BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA ELECTRÓNICA

Implementación en ROS de un control para seguimiento de trayectoria para un cuatrirotor

Tesis que presenta

Ivan Abel Cortés Benito

Para obtener el grado de

Ingeniero

En la licenciatura de

Ingeniería en Mecatrónica

Directores de Tesis

Dr. José Fermi Guerrero Castellanos
Dr. Hugo Rodríguez Cortés

Ciudad de Puebla, México

Febrero, 2023

Agradecimientos

Antes que nada, me gustaría agradecer a mi familia, especialmente a mi madre Monica y a mi padre Avelino, por apoyarme a lo largo de mi vida y acompañarme tanto en mis momentos de éxito como en mis momentos más difíciles. Muchos de mis objetivos han sido posible gracias a ellos, sus enseñanzas, su apoyo y su confianza en mi.

Agradecimientos a mis hermanos y hermana, mis tíos y tías, y a toda mi familia que me ha apoyado de manera incondicional.

Un agradecimiento a mis profesores de la Facultad de Ciencias de la Electrónica perteneciente a la Benemérita Universidad Autónoma de Puebla.

Un agradecimiento especial al Dr. Hugo Rodríguez Cortés, perteneciente al Centro de Investigación y Estudios Avanzados del IPN quien me apoyó de manera incondicional a alcanzar este objetivo profesional, además de ser uno de los principales pilares en el desarrollo de este trabajo.

Un gran agradecimiento al Dr. José Fermi Guerrero Castellanos por su apoyo en la realización de este trabajo.

Un agradecimiento a mis amigos que conocí durante mi trayecto en esta facultad asi como también en el CINVESTAV. Especialmente a Gustavo Fuentevilla y Marco Ramírez quienes me brindarón su ayuda y apoyo incondicional cada vez que lo necesitaba.

Cuando un sueño tiene fecha, se convierte en meta.
-Cmte. Sergio Cortés-

Índice general

Índice de figuras

Índice de tablas

Resumen

Los vehículos aéreos no tripulados conocidos como UAV (Unmanned Aerial Vehicles) por sus siglas en inglés, tienen diversas áreas de aplicación, estas pueden ir desde la seguridad y defensa, hasta actividades de recreación. La mayoría de los drones comerciales tiene la característica de poder ser pilotados manualmente por un operador. En este trabajo, se desarrolla un algoritmo de control para seguimiento de trayectorias para la dinámica lenta de un dron, en particular de un cuatrirotor. El algoritmo de control propuesto se válida por medio de una cosimulación dentro del ambiente de Windows, entre el simulador para robótica Gazebo y el paquete de simulación MATLAB/Simulink, la comunicación entre ambos simuladores se realiza mediante el meta sistema operativo ROS (Robot Operating System).

Abstract

Unmanned aerial vehicles known as UAVs, have a variety of applications, ranging from security and defense to recreational activities. Most of the commercial drones have the characteristic of being manually piloted by an operator. This work develops a control algorithm for trajectory tracking for the slow dynamics of a drone, in particular of a quadrotor. The proposed control algorithm is validated by means of a co-simulation in the Windows environment, between the robotics simulator Gazebo and the simulation package MATLAB/Simulink, the communication between the two simulators is performed by means of the ROS (Robot Operating System).

Capítulo 1

Introducción

Los vehículos aéreos no tripulados (VANT), comúnmente llamados Drones, se han popularizado en los últimos años debido a su amplia gama de aplicaciones en labores de vigilancia, patrullaje o fotografía aérea entre otras. La importancia de los vehículos aéreos no tripulados ha crecido y se prevé que tendrá más impacto en un futuro cercano [1]. El estudio de estos vehículos tiene sus registros desde sucesos como la primera guerra mundial, en donde el objetivo era el análisis de prototipos no tripulados de aeronaves tripuladas [2]. Desde el ya lejano 1917 hasta la actualidad el diseño de estas aeronaves ha sido objeto de constantes y notables mejoras con cientos de plataformas diferentes en peso, dimensiones y mecánica de vuelo. Hoy en día los VANTs participan en innumerables misiones, tanto civiles como militares.
Actualmente distintos sectores de investigación y desarrollo tecnológico trabajan en aportes ya no solo del diseño físico del VANT (hardware), ahora también con mejoras en su funcionamiento (programación, control, navegación autónoma, etc.). Estas innovaciones impactan no solo al uso profesional, también a usuarios con poca o nula experiencia. La utilidad y fácil adquisición de dichas unidades, ha ido incrementando la necesidad de entregar al consumidor aeronaves que puedan ser utilizadas con facilidad y seguridad.

Los VANTs se pueden clasificar en dos categorías con características diferentes que los hace idoneos para ciertas aplicaciones. La primera clase son los llamados vehícu-

los aéreos de ala fija. Estos vehículos presentan ventajas significativas en la capacidad para el transporte de carga y velocidades de vuelo, sin embargo son incapaces de mantener vuelos estacionarios. La segunda clase son los conocidos como vehículos aéreos de ala rotativa. Su principal característica, en contraste con los vehículos aéreos de ala fija, es la capacidad de mantener vuelos estacionarios. Además, tienen una mejor maniobrabilidad. Las figuras 1 y 2 muestran ejemplos de vehículos aéreos de ala fija y de ala rotativa respectivamente.

Figura 1: VANT de ala fija.

Figura 2: VANT de ala rotativa.

1.1. Cuatrirotores

Los cuatrirotores son aeronaves de ala rotativa equipados con cuatro motores eléctricos tipo brushless y una hélice; pueden realizar maniobras de una manera muy ágil, entre sus ventajas en términos de dinámica de vuelo. Esto hace que puedan ingresar en lugares donde los humanos difícilmente podrían, ya sea por restricciones físicas o de seguridad. Cada una de las aplicaciones requiere trabajo constante por parte de los equipos de desarrollo para implementar herramientas permitan que el robot pueda ejecutar diferentes tareas. Sin embargo, muchas de estas tareas aún requieren control humano y no funcionan de manera autónoma. Por esto, se vuelve de vital importancia desarrollar una estrategia que permita que estos VANTs realicen todo tipo de tareas sin necesidad de intervención humana.

En los últimos años, la tecnología se ha desarrollado a un ritmo frenético. En particular, los avances en la minaturización de componentes electrónicos han traído consigo múltiples avances, donde los Vehículos Aéreos No Tripulados son un claro ejemplo [3]. Los pilotos de las primeras Aeronaves Pilotadas a Distancia (RPA), requerían tener línea de vista con la aeronave, los avances en telecomunicaciones permitieron al piloto tener acceso a la telemetría de vuelo desde un punto remoto. Este avance impulso las aplicaciones que abarcan desde el uso no profesional relacionado con actividades deportivas y de ocio personal, hasta actividades profesionales relacionadas con la logística, topografía, monitorización de espacios abiertos e investigaciones científicas [4] [5] [6].

Con el paso de los años y fundamentalmente con la mejora e invención de nuevas tecnologías como el GPS S y DGPS (GPS diferencial), han ido apareciendo varios modos de utilización hasta llegar al actual modo autónomo, que permite a un dron despegar, realizar cualquier intervención de forma periódica, y aterrizar sin la intervención ni presencia humana, para ello el piloto sólo deberá programar la ruta a través de un software e indicar el momento requerido para la toma de cualquier dato.

1.2. Funcionamieno del cuatrirotor

Cuando un rotor gira, se genera una fuerza de empuje perpendicular al plano de rotación de la hélice. Dicha fuerza de empuje es proporcional al cuadrado de la velocidad angular de la hélice [7].
Para lograr el vuelo estacionario (hover) en un cuatrirotor es necesario que la suma de las fuerzas de empuje de los 4 rotores sea igual al peso del vehículo. Por otro lado, un rotor al girar genera un momento reactivo debido a las fuerzas aplicadas sobre la hélice, este momento reactivo es de sentido contrario al giro de la hélice. Debido a este fenómeno, dos hélices giran en sentido horario y las otras dos giran en sentido antihorario, con el fin de contrarrestar los momentos reactivos que el cuerpo experimenta al hacer girar los rotores; así cada par de motores opuestos giran en el mismo sentido. La figura 1.3 muestra el sentido de giro de las hélices para un cuatrirotor en configuración de cruz.

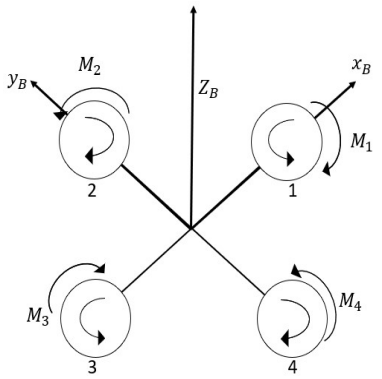

Figura 1-3: Cuatrirotor en vuelo estacionario.

