aumentar el kp a 0.02 produce un efecto negativo en la trayectoria debido a que al realizar correcciones más agresivas, de cierta manera, en los tramos acelerados de desacelerados se ignora un poco la velocidad calculada debido a que siempre va a existir un error, entonces la corrección anula estos cálculos. Esto se representa por medio de picos más pronunciados en las curvas de velocidad. Por cuestiones de brevedad estas gráficas no se mostrarán, si se desea hacer una revisión de estas, se encuentran en los documentos anexos.

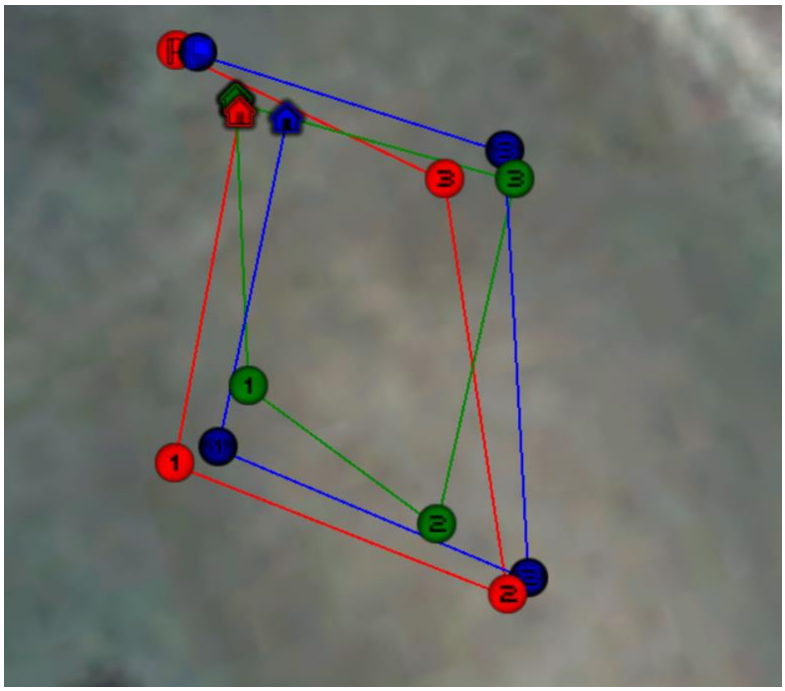

Figura 30. Pruebas realizadas con el nodo "PVT_Movimiento_Vectores" con kp=0.02.

Adicionalmente, en la tabla 10 se muestran los errores en cada punto de ruta que se realizó. Estos errores se distribuyen bastante y debido al control agresivo, el error al llegar a los puntos es muy bueno (el mayor es de -2.289 metros). Sin embargo, al revisar la trayectoria que se realizó en cada momento, está difiere bastante de la ideal y presenta unas diferencias significativas que hacen que este valor de kp no sea el ideal.

Prueba	Error punto 1 (metros)	Error punto 2 (metros)	Error punto 3 (metros)	Error punto final (metros)
Rojo	[0.721, 1.973]	[-1.448, 2.882]	[2.0417, -1.723]	[1.518, -1.896]
Azul	[-0.973, 1.868]	[-2.289, 2.618]	[-0.493, -2.120]	[0.664, -1.475]

Tabla 11. Error en metros en cada punto de ruta alcanzado para cada una de las pruebas mostradas con el nodo de vectores y kp=0.02.

6.3.2.3. Pruebas con kp variable

Basándose es los resultados de las pruebas mostradas anteriormente y el error que se incrementa en los tramos de velocidad constante, se decidió contrarrestar esto haciendo que la constante de control kp cambiara su valor dependiendo del tramo que se estuviera realizando. De esta manera, en los tramos de aceleración y desaceleración, se estableció el valor de kp=0.01, debido a se comprobó que este valor funciona de manera correcta en estos tramos y no se requiere una mayor corrección y en el tramo de

velocidad constante se estableció un valor de kp=0.03 para que la corrección realizada resultara algo más significativa con respecto a la publicación de velocidad que se realiza en este tramo.

Punto	X	Y
Origen	606673	520308
Punto 1	606674	520328
Punto 2	606695	520329
Punto 3	606694	520309
Punto 4 y destino	606673	520308

Tabla 12. Coordenadas UTM de la trayectoria de referencia para las pruebas realizadas en la ciuc¿dad de Bogotá.