Al variar las velocidades angulares de sus rotores se pueden generar momentos alrededor de los ejes X_B, Y_B y Z_B, esto produce movimiento traslacional cartesiano y rotacional alrededor del eje vertical del cuatrirotor. A partir del cambio en las velocidades angulares de los rotores, se puede realizar los siguientes movimientos:

- **Movimiento en dirección** Z_B: Para lograr un movimiento de ascenso en el cuatrirotor es necesario aumentar la velocidad angular de los cuatro rotores en la

misma proporción. Si se desea un descenso, es necesario disminuir la velocidad de giro de los rotores en la misma proporción.

- **Movimiento en dirección** X_B: Para que el cuatrirotor se mueva a lo largo del eje X_B, es necesario aumentar la velocidad angular del rotor 3 en la misma proporción que se disminuye la velocidad angular del rotor 1, además se debe mantener constantes las velocidades angulares de los rotores 2 y 4, como se muestra en la figura 1-4.a. Si se desea un movimiento en dirección X_B, es necesario aumentar la velocidad angular del rotor 1 en la misma proporción que se disminuye la velocidad angular del rotor 3, además de mantener constantes las velocidades angulares de los rotores 2 y 4, como se muestra en la figura 1-4.b

- **Movimiento en dirección** Y_B: Para que el cuatrirotor se mueva a lo largo del eje Y_B, es necesario aumentar la velocidad angular del rotor 4 en la misma proporción que se disminuye la velocidad angular del rotor 2, además se debe mantener constantes las velocidades angulares de los rotores 1 y 3, como se muestra en la figura 1-5.a. Si se desea un movimiento en dirección Y_B, es necesario aumentar la velocidad angular del rotor 2 en la misma proporción que se disminuye la velocidad angular del rotor 4, además de mantener constantes las velocidades angulares de los rotores 1 y 3, como se muestra en la figura 1-5.b.

- **Giro alrededor del eje** Z_B: Para hacer que el cuatrirotor gire alrededor de Z_B (izquierda), es necesario aumentar la velocidad angular de los rotores 2 y 4 en la misma proporción que se disminuye la velocidad angular de los rotores 1 y 3, como se muestra en la figura 1-6.a. De la misma manera, para hacer que el cuatrirotor gire alrededor de Z_B (derecha), es necesario aumentar la velocidad angular de los rotores 1 y 3 en la misma proporción que se disminuye la velocidad angular de los rotores 2 y 4, como se muestra en la figura 1-6.b.

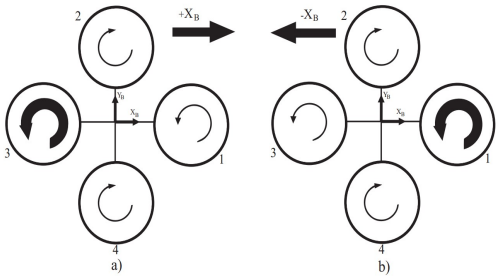

Figura 1-4: Movimiento del cuatrirotor a lo largo del eje X_B.

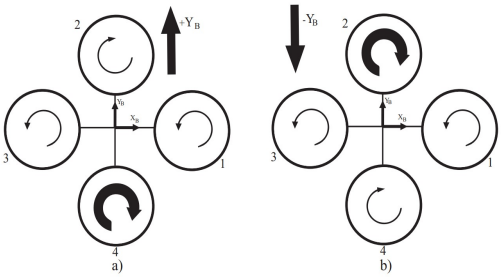

Figura 1-5: Movimiento del cuatrirotor a lo largo del eje Y_B.

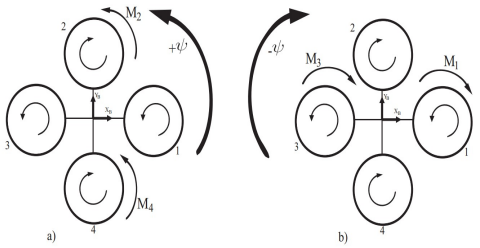

Figura 1-6: Giro del cuatrirotor a lo largo del eje Z_B.

1.3. Estado del arte

Desde el punto de vista de control, el cuatrirotor se ha convertido en una plataforma experimental preferida para evaluar varias técnicas de control. La complejidad que hace interesante el problema del control de un cuatrirotor es estabilizar los 3 grados de libertad cartesianos y 3 rotacionales, a partir de 4 entradas de control.

Para un operador humano sería complicado regular de forma independiente el empuje de cada uno de los rotores para lograr los movimientos descritos. Por lo tanto, es necesario un sistema de control que coordine las variaciones de empuje de cada rotor para generar dichos movimientos. La efectividad del sistema de control está supeditada a la disponibilidad de los estados del cuatrirotor. Por ejemplo, si es posible medir fiablemente la posición y velocidad angular del cuatrirotor es fácil diseñar un sistema de control que controle la orientación, de tal forma que el operador podría enviar referencias de posición angular al sistema de control. Si además es posible medir la velocidad traslacional del vehículo entonces el sistema de control puede modificarse para que el usuario ahora envíe referencias de velocidad traslacional.

La tecnología actual permite medir de forma precisa y con un alto ancho de banda a la posición y velocidad angular de un vehículo utilizando un sistema de referencia de orientación y rumbo (AHRS) basado en la información de una central de medición inercial (IMU). Además, utilizando información visual, también es posible medir la velocidad traslacional del vehículo a un ancho de banda aceptable. El AHRS y el sensor visual pueden ir a bordo del vehículo por lo que se consideran esenciales para navegación autónoma.

Por otro lado, no existe un sensor que pueda ir abordo del vehículo y sin la necesidad de referencias externas permita medir la posición traslacional Cartesiana. Para medir la posición vertical si existen diversos tipos de sensores. Es así que el cuello de botella para que un vehículo pueda realizar operaciones autónomas es determinar su posición Cartesiana; la opción más viable es a través del procesamiento de imágenes. En la actualidad existen algoritmos de procesamiento de imágenes que pueden determinar la posición de un vehículo tal como el algoritmo de mapeo y localización simultáneo (SLAM). No obstante, el ancho de banda al cual se determina la posición es mucho

menor que el de los algoritmos para determinar la orientación y velocidad traslacional.

Por las razones anteriores los drones comerciales siguen un enfoque de control de lazos externos e internos. Los lazos internos de control se encargan de regular los estados que pueden medirse fiable y rápidamente, mientras que los lazos internos se encargan de regular los estados que pueden medirse después de un procesamiento complejo de la información.

1.4. Objetivos

Objetivo General: Diseñar una estrategia de control para seguimiento de trayectorias en un dron comercial y validar el controlador propuesto utilizando una cosimulación en el ambiente Windows utilizando la plataforma Gazebo-MATLAB/Simulink.

Objetivos particulares

- Obtener el modelo dinámico de un dron comercial.
- Diseñar un algoritmo de control para seguimiento de trayectoria.
- Realizar la comunicación en el ambiente Windows del simulador Gazebo y MATLAB/Simulink, utilizando ROS.
- Implementar el control diseñado utilizando el simulador Gazebo y MATLAB/Simulink.

La estructura de este documento se presenta a continuación. Esta tesis consta de 5 capítulos. En el Capítulo 2 se aborda, la descripción del cuatrirotor AR Drone 2.0, una explicaión de ROS (Robot Operating System), el simulador de ambiente real Gazebo, la paqueteria del AR Drone 2.0 autonomy y la herramienta de Matlab Robot system toolbox. Posteriormente en el Capítulo 3 se describe el modelo dinámico de un cuatrirotor bajo la acción de los lazos de control que se utilizan comercialmente y se plantea el desarrollo del controlador para el seguimiento de trayectoria para la posición traslacional y angular, alrededor del eje vertical, del cuatrirotor. En el Capítulo 4 se presenta

el desarrollo y resultado de las simulaciones numéricas necesarias para la validación de este trabajo, estas mediante Matlab/Simulink, ROS y el simulador Gazebo. Finalmente, en el capítulo 5 se presentan las conclusiones y trabajo futuro.

Capítulo 2

Descripción de la plataforma experimental

2.1. Ar Drone 2.0

Figura 2-1: Dron de ala rotativa conocido como AR Drone 2.0 de la compañia Parrot.

EL Ar Drone 2.0 es un cuatrirotor comercial en configuración X que puede pilotarse remotamente a través de un canal de comunicación WiFi. El AR Drone crea una red WI-FI con un ESSID[1] usualmente llamado ardrone2-XXX,normalmente este autoasigna la dirección IP "192.168.1.1".

[1]Extended Service Set Identifier por sus siglas en ingles. Identifica una red inalámbrica o un grupo de redes inalámbricas.

La latencia de transmisión de los comandos de control es crítica para la experiencia del usuario. Estos comandos se envian generalmente a 30 veces por segundo [8].

La estructura física del Ar Drone esta conformada por:

- **Estructura inferior:** El cuerpo del AR Drone está fabricado en polipropileno expandido, un plástico muy ligero y de gran resistencia. El polipropileno expandido soporta la batería, que está situada sobre el cuerpo principal del drone y cuenta con un recubrimiento de espuma que absorbe las vibraciones del motor. La batería está ligeramente desplazada hacia la parte posterior del dispositivo para mantener el centro de gravedad del AR Drone. El soporte de la batería también incluye un puerto USB que permite guardar vídeos y fotos directamente en un dispositivo USB.

- **Sensores de movimiento:** La placa de navegación tiene un telémetro de ultrasonido para medir la altura del AR Drone hasta 6 metros. El telémetro proporciona mediciones a una frecuencia de 25 Hz.
  En el centro de gravedad del AR Drone se ubica un acelerómetro digital de tecnologia MEMS (microsistema electromecánico) de 3 ejes que opera en un rango de +/- 2g. La digitaliza mediante un convertidor CAD de 10 bits integrado. Los datos se envían al microcontrolador.
  Un giroscopio MEMS para los ejes con piezoeléctrico de precisión para su estabilidad y control del rumbo. Ambos sensores miden hasta 500 °/s. Estos sensores analógicos se digitalizan a través del convertidor CAD de 12 bits del microcontrolador.

- **Transmisión de video:** El AR Drone 2.0 cuenta con una camara frontal CMOS con una lente de ángulo de 90 grados. Las resoluciones de imagen es de 360p (640x360) o de 720p (1280x720), y la velocidad de fotograma de transmisión de video se puede ajustar entre 15 y 30 FPS.

El AR Drone 2.0 incluye una RAM de 1 GB y 200 MHz. Tiene un chip WiFi de Atheros y un puerto USB para instalar las actualizaciones y futuros desarrollos del dispositivo. Tiene un sensor de presión y su cámara inferior opera a 60 fps (frames per second).

- **Batería:** Cuenta con una batería tipo LiPo (Litio-ion polímero) de 1000mAh que le permite al drone realizar un tiempo de vuelo de 12 minutos, pudiendo alcanzar velocidades mayores a 5m/s. está es recargable. El AR Drone 2.0 monitorea el voltaje de la batería y convierte esta tensión en un porcentaje de la vida util de la batería. Cuando el drone detecta un voltaje bajo de la batería, envía un mensaje de advertencia al usuario, luego este realiza un aterrizaje de emergencia.

- **Cruz central:** Es la columna vertebral del AR Drone 2.0. Los tubos están fabricados en fibra de carbono y los 4 soportes de los motores en plástico tipo PA 66. Estos materiales han sido seleccionados para proporcionar ligereza y resistencia a la cruz central.

  La estructura permite una fácil extracción y ensamblaje de los motores y los soportes de los motores incorporan dos cojinetes planos autolubricantes en bronce.

  La cruz central tiene un par de cables. Uno para alimentar los motores y otro para controlar la velocidad de rotación. Estos cables están divididos en cuatro y terminan en conectores para cada motor.

- **Motores:** El AR Drone 2.0 funciona con motores sin escobillas con tres fases de corriente que son controladas por un microcontrolador. El AR Drone detecta automáticamente el tipo de motores y ajusta los controles del motor, también detecta si los motores están girando o están detenidos. En caso de que una hélice giratoria encuentre algún obstáculo, El AR Drone detecta si alguna de las hélices está bloqueada y en ese caso el microcontrolador lo detecta y detiene todos los motores de inmediato. Este sistema de protección evita golpes repetidos.

  El motor tiene una potencia de 15 vatios y realiza 28.000 revoluciones por minuto (rpm) en vuelo estático. Esta velocidad corresponde a 3.300 rpm en la hélice

debido a la reducción por el sistema de engranajes. El rango de velocidades de los motores oscila entre 10.350 y 41.400 rpm.

2.2. ROS (”*Robot Operating System*”)

Robot Operating System, proporciona una estructura de programación para el desarrollo enfocado hacia la robótica de codigo abierto. Este sistema se utiliza en aplicaciones dirigido a la investigación, también cuenta con la abstracción de hardware, el control de dispositivos de bajo nivel, la implementación de funciones de uso común, el paso de mensajes entre procesos y la gestión de paquetes. También proporciona herramientas y bibliotecas para obtener, construir, escribir y ejecutar código en varias computadoras.

El Sistema Operativo (ROS) ha revolucionado la comunidad de desarrolladores, proporcionándole un conjunto de herramientas, infraestructura y las mejores prácticas para construir nuevas aplicaciones y robots. ROS permite el diseño de manera individual de procesos que se pueden acoplar de manera flexible en tiempo de ejecución. Estos procesos se pueden agrupar en paquetes que pueden ser fácilmente compartidos y distribuidos, con lo que se logra la reutilización de código. Actualmente ROS solo puede ser ejecutado en plataformas basadas en linux.
En el año 2007 el laboratorio de inteligencia artificial de Standford empezó a desarrollar ROS bajo el nombre de Switchyard y posteriormente en 2008, el desarrollo continuó a cargo del instituto de investigación de robótica llamado Willow Garage. Actualmente ROS permite tener acceso a los servicios estándar de cualquier sistema operativo.
ROS proporciona las siguientes capacidades para la programación de robots:

- **Comunicación entre procesos:** ROS proporciona una interfaz de paso de mensajes para la comunicación entre dos procesos. Por ejemplo, una cámara procesa una imagen y encuentra sus coordenadas, luego estas coordenadas son enviadas a un proceso de seguimiento. El proceso del rastreador hace el seguimiento de la imagen mediante el uso de motores.

- **Independecia de lenguajes de programación:** La ventaja del ROS es que so-

porta los lenguajes de programación más populares utilizados en la programación de robots, incluyendo C++ y Python. Existen bibliotecas experimentales para lenguajes como Java, entre otros. ROS proporciona bibliotecas de cliente para estos lenguajes, lo que significa que el programador puede obtener las funcionalidades ROS en los lenguajes mencionados.

- **Amplias herramientas y simuladores:** ROS está construido con muchas herramientas de línea de comandos y GUI para depurar, visualizar y simular aplicaciones de robótica. Estas herramientas son muy útiles para trabajar con un robot. Por ejemplo, la herramienta Rviz 11 se utiliza para la visualización con cámaras, escáneres láser, inercia unidades de medida, y así sucesivamente. Para trabajar con simulaciones de robots, existen simuladores como Gazebo.

ROS no es un sistema operativo real, es un meta-sistema operativo que porporciona ciertas funcionalidades de sistema operativo como: control de dispositivos de bajo nivel, administración de paquetes y abstracción de hadware. Esta abstracción de hadware permite a los programadores, programar algun dispositivo ,lo cual tiene como ventaja que dicho codigo para ese dispositivo funcione de la misma manera con diferentes proveedores.

2.3. Arquitectura y entorno de ROS

La estructura de ROS se basa en un ciclo de publicación-suscripción de mensajes.

Conceptos de ROS

- **Nodos:** Estos son procesos que realizan una determinada función,estas utilizan una API de ROS para realizar cálculos.

- **Tópico:** Un tópico son buses donde los nodos se pueden conectar para enviar y recibir mensajes. Un nodo puede publicar o suscribir cualquier número de temas.

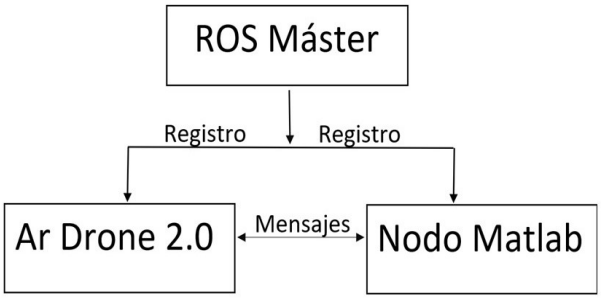

Figura 2-2: Comunicación entre nodos de ROS.

- **Maestro:** También conocido como **ROS Master**, se encarga de almacenar los tópicos y servicios de información y registros de los nodos. También realiza devoluciones de llamadas a los nodos cuando intercambian información de esta manera crean una comunicación dinámica.

- **Mensaje:** Estos son enviados a través de los topicos. Existen mensajes basados en tipos de datos primitivos de tal manera que los usuarios pueden escribir sus propios mensajes. Los mensajes pueden contener estructuras y arreglos de manera arbitraria.

- **Sevicios:** Es similar al modelo de publicación/subscripción descrito anteriormente, pero en este caso se tiene un proceso de petición/respuesta. Una llamada de servicio es una función que puede llamar cada vez que un nodo de cliente envía una solicitud.

- **Servidor de parámetros:** Un programa que normalmente se ejecuta junto con el maestro de ROS, donde el usuario puede almacenar varios parámetros o valores en este servidor y todos los nodos tiene acceso a ellos.