Estas pruebas fueron realizadas en la ciudad de Bogotá, en el parque el Country, que cuenta con una gran zona verde y un espacio muy amplio libre de obstáculos que se presentaba propicio para realizar las mediciones. Esto porque para disminuir el error de la medición del GPS, es óptimo probar en lugares donde no haya muchos árboles, edificios cercanos, o incluso, muchas redes wifi. Para el diseño de esta trayectoria, debido a que no se tenían las limitantes de espacio, se diseñó una ruta un poco más larga y que se asemejara aun cuadrado. Los puntos de ruta y la trayectoria ideal se pueden observar respectivamente en la tabla 11 y en la figura 32.

Figura 31. Trazado ideal de la trayectoria de referencia.

En la figura 32, donde se muestra la trayectoria ideal para esta serie de pruebas, se muestra en contexto, para que se observe la gran extensión del terreno de pruebas y la distancia del recorrido (aproximadamente 20 metros entre cada punto de ruta). En la figura 33 se pueden observar las pruebas realizadas, con un acercamiento mucho mayor. En esta serie de pruebas se observa claramente que en comparación con la trayectoria son las más acertadas que se realizaron, no solo en términos de la precisión en la finalización de los puntos, sino en términos de la eficacia con la que se realizó la trayectoria durante cada momento del recorrido. Esto se presenta de vital importancia, porque a la hora de integrar el proyecto para realizar determinadas tareas, tales como el escaneo de un área, se requiere que la trayectoria se realice de manera

precisa en cada momento, no solo que se llegue al punto especificado. Las trayectorias amarilla y azul, corresponden al nodo normal "PVT_Movimiento_Vectores", con el único cambio de la constante kp variable. La trayectoria roja, corresponde a una prueba que se realizó para demostrar que el *drone*, además de recorrer los determinados puntos de ruta, se puede quedar haciendo vuelo estacionario sobre cada uno de ellos, nuevamente, con el fin de integrar el proyecto hacía determinados requerimientos específicos. En esta prueba se realizó un vuelo estacionario de aproximadamente 6 segundos una vez que se llegaba a cada punto.

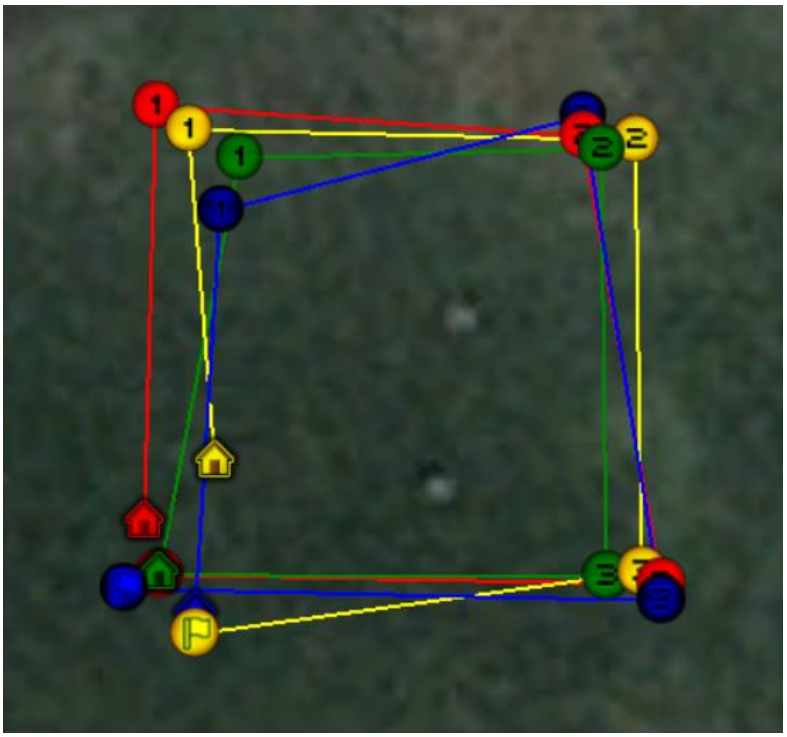

Figura 32. Pruebas realizadas con el nodo "PVT_Movimiento_Vectores" con kp variante.

Este vuelo estacionario se realiza de manera correcta, sin embargo, debido a que estas pruebas fueron realizadas en Bogotá (a 2500 metros sobre el nivel del mar) el funcionamiento del *drone* no es el mejor y en el vuelo estacionario, este se desplaza un par de metros de su lugar de origen. Este error se observar en la figura xx donde hay un par de picos de error, ocasionados porque la trayectoria de referencia indica que este debería quedarse en el mismo lugar, sin embargo, este se desplaza un poco realizando el vuelo estacionario y se origina el pico de error, que luego es rápidamente corregido al iniciar la trayectoria, donde en las velocidades se produce un pico debido a la corrección que se realiza del error. En la gráfica de la velocidad en la figura xx, se pueden observar los valles donde la velocidad en ambos ejes es muy cercana a cero, lo que corresponde al vuelo estacionario, donde no hay velocidad porque el *drone* se queda teóricamente en la misma posición, estos valles están encerrados en los pequeños rectángulos negros para resaltar esta característica especifica de la prueba.

Las otras pruebas realizadas muestran también la precisión y las ventajas de usar una constante proporcional que varía dependiendo del tramo del recorrido, ya que ambas realizan una trayectoria muy parecida a la ideal o teórica. Es importante resaltar el caso de la trayectoria amarilla, donde a pesar de empezar el recorrido desplazado algunos metros de la posición de origen teórica, logra corregir el curso y realizar el resto del recorrido de manera satisfactoria.

Prueba	Error punto 1 (metros)	Error punto 2 (metros)	Error punto 3 (metros)	Error punto final (metros)
Rojo	[1.648, -3.627]	[1.488, -0.919]	[-2.915, 0.694]	[-0.222, -0.696]
Azul	[-1.744, 1.643]	[1.490, -2.026]	[-2.916, 1.485]	[1.469, 0.042]
Amarillo	[-0.046, -2.415]	[-1.032, -0.813]	[-2.061, 0.114]	[-1.918, 2.097]

Tabla 13. Error en metros en cada punto de ruta alcanzado para cada una de las pruebas usando el nodo de vectores y kp variante.