- **Bolsas (Bags):** Son un mecanismo para guardar y reproducir datos de mensajes de ROS

Todos los nodos de ROS se comunican de manera directa entre ellos, el maestro proporciona información de búsqueda, al igual que un servidor DNS. Los nodos que se suscriben a un tópico requieren de una conexión con los otros nodos que realizan la

publicación de dicho tópico. de tal manera que se establesca una conexión como se muestra en la figura 2.3. La comunicación entre nodos se realiza por un protocolo de comunicación TCPROS. Este es una capa de transporte para los mensajes y servicios de ROS y utiliza los conectadores estándar TCP/IP.

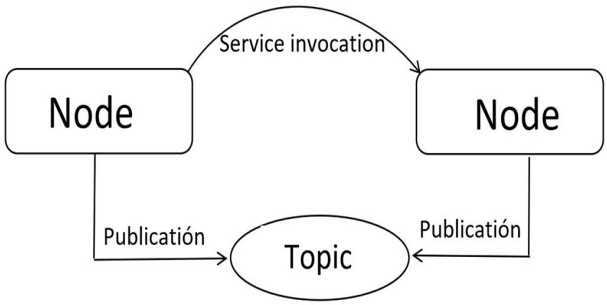

Figura 2-3: Diagrama de flujo de información entre nodos de ROS.

2.4. Ejecución y comandos de ROS

ROS cuenta con varias herramientas que permiten simular y visualizar entornos para el uso de investigación en la rama de la robotica, también permite ejecutar varios nodos en un mismo tiempo y observar el flujo de información entre ellos. Dentro de estas herramentas se describen las mas utilizadas:

- **roscore:** Ejecuta el nucleo de ROS, permitiendo asi la iniciar la comunicación entre nodos, servicios, parámetros, mediante el nodo maestro.
- **roscd:** Permite acceder a un paquete o a una localización en común.
- **rosdep:** Permite instalar dependencias del sistema.
- **rosrun:** Permite ejecutar un archivo ejecutable en un paquete arbitrario.
- **rosmsg:** Permite obtener la información de un tipo de mensaje.
- **rosnode:** Muestra la conectividad la lista de la conectividad de nodos en ejecución.

numero	Comandos
rostopic bw	Muestra el ancho de banda consumido por un tópico.
rostopic delay	Retardo de la visualización de un tópico
rostopic echo	Muestra los mensajes del tópico por la pantalla
rostopic find	Busqueda de un tópico por tipo
rostopic hz	Muestra la velocidad de publicación de un tópico
rstopic info	Muestra la información de un tópico que se encuentre activo
rostopic list	Lista de tópicos disponibles
rostopic pub	Publica un mensaje en un tópico

Tabla 2.1: Comandos de la herramienta rostopic.

- **roslaunch:** Puede ejecutar archivos con formato .launch, en ellos se puede configurar varios nodos y de esta manera ejecutarlos al mismo tiempo.
- **rosservice:** Muestra el tiempo de ejecución de un servicio e imprime mensajes que se envían al tópico.
- **rosparam:** Obtiene y establece valores del servidor de parámetros desde la ventana de comandos.
- **rostopic:** Muestra el tiempo de ejecución de los tópicos e imprime los mensajes que se envian a un tópico.
- **rosversión:** Muestra la versión de ROS instalada.

Con estas diversas herramientas proporcionadas por ROS, se pueden obtener diversas características sobre nodos, tópicos y servicios, dentro de cada una de estas herramientas existen varios comandos, con los cuales se pueden obtener cierta información. Para este trabajo de investigación se utilizará la herramienta rostopic. En la tabla 2.1 se muestran los diferentes comandos que contiene.

Dentro del servicio Maestro de ROS se encuentra la heramienta de rosnode, la cual permite obtener la información de los nodos que se encuentran activos asi como también los comandos correspondientes a esta herramienta, permite borrar, terminar y/o probar

la conectividad de un nodo. En la tabla 2.2 se muestran los diferentes comandos que contiene la herramienta rosnode.

Comando	Descripción
rosnode ping	Testea la conexión entre nodos
rosnode cleanup	Borra la información de los nodos que no esten activos
rosnode info	Muestra la información de un nodo que se encuentre activo
rosnode list	Lista de nodos activos
rosnode kill	Detiene la ejecución de un nodo
rosnode machine	Muestra una lista de nodos que se encuentran ejecutandose en una o varias maquinas.

Tabla 2.2: Comandos de la herramienta rosnode.

2.5. Gazebo

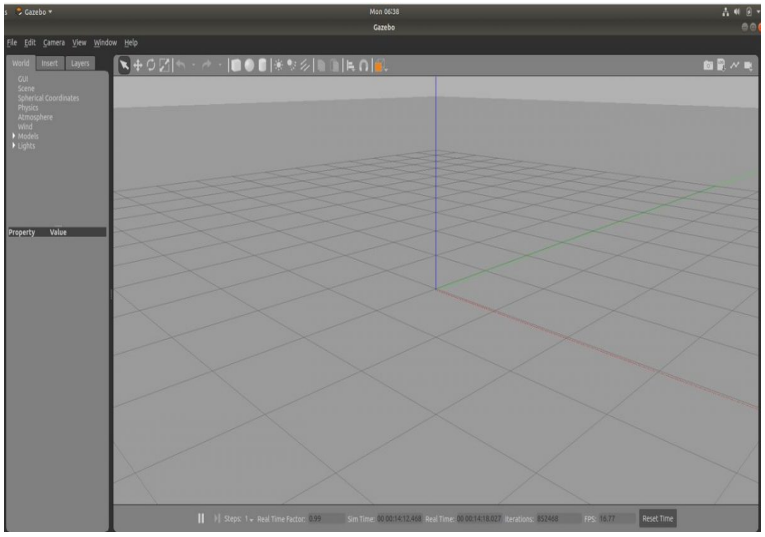

Figura 2-4: Simulador Gazebo.

Gazebo es un simulador 3D muti-robot que ofrece la posibilidad de simular sistemas complejos de robots con sensores de una manera precisa y eficiente, tiene como ventaja

emular parámetros físicos del mundo real. Genera, tanto la realimentación realista de sensores, como las interacciones entre los objetos físicamente plausibles, incluida una simulación precisa de la física de cuerpo rígido.

A partir de esta definición, se entiende que Gazebo es una herramienta gratuita de simulación, que resulta de vital importancia para la prueba de los algoritmos que van siendo elaborados y configurados y, así, brinda la posibilidad de visibilizar los aciertos y errores en los que se pueda incurrir en el proceso de desarrollo del robot de que se trate. De esta forma, es posible indicar que la herramienta ROS-Gazebo constituye una completa opción de programación robótica, de código abierto y con una importante gama de posibilidades para emprender desarrollos en el campo de la robótica.

Gazebo tiene una arquitectura cliente/sevidor que utiliza las herraminetas de ROS de suscripción y publicación de mensajes en los tópicos y de esta manera generar procesos de comunicación entre los diferentes componentes en el sistema. Las características importantes de este simulador son las siguientes:

- **Graficos 3D:** Considera aspectos de iluminación, sombras y texturas de alta calidad.
- **Sensores y ruido:** Genera sensores de datos con la posibilidad de agregar y ajustar diferentes patrones de ruido.
- **Plugins:** Desarrollo de plugins personalizados para robots, sensores y controles.
- **Modelos de robots:** Gazebo ofrece una gran variedad de diferentes modelos de robots existentes.

La arquitectura de Gazebo está basada en el motor Open Dynamics Engine [9] [10] creado por Russel Smith; es un motor de física ampliamente utilizado en la comunidad de código abierto. Está diseñado para simular la dinámica y la cinemática asociadas a cuerpos rígidos articulados. Este motor incluye muchas características, como numerosas articulaciones, detección de colisiones, funciones de masa y rotación, etc. de colisiones, funciones de masa y rotación, y muchas geometrías, incluidas mallas triangulares arbitrarias; aunque su desarrollo final se atribuye a Andrew Howard y Nate Koenig, por sus investigaciones en la Universidad del Sur de California [11] .

2.6. AR Drone autonomy

El paquete AR Drone autonomy es un driver de ROS creado para el cuatrirotor Parrot Ar Drone 2.0. Este driver está basado en la versión oficial AR Drone SDK 2.0.1 y es una derivación del driver AR Drone Brown. Este paquete es desarrollado en el Autonomy Lab de la universidad Simon Fraser.

Es posible utilizar el paquete AR Drone autonomy para planear y ejecutar tareas mediante la descripción de la trayectoria y orientación que el drone debe seguir. Dentro del paquete AR Drone autonomy se encuentra el nodo AR Drone driver que se describe en la figura 2.4.