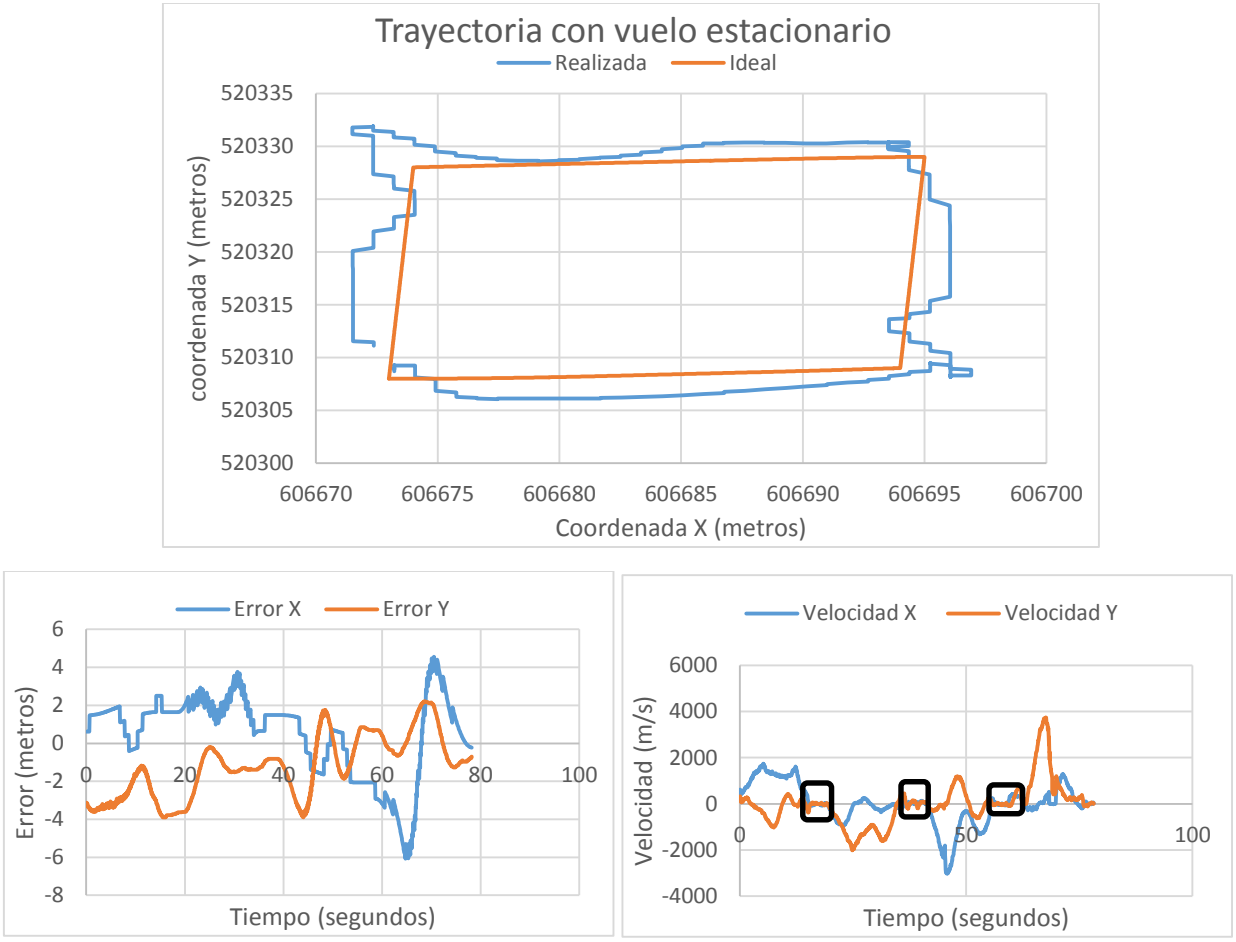

Figura 33. Gráficas de trayectoria, error y velocidad para la prueba con vuelo estacionario de 6 segundos (trayectoria roja en la Figura 33).

En general, en la tabla 12, se puede observar que los errores de finalización de trayectoria no fueron grandes, sin embargo, se puede llegar a decir que estas pruebas realizadas en otro lugar con menos altura sobre el nivel del mar, hubieran resultado mucho más satisfactorias, esto porque al error del GPS, en estas pruebas se le sumó la inestabilidad presentada por el *drone* en estas condiciones de altura, esto hace que este no responda de manera correcta a todos los comandos enviados. Sin embargo, observando las curvas del error en ambas coordenadas, este se presenta mucho más estable durante todo el recorrido, sin picos muy altos y con un seguimiento constante. Por la ejecución de la trayectoria en términos generales y por lo encontrado en todas las pruebas, se puede decir que el mejor método encontrado es el de los vectores de velocidad,

usado con una constante proporcional que cambia dependiendo del tramo del recorrido en el que se encuentra el *quadrotor*.

6.4. Validez de los resultados

Juntando todas las pruebas obtenidas y los errores que se encontraron en el sistema y en la ejecución de las diferentes trayectorias, se puede hablar del cumplimiento del objetivo en términos de la precisión obtenida y de la ejecución de las trayectorias dispuestas. Ambos nodos implementados, el "PVT_Movimiento_Rotar" y el "PVT_Movimiento_Vectores" lograron ejecutar, con errores aceptables, las trayectorias ordenadas, pasando por los diferentes puntos de ruta, realizando de manera satisfactoria el perfil de velocidad trapezoidal calculado, para cada uno. El error obtenido en el peor de los casos fue de 4 metros en la finalización de un punto, mientras que en el mejor de los casos, se llegó a una distancia de 0.042 metros del punto de ruta.

Si se realiza una valoración general del proyecto, teniendo en cuenta las limitaciones de tiempo, de la plataforma de pruebas usada (AR *Drone* 2.0), del error inherente a las mediciones del GPS, los resultados obtenidos presentan un éxito que deja abiertas las pruebas a futuros trabajos donde se pueda mejorar la exactitud, partiendo de las limitaciones mencionadas.

Pruebas realizadas	Error máximo en la llegada a los puntos de ruta (metros)	Error mínimo en la llegada a los puntos de ruta (metros)	Error promedio de llegada (metros)
Nodo de rotar	4.521	0.055	1.580
Nodo de vectores (kp=0.01)	4.229	0.063	1.738
Nodo de vectores (kp=0.02)	2.882	0.493	1.667
Nodo de vectores (kp variante)	3.627	0.042	1.479

Tabla 14. Consolidación de los errores de llegada a los puntos de ruta en las pruebas realizadas.