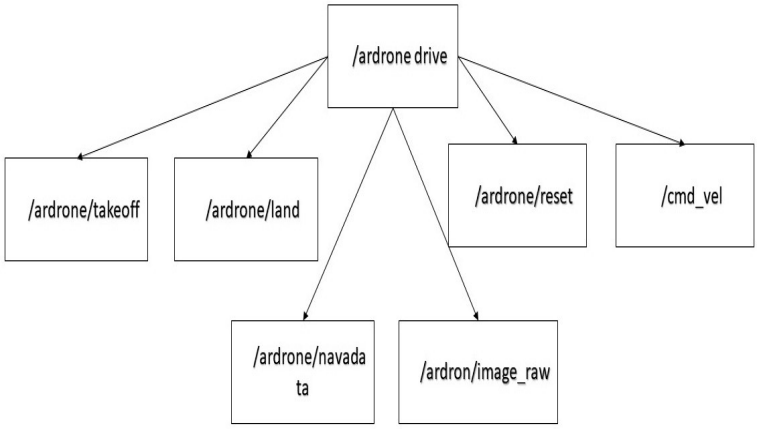

Figura 2.5: Paqueteria del Ar Drone autonomy.

- **ardrone drive:** Provee una interfaz de control con el dron, también se encarga de mantener una conexión automática enviando un paquete específico de datos.
- **cmd vel:** Un mensaje de tipo geometry msgs/Twist es usado para controlar la velocidad (traslacional y angular) del AR Drone. Todos los valores deben ser especificados entre -1 y 1.
- **ardrone image raw:** Publica un mensaje con las imágenes de la cámara.

- **takeoff:** Se envía un mensaje del tipo std msgs/Empty, es decir un mensaje en blanco. El AR Drone despegará.
- **land:** Se envía un mensaje del tipo std msgs/Empty, es decir un mensaje en blanco. El AR Drone aterrizará.
- **navdata:** Proporciona los datos transmitidos por los sensores del dron.[2]

2.7. Robot System Toolbox

Robotics System Toolbox es una herramienta de MATLAB que fue añadida en la versión de 2015, esta caja de herramientas proporciona una interfaz entre el software MATLAB/Simulink y ROS, esta interfaz permite probar, verificar y simular utilizando las herramientas de ROS. Este toolbox es compatible con el codigo C++, lo cual permite generar un nodo de ROS a partir de MATLAB. Con la ayuda de este nodo es que permite trabajar con mensajes de ROS, publicar y susbcribirse a los tópicos, acceder a servicios entre otros, es decir, permite manejar ROS desde MATLAB.

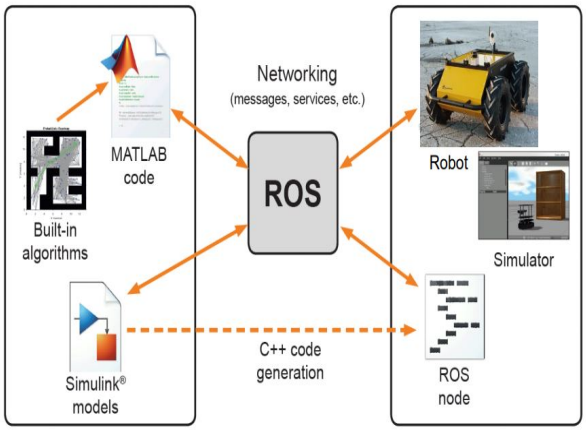

Figura 2-6: Diagrama de comunicación entre MATLAB y GAZEBO con ROS.

[2]En los tutoriales de funcionamiento de esta utilieria se dice que se envían al dron comandos de ángulo, sin embargo en realidad se envían comandos de velocidad.

Capítulo 3

Modelado y control del dron comercial AR Drone 2.0

3.1. Modelo dinámico del AR Drone 2.0

El modelo dinámico del cuatrirotor supone una estructura mecánica rígida en forma de cruz con un rotor montado en cada extremo. Para describir el modelo de movimiento del cuatrirotor, se consideran dos ejes de referencia, el eje inercial (I) y el eje cuerpo (B), véase la figura 3-1. En coordenadas mixtas, cuerpo e inercial, el siguiente conjunto de ecuaciones diferenciales 3.1, describe el modelo dinámico del cuatrirotor [12].

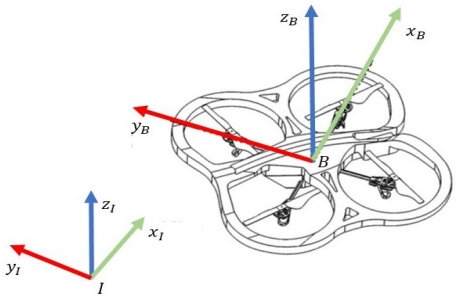

Figura 3-1: Marcos de referencia X_I,Y_I,Z_I y cuerpo X_B,Y_B,Z_B.

$$\begin{aligned} \dot{X} &= RV^B \\ m\dot{V^B} + m\Omega \times V^B &= R^\top F_g^I + F_T^B + F_A^B \\ \dot{R} &= R\Omega^\wedge \\ J\dot{\Omega} &= -\Omega^\wedge J\Omega + M_T^B + M_A^B \end{aligned} \tag{3.1}$$

Donde $X = \begin{bmatrix} x & y & z \end{bmatrix}^\top$ es la posición translacional con respecto al marco de ejes inerciales, $V^B = \begin{bmatrix} u & v & w \end{bmatrix}^\top$ es la velocidad de traslacional expresada en ejes cuerpo, R es la matriz de rotación de ejes cuerpo a ejes inerciales y $\Omega = \begin{bmatrix} p & q & r \end{bmatrix}^\top$ es la velocidad de rotación expresada en ejes cuerpo. Además $(.)^\wedge$ es un mapa de $\mathbb{R}^3$ a la álgebra de Lie $SO(3)$ definida como [13]

$$SO(3) = \{\omega_1^\wedge \in \mathbb{R}^{3\times 3} | \omega_1 \times \omega_2 = \omega_1^\wedge \omega_2, \forall \omega_1, \omega_2 \in \mathbb{R}^3\}$$

Las fuerzas externas que actúan sobre la estructura mecánica del cuatrirotor estan dadas por

$$F_g^I = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix},\ F_T^B = \begin{bmatrix} 0 \\ 0 \\ T_T \end{bmatrix},\ F_A^B = \begin{bmatrix} q_d S_f C_x \\ q_d S_f C_y \\ q_d S_s C_z \end{bmatrix} \tag{3.2}$$

Mientras que los momentos externos son

$$M_T^B = \begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix},\ M_A^B = \begin{bmatrix} q_d S_f l C_{m_x} \\ q_d S_f l C_{m_y} \\ q_d S_s l C_{m_z} \end{bmatrix} \tag{3.3}$$

Donde T_T el empuje total de los cuatro rotores y M_x, M_y y M_z son los momentos alrededor de los ejes cuerpo generados por los rotores.

$$q_d = \frac{1}{2}\rho V^2$$

es la presión dinámica, con ρ la densidad del aire y

$$V = \sqrt{u^2 + v^2 + w^2}$$

adicionalmente, S_f y l son características geométricas del cuatrirotor C_x, C_y y C_z son los coeficientes de fuerza aerodinámica y C_{m_x}, C_{m_y} son los coeficientes de momentos

aerodinámicos. En términos de los ángulos de Tait-Bryan [14]

$$\Phi = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}$$

con ϕ, θ ψ los ángulos de balanceo, cabeceo y guiñada, respectivamente, la matriz de rotación R se lee como

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

Los pilotos automáticos internos del vehículo aéreo modelan la respuesta del sistema a distintos niveles en función de los sensores disponibles. Por lo tanto, si se miden los estados rotacionales, el autopiloto interno puede modelar la respuesta del vehículo aéreo como un sistema de segundo orden para la dinámica de traslación y como un sistema de primer orden para el resto de la dinámica de rotación. Por lo tanto si el piloto automático ordena, de forma robusta y arbitrariamente rápida, esto es

$$\lim_{t\to T}(\phi) = \phi_d,\ \lim_{t\to T}(\theta) = \theta_d,\ \lim_{t\to T}(r) = r_d \tag{3.4}$$

para algún $T \in \mathbb{R}$ acotado, con ϕ_d, θ_d y r_d las referencias deseadas para la velocidad angular de balanceo, cabeceo y guiñada respectivamente, la dinámica del cuatrirotor de (1) se convierte en

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} c\theta_d c\psi_d & c\psi_d s\theta_d s\phi_d - s\psi_d c\phi_d & c\psi_d s\theta_d c\phi_d + s\psi_d s\phi_d \\ s\psi_d c\theta_d & s\psi_d s\theta_d s\phi_d + c\psi_d c\phi_d & s\psi_d s\theta_d c\phi_d - s\phi_d c\psi_d \\ -s\theta_d & c\theta_d s\phi_d & c\theta_d c\phi_d \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix},$$

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} vr_d \\ -ur_d \\ 0 \end{bmatrix} + g \begin{bmatrix} -s\theta_d \\ c\theta_d s\phi_d \\ c\theta_d c\phi_d \end{bmatrix} + \frac{T_T}{m} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + \frac{1}{m} \begin{bmatrix} q_d S_f l C_x \\ q_d S_f l C_y \\ q_d S_s l C_z \end{bmatrix},\ \dot{\psi} = r_d \tag{3.5}$$

En el modelo reducido del cuatrirotor 3.5 las entradas de control son ϕ_d, θ_d, r_d y T_T. Si, además, se mide la velocidad traslacional V^B, el piloto automático puede modificarse

para conformar a la dinámica traslacional como un conjunto de sistemas de primer orden. En este caso, los comandos del piloto automático, robustos y arbitrariamente rápidos.

$$\lim_{t\to T_0} u = u_d,\ \lim_{t\to T_0} v = v_d,\ \lim_{t\to T_0} w = w_d$$

para algún $T_0 > T$ acotado, siendo u_d, v_d y w_d las referencias de velocidad. Por lo tanto, la dinámica restante del cuatrirotor se lee como:

$$\begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix} = \begin{bmatrix}\cos\psi & -\sin\psi\\ \sin\psi & \cos\psi\end{bmatrix}\begin{bmatrix}u_d\\ v_d\end{bmatrix},$$
$$\dot{z} = w_d,$$
$$\dot{\psi} = r_d \tag{3.6}$$

Este es el caso del vehículo aéreo utilizado en este trabajo. Utilizando la información del sistema de referencia de altitud, así como de los algoritmos de flujo óptico y SLAM, el controlador interno modela el vehículo aéreo como una dinámica de primer orden expresada en la ecuación 3.6.

La documentación disponible sobre Ar Drone Autonomy especifica que las señales que se envían para controlar la posición cartesiana son ángulos de referencia ϕ_d, θ_d. No obstante, de ser este el caso, el piloto tendría que lidiar con la dinámica no lineal de segundo orden descrita en la ecuación (5). Debido a que con la cámara inferior, el piloto automático del Ar Drone, puede calcular su velocidad traslacional del dron a través del flujo óptico en este trabajo de tesis se considera que las entradas que envía Ar Drone Autonomy son referencias de velocidad. Un indicador más que justifica esta consideración es que el Ar Drone Autonomy emplea un twist para enviar la información en Gazebo;un twist es un vector de velocidades lineales y angulares.

3.2. Diseño del control de trayectoria

Una vez obtenido el modelo cinemático del cuatrirotor así como también las referencias de la velocidad, definidas anteriormente, se procede a el diseño del control de trayectoria de la siguiente manera.

El error de seguimiento de trayectoria se define como la diferencia entre la posición real menos la posición que se desea seguir, esto es

$$\tilde{X} = X - X_d$$

con

$$X = \begin{bmatrix} x \\ y \end{bmatrix}, \quad X_d = \begin{bmatrix} x_d \\ y_d \end{bmatrix}$$

con x_d e y_d funciones del tiempo que definen a la trayectoria deseada. Tomando al modelo dinánico reducido del cuatrirotor 3.6 puede verificarse fácilmente que

$$\dot{\tilde{X}} = R(\psi)V^B - \dot{X}_d \tag{3.7}$$

donde

$$R(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi \\ \sin\psi & \cos\psi \end{bmatrix}, \; V^B = \begin{bmatrix} u_d \\ v_d \end{bmatrix}$$

Se definen los errores a lo largo y alrededor del del eje vertical, respectivamente, como sigue

$$\begin{aligned} \tilde{z} &= z - z_d \\ \tilde{\psi} &= \psi - \psi_d \end{aligned} \tag{3.8}$$

De manera que

$$\begin{aligned} \dot{\tilde{z}} &= w_d - \dot{z}_d \\ \dot{\tilde{\psi}} &= r_d - \dot{\psi}_d \end{aligned} \tag{3.9}$$

A partir de las ecuaciones (3.7) y (3.9) se definen las velocidades deseadas como

$$\begin{aligned} V^B &= R(\psi)^\top \left(-K\tilde{X} + \dot{X}_d\right) \\ w_d &= -k_z\tilde{z} + z_d \\ r_d &= -k\psi\tilde{\psi} + \psi_d \end{aligned} \tag{3.10}$$

con $K \in \mathbb{R}^{2\times 2}$ una matriz positiva definida y $k_z, k_\psi \in \mathbb{R}$ constantes positivas.

El modelo dinámico reducido del cuatrirotor (3.6) en lazo cerrado con la ley de control en (3.10) se lee como

$$\begin{aligned} \dot{\tilde{X}} &= -K\tilde{X} \\ \dot{\tilde{z}} &= -k_z\tilde{z} \\ \dot{\tilde{\psi}} &= -k_\psi\tilde{\psi} \end{aligned} \tag{3.11}$$

un sistema de ecuaciones lineales con convergencia exponencial al punto de equilibrio $\tilde{X} = 0$, $\tilde{z} = 0$ y $\tilde{\psi} = 0$.

Las señales que se envían al cuatrirotor son:

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = R(\psi)^\top \left(-K\tilde{X} + \dot{X}_d \right)$$
$$u_3 = -k_z \tilde{z} + z_d$$
$$u_4 = -k_\psi \tilde{\psi} + \psi_d$$

Capítulo 4

Simulaciones numéricas y resultados

En este capítulo se muestra como realizar el nodo entre matlab y ROS, así como también se describe el diagrama a bloques de Simulink y finalmente se muestra la simulación del control de la trayectoria para el Ar Drone 2.0.

4.1. Conexión entre MATLAB y Gazebo

Para las simulaciones del control de trayectoria del dron se ocupan dos computadoras; una con los programas previamente instalados y otra donde se encuentra el software de MATLAB. También es importante que ambas computadoras se encuentren conectados a una misma red de internet.
Para realizar el nodo de ROS entre MATLAB/Simulink y Gazebo se debe correr en la terminal de ubuntu el comando **roscore** el cual como se mencionó anteriormente, inicializa el nodo de ROS como se muestra en la figura 4-1.

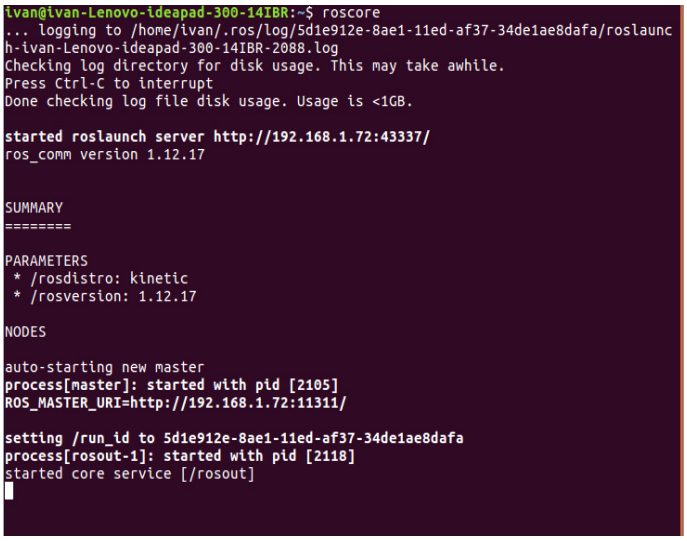

Figura 4-1: Inicialización del nodo ROS con el comando **roscore.**

Por otro lado en la ventana de comandos de matlab MATLAB el nodo con ROS se inicializa con el comando **rosinit 'xxx.xxx.xxx'** donde en **'xxx.xxx.xxx.xxx'** se escribe la dirección IP de la pc con el sistema operativo ubuntu.
Para comprobar que la conexión entre ambos programas esta realizada, MATLAB cuenta con las herramienta Test Connection y Connect to Robot, con la cual verifica dicha conexión. Como un ejemplo, se muestra en la figura 4-2, utilizando la herramienta de test connectión donde se puede observar que no existe una conexión entre matlab y ROS ya que la dirección IP de la pc con el sistema operativo ubuntu no se encuetra configurado de manera correcta.

Para realizar la correcta configuración de la IP, se utiliza la herramienta de Connect to Robot, donde simplemente se tiene que ingresar la dirección IP de la computadora con el sistema operativo de ubuntu asi como también el nombre de usuario y la contraseña, como se muestra en la figura 4-3.
Una vez realizado esta configuración, nuevamente se puede comprobar que la conexión esté hecha de manera correcta como se se observa en la figura 4-4.