6.5. Costos

El desarrollo del proyecto puede dividirse en tres tipos de costos relacionados con el mismo, el asociado con los equipos físicos y las plataformas que fueron usadas para el mismo, el destinado a la manutención y transporte requerido para realizar las pruebas en Villavicencio, Colombia, durante dos días, donde se realizaron la mayoría de las pruebas y el relacionado con la mano de obra de los autores del trabajo. En la tabla xx se pueden observar en detalle los costos desglosados.

Descripción	**Costos**
AR *Drone* 2.0	$ 766,100.00
Hélices de repuesto	$ 45,000.00
Repuesto de la cruz central del AR *Drone* 2.0	$ 140,000.00
Estación Base Linux PC (XPS 13 Ultrabook)	$ 1,900,000.00
Total Equipos Físicos	$2,851,100.00

Alimentación (por persona)	$ 25,000.00
Transporte (por persona)	$ 30,000.00
Hospedaje (por persona)	$ 50,000.00
Total viaje (por persona)	$ 105,000.00

Mano de obra (por persona)	$3,000,000.00

Total	$9,061,100.00

Tabla 15. Costo aproximado de la realización del proyecto de seguimiento de trayectorias.

7. CONCLUSIONES

Los objetivos del proyecto se cumplieron a cabalidad, ya que se desarrolló de manera completa un conjunto de algoritmos integrados dentro del entorno de desarrollo para robótica ROS, dentro de un paquete llamado "*drone*_GNC", que se encargan de realizar la planeación y el seguimiento de trayectorias espaciales por medio de puntos de ruta GPS. Estos algoritmos, que fueron desarrollados en C++, se encargan de realizar la planeación del perfil de velocidad trapezoidal y de ejecutar el seguimiento de este, haciendo uso del GPS y de la IMU a bordo del UAV. Para realizar las pruebas de simulación, se usó el paquete de ROS "tum_simulator" con el que se verificó la exactitud del método implementado. Adicionalmente, en la realidad se realizaron más de 30 pruebas en Villavicencio y Bogotá, que sirvieron para evaluar el desempeño de los algoritmos en términos de la cercanía a los puntos de ruta ingresados.

Con los algoritmos finales desarrollados en cada uno de los nodos, se logró llegar a unos resultados exitosos en las pruebas realizadas, donde los errores se encuentran dentro del rango aceptable, el mismo que tiene el GPS (± 2 metros), al seguir una trayectoria determinada con cuatro puntos de ruta ingresados en coordenadas UTM. El algoritmo de generación de trayectorias funcionó siempre de manera correcta y ambos métodos de movimiento implementados, el de rotar y el de movimiento por vectores, cumplieron su objetivo de llevar el *drone* de un punto inicial a uno final, pasando por determinados puntos de ruta, orientándose por medio del GPS. A pesar de que todas los algoritmos cumplieron con el objetivo, se determinó al final que la manera más precisa que se logró desarrollar para recorrer los puntos, fue aquella en la que haciendo uso del método de los vectores de movimiento, se estableció la constante kp en un valor variable (kp=0.01 en los tramos acelerados y kp=0.03 en el tramo de velocidad constante). Esto condujo a una mejoría significativa en la ejecución de la ruta y en la estabilidad general del sistema, haciendo que el error general del recorrido se viera acotado a un rango más pequeño que en los otros métodos y pruebas. Sin embargo, el sistema resulta bastante dependiente de las condiciones del terreno, donde se requiere un espacio abierto sin obstáculos, donde no se presente mucha interferencia para el dispositivo GPS y donde además, se presenten condiciones de viento casi nulas.