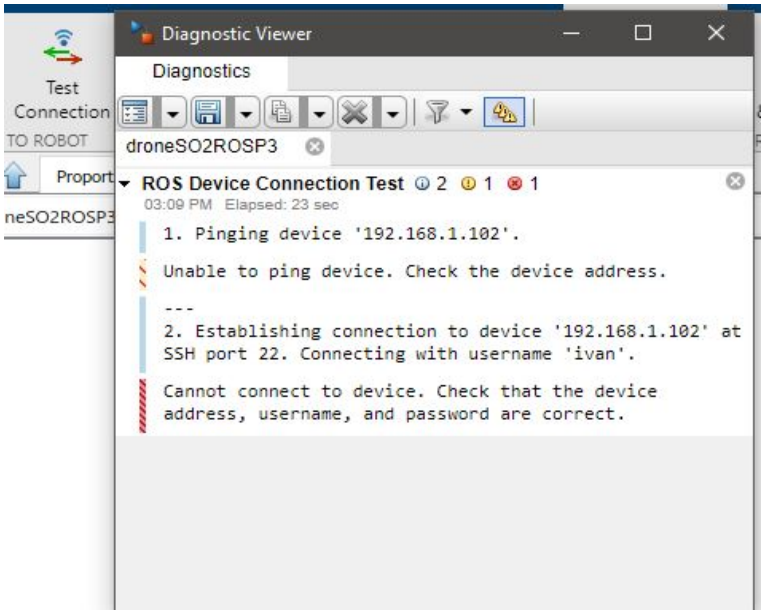

Figura 4-2: Comprobación de conexión de MATLAB y ROS.

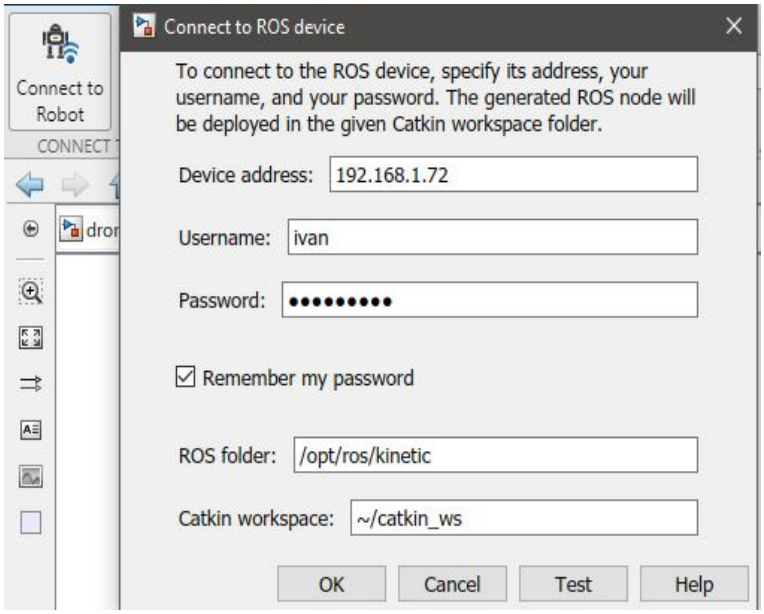

Figura 4-3: Configuración de IP para la conexión de MATLAB y ROS.

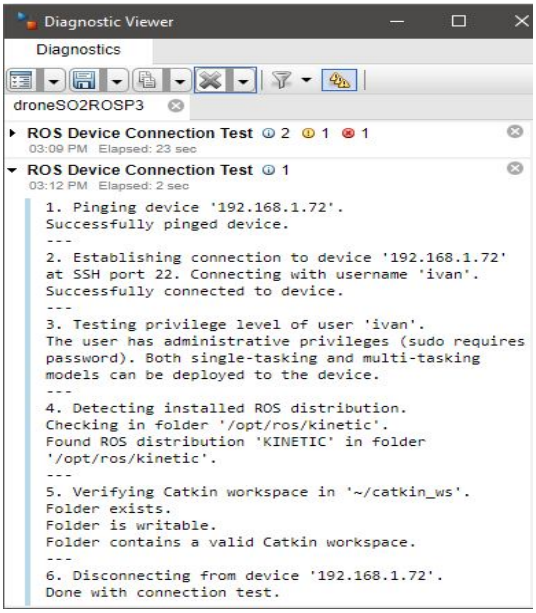

Figura 4-4: Comprobación de conexión de MATLAB y ROS 2.

4.2. Descripción del diagrama de Simulink

A continuación en la figura 4-5 se muestra el diagrama a bloques de simulink utilizado para el envío y recolección de datos entre MATLAB-ROS-Gazebo, también se muestra el bloque del controlador diseñado para el seguimiento de trayectoria del dron.

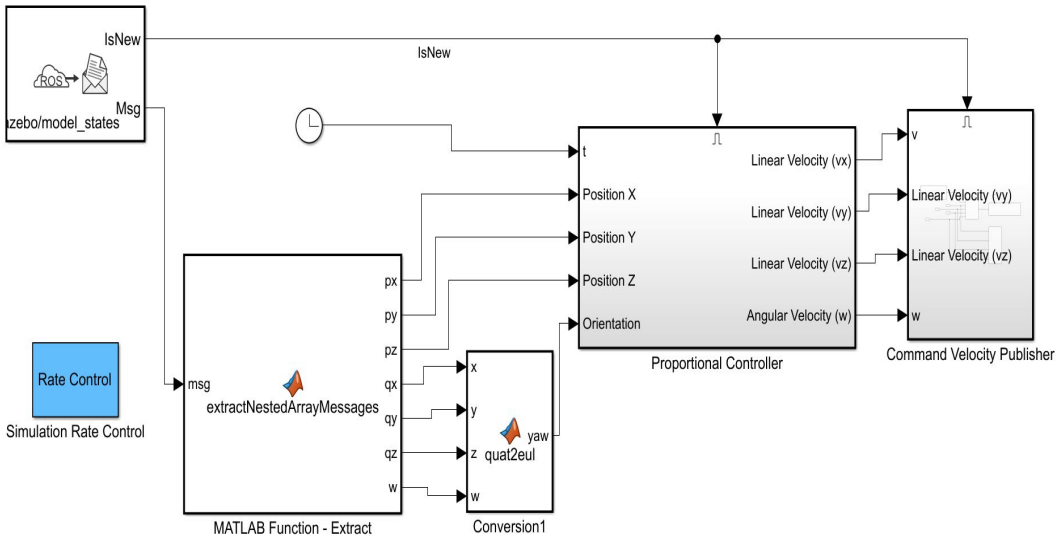

Figura 4-5: Diagrama a bloques Matlab/Simulink.

El bloque Subscribe mostrado en la figura 4-6 crea un bus no virtual de Simulink que corresponde al tipo de mensaje ROS especificado. El bloque utiliza el nodo del modelo Simulink para crear un suscriptor ROS para un tema específico.

En cada paso de la simulación, el bloque comprueba si hay un nuevo mensaje disponible en el tema específico y lo convierte en una señal de bus de Simulink. El puerto msg emite este nuevo mensaje. Si no hay un nuevo mensaje disponible, msg emite el último mensaje ROS recibido.

En este bloque se suscribe a el tópico model states, este tópico manda las cordenadas de la posición x, y, z, y la orientación en cuaterno p, q, r y w, del dron.

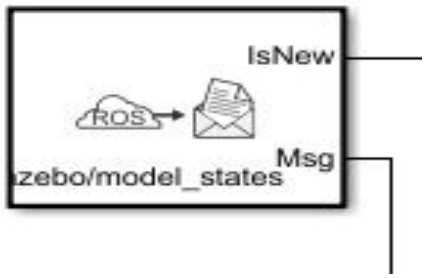

Figura 4-6: Bloque Subscribe.

Una vez que se reciben los datos de la posición y la orientación del dron, estas llegan al bloque mostrado en la figura 4-7, este bloque es definido por el usuario, lo que se realiza en este bloque es obtener la posición traslacional del dron (x, y, z) y la orientación (ϕ, θ, ψ), también se debe indicar en que posición en el espacio dentro de Gazebo se encuentra el dron, esto se verifica con el comando **rostopic echo/gazebo/model states**. En las salidas de este bloque se obtienen de forma individual dichas posiciones del dron $p_x = x$, $p_y = y$, $p_z = z$, estas posiciones se pueden trabajar sin ningun problema, pero ya que para la orientación del dron se da en forma de cuaterno, las posiciones qx ,qy, qz, w, se le aplica una conversión a ángulos de Euler con la función especial de matlab **quat2eul** esto se muestra en la figura 4-8

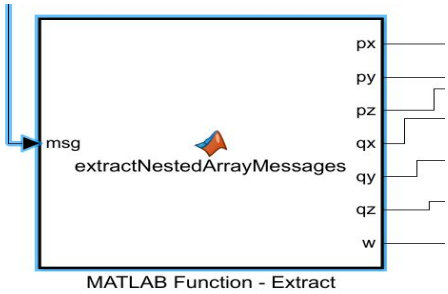

Figura 4-7: Bloque definido por el usuario.

Figura 4-8: Bloque de conversión de cuaterno a ángulos de euler.

Una vez que se tienen la orientación del dron en ángulos de Euler, estos datos, como los de posición de dron, son los que se ocupan para el controlador. Estos datos entran en el siguiente bloque que se muestra en la figura 4-9, donde se tiene un tiempo t, las posiciones x,y,z y la orientación w respectivamente. Y en salidas de este bloque se mandan las velocidades u_d,v_d,w_d y una velocidad angular r_d.