Es necesario hablar también de las múltiples limitaciones con las que se encontró el desarrollo del proyecto, la primera de ellas ocasionada por el uso del AR *Drone* 2.0, que a pesar de ser una plataforma económica y con determinadas bondades, se queda corta en aspectos como su gran sensibilidad al viento, su poca capacidad de volar en alturas superiores a los 2000 metros sobre el nivel del mar, el corto alcance de la red de comunicación wifi y sobre todo, la falta de precisión del GPS. Esta limitación hace que la exactitud de los resultados no pueda ser mayor, debido a dos aspectos principales del GPS, el error y la tasa del envío de datos. Estos factores hacen que el control no pueda actuar siempre de manera precisa, debido a que en muchas ocasiones los datos recibidos no son correctos, interfiriendo con el cálculo del error y los comandos de velocidad enviados al *drone*. Además, la lentitud de los mismos hace que se actúe con determinado retardo, ya que el control y los comandos de movimiento se ejecutan a una velocidad mucho mayor de lo que se realiza la recepción de los datos. Por esos motivos, se podría sugerir el uso de otro UAV para trabajos

futuros, que presente una mayor estabilidad en el vuelo, que permita que los comandos de velocidad se ejecuten de manera mucho más exacta. Adicionalmente, se debería utilizar otro dispositivo GPS que entregara una mayor precisión y confiabilidad en las medidas entregadas, lo que permitiría desarrollar un control más robusto.

Para finalizar, es necesario resaltar que este trabajo puede presentar una base importante para futuros desarrollos. El hecho de que el paquete desarrollado queda como software de libre acceso, permite que se pueda llegar a considerables mejoras al paquete, que por la gran limitante de tiempo del trabajo, no fue posible realizar. La primera de estas mejoras podría ser el desarrollo de un método de control más robusto que permitiera que el seguimiento de la trayectoria se realizara con mucha más exactitud. Otro posible avance incluye el desarrollo de una interfaz gráfica de usuario que brindara una mejor experiencia de uso a la hora de seleccionar los diferentes puntos de ruta. Así mismo, resaltando el macro proyecto titulado: **De-MiBot: Robot aéreo para detección de minas explosivas en campos rurales,** dentro del que se integra este trabajo de grado, se podrían generar muchos proyectos que se enfocarían en realizar la integración de todas las partes previamente desarrolladas, logrando llegar al objetivo final de integrar un robot aéreo tipo *quadrotor* como herramienta complementaria para las labores de detección de minas antipersona en terrenos rurales. Con el objetivo en mente de tener una plataforma capaz de recorrer el terreno minado en cuestión de manera autónoma.

8. BIBLIOGRAFÍA

[1] S. Bouabdallah, P. Murrieri, and R. Siegwart, "Design and control of an indoor micro *quadrotor*," IEEE Int. Conf. Robot. Autom. 2004. Proceedings. ICRA '04. 2004, vol. 5, 2004.

[2] J. Engel, J. Sturm, and D. Cremers, "Accurate figure flying with a quadrocopter using onboard visual and inertial sensing," IMU, 2012.

[3] G. M. Hoffmann, S. L. Waslander, and C. J. Tomlin, "*Quadrotor* Helicopter Trajectory Tracking Control," Electr. Eng., vol. 44, no. August, pp. 1–14, 2008.

[4] H. H. H. Huang, G. M. Hoffmann, S. L. Waslander, and C. J. Tomlin, "Aerodynamics and control of autonomous *quadrotor* helicopters in aggressive maneuvering," 2009 IEEE Int. Conf. Robot. Autom, 2009.

[5] T. Puls, M. Kemper, R. Kuke, and a. Hein, "GPS-based position control and waypoint navigation system for quadrocopters," 2009 IEEE/RSJ Int. Conf. Intell. Robot. Syst., pp. 3374–3379, 2009.

[6] B. Hofmann-Wellenhof, H. Lichtenegger, and J. Collins, Global positioning system: theory and practice / B. Hofmann-Wellenhof, H. Lichtenegger, J. Collins. 2001.

[7] T. Puls, H. Winkelmann, S. Eilers, M. Brucke, A. Hein, T. Kröger, and F. M. Wahl, "Interaction of Altitude Control and Waypoint Navigation of a 4 Rotor Helicopter," Adv. Robot. Res., pp. 287–298, 2009.

[8] J. Wendel, O. Meister, C. Schlaile, and G. F. Trommer, "An integrated GPS/MEMS-IMU navigation system for an autonomous helicopter," Aerosp. Sci. Technol., vol. 10, pp. 527–533, 2006.

[9] G. Hoffmann, H. Huang, and S. Waslander, "*Quadrotor* helicopter flight dynamics and control: Theory and experiment," Am. Inst. Aeronaut. Astronaut, pp. 1–20, 2007.