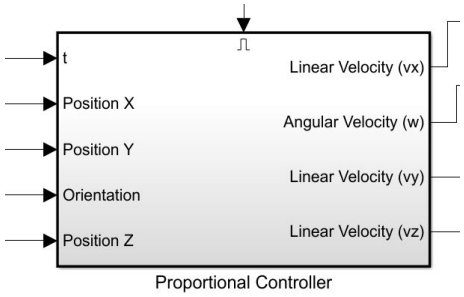

Figura 4-9: Bloque para el controlador de trayectoria.

El bloque de la figura 4-9 contiene un subsistema donde como primera parte las posiciones x,y,z entran de forma matricial y para la orientación entran a 2 bloques uno con la función seno $(\sin)$ y otro con una función coseno $(\cos)$, todas estas entradas llegan a un bloque como se muestra en la figura 4-10 que es definida por el usuario y es donde se programa el controlador y las trayectorias deseadas. Como se mencionó anteriormente las salidas de este bloque son las mismas del bloque de la figura 4-9.

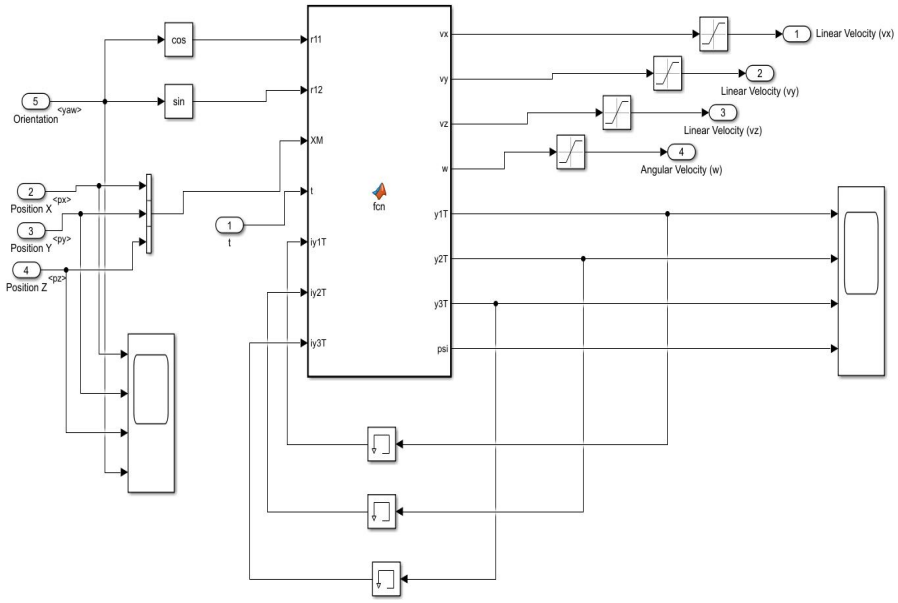

Figura 4-10: Bloques para el controlador Proporcional P.

El paso siguiente es mandar mandar las velocidades u_d,v_d,w_d y r_d al nodo de ROS. Para esto se tiene el bloque de la figura 4-11, lo que realiza este bloque es publicar dichas velocidades a ROS. Este bloque de igual manera que el bloque del control de la figura 4-9, contiene un subsistema con otros dos bloques, un publicador y otro con un mensaje en blanco. Este susbsistema muestra en la figura 4-12.

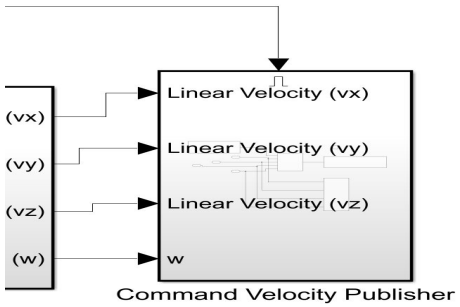

Figura 4-11: Bloque para el publicador de mensajes en ROS.

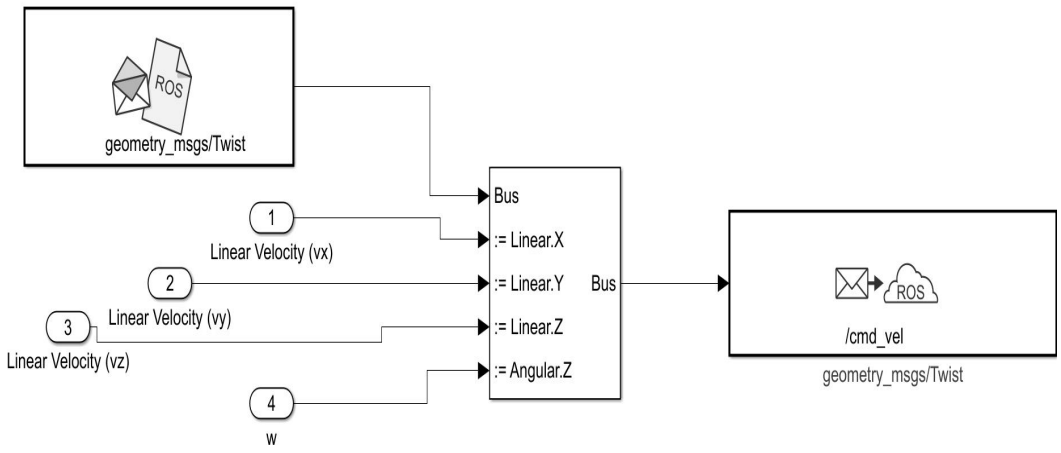

Figura 4-12: Bloque para el publicador de mensajes en ROS.

Donde se puede observar que se tienen las velocidades provenientes del bloque del controlador, así como también dos bloques de ROS. El bloque que se encuentra en el lado derecho llamado /cmd vel es el encargado de publicar las velocidades u_d,v_d,w_d y r_d en el nodo de ROS y de esa manera ROS las mandara al simulador Gazebo. El bloque Blank Message que se encuentra del lado izquierdo de la imagen figura 4-12 llamado geometry msgs/Twist, crea un bus no virtual Simulink correspondiente al tipo de mensaje ROS seleccionado; en este caso son las velocidades. El bloque crea buses de mensajes ROS que funcionan con bloques de servicio de publicación, suscripción o llamada . En cada muestra, el bloque emite una señal en blanco,para el tipo de mensaje designado.

4.3. Resultados de la simulación

Una vez programado el control 3.10, la trayectoria deseada que debe seguir el cuatrirotor es la de un circulo, dado que el control requiere de la posición y la velocidad de dicha trayectoria, se define la posición x_d, y_d, de la siguiente manera.

$$x_d = a\cos(\gamma t), y_d = a\sin(\gamma t) \tag{4.1}$$

y la derivada de la trayectoria de 4.1 es

$$\dot{x} = a\gamma \sin(\gamma t), \dot{y} = -a\gamma \cos(\gamma t) \tag{4.2}$$

Donde a es el radio del circulo y γ es la frecuencia.
A continuación se presenta la tabla 4.1 con los valores ocupados para la simulación.

Parámetros	valor
a	2
γ	0.01
K	diag$\{5, 2\}$
k_z	6
k_ψ	6

Tabla 4.1: Tabla de valores a, γ, $K \in \mathbb{R}^{2\times 2}$, k_z y k_ψ.

La simulación comienza haciendo despegar al drone desde la ventana de comandos de ubuntu, inmediatamente despues de que comienza la simulación en matlab se crea el nodo con ROS enviando de esta manera las velocidades hacia el dron en el simulador Gazebo. El tiempo de simulación fue de 630 segundos, ya que este fué el tiempo que le tomó al cuatrirotor completar su trayectoria.

En la figura 4-13 se muestra el recorrido circular realizada por el cuatrirotor donde también se puede observar que el radio coincide con el paramétro propuesto, que es de dos metros.

En la figura 4-14, se puede observar de igual manera las posiciones del cuatrirotor en las coordenadas x y y donde se observa que de igual manere corresponden al radio de la trayectoria circular, en las figura 4-15 se presenta el movimiento del cuatrirotor en la coordenada z, que como se puede observar se mantiene entre valores de 0.94 y 1.04 y en la figura 4-16 se muestra el movimiento del ángulo de guiñada. Tanto el movimento en z como en ψ, se mantiene cerca de valores constante ya que en este trabajo, solo se controla la parte translacional del cuatrirotor.
Las graficas de los errores de las trayectorias se muestran en las figuras 4-17-4-20. Se tienen cuatro errores que corresponden a las trayectorias de $\tilde{x}$, $\tilde{y}$, $\tilde{z}$ y $\tilde{psi}$, donde se puede observar que dichos errores convergen a un valor muy cercano a cero y se

mantienen en este mismo valor durante la trayectoria realizada por el cuatrirotor