[10] M. Monajjemi. (2012, Julio 19). Ar*drone*_autonomy: A ROS Driver for AR*Drone* 1.0 & 2.0. [En linea]. Disponible: https://github.com/AutonomyLab/ar*drone*_autonomy.

[11] H. Huang, J. Sturm. (2013, Junio 27). Tum_simulator. [En linea]. Disponible: http://wiki.ros.org/tum_simulator

[12] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System," in ICRA Workshop on Open Source Software, 2009, vol. 32, pp. 151–170.

[13] A. Barrientos, J. Colorado, J. Cerro, A. Martinez, C. Rossi, and D. Sanz, "Aerial Remote Sensing in Agriculture : A Practical Approach to Area Coverage and Path Planning for Fleets of Mini Aerial Robots," vol. 28, no. 5, pp. 667–689, 2011.

[15] A. Martinez, E. Fernandez, "Learning ROS for Robotics Programming", Packt Publishing Ltd, 2013.

[16] S. Bouabdallah, A. Noth, and R. Siegwart, "Autonomous Systems Laboratory Dynamic Modeling of UAVs," pp. 0–18, 2006.

[17] R. W. Beard, "Quadrotor Dynamics and Control," pp. 1–47, 2008.

[18] T. Krajnık, V. Vonasek, D. Fiser, and J. Faigl.: AR-drone as a platform for robotic research and education. In Proc. of the Communications in Computer and Information Science (CCIS). 2011.
[19] I. Hu, Van, "Technical documentaction for AR.Drone project", 2013

[20] P. Correia, "Guía práctica del GPS", Marcombo, 2000, [En linea]. Disponible: http://books.google.es/books?hl=es&lr=&id=iTW7BBKScHsC&oi=fnd&pg=PA14&dq=guia+pr%C3%A1ctica+del+GPS&ots=k4wD0QiBAI&sig=ZRkr70ppEyzKEo5VQbKNPUtV-Z0#v=onepage&q=guia%20pr%C3%A1ctica%20del%20GPS&f=false

[21] A. El-Rabbany, "Introduction to GPS: The Global Positioning System", 2002, [En linea]. Disponible: http://books.google.es/books?hl=es&lr=&id=U2JmghrrB8cC&oi=fnd&pg=PR13&dq=Introduction+to+GPS:+The+Global+Positioning+System&ots=9LxWpTXHGT&sig=_dwGFznEtyt1BuXn4_rnuO8LCos#v=onepage&q=Introduction%20to%20GPS%3A%20The%20Global%20Positioning%20System&f=false

[22] "Coordenadas geográficas", [En línea]. Disponible: https://geodados.wordpress.com/2010/05/12/coordenadas-geograficas/

[23] "Distributions ROS", [En línea]. Disponible: http://wiki.ros.org/Distributions.

[24] S. Bouabdallah, "Design and control of quadrotors with application to autonomous flying," vol. 3727, 2007.

[25] S. Bouabdallah and R. Siegwart, "Full Control of a Quadrotor," no. 1, pp. 153–158, 2007.

[26] J. Colorado, "Towards Miniature MAV Autonomous Flight : A Modeling & Control Approach", Madrid, Abril 2009. Trabajo de grado (Maestría en Automática, Robótica). Universidad Politécnica de Madrid. Departamento de automática e ingeniería electrónica.

[27] J. Rodriguez, C. Castiblanco, "UAV para la detección de minas antipersona utilizando algoritmos de visión", Bogotá, 2013. Trabajo de grado (Ingeniería electrónica). Pontificia Universidad Javeriana. Facultad de ingeniería, departamento de robótica.

[28] R. Sanchis, "Implementación digital de controladores PID", pp 1-7.

[29] S. Piskorski, N. Brulez, P. Eline, "AR. Drone Developer Guide", Mayo, 2011.

[30]G.V. Raffo, "Modelado y control de un helicóptero quadrotor". Sevilla, 2007. Trabajo de grado de grado (Maestría en Automática, Robótica y Telemática). Universidad de Sevilla.

9. ANEXOS

Los anexos mencionados a lo largo del libro, que incluyen los algoritmos, pseudocódigos, algunos diagramas y el paquete "*drone*_GNC" desarrollado e implementado en ROS, las simulaciones completas, los datos y gráficas de las pruebas de campo y algunos videos de estas, se encuentran en el CD entregado junto con este documento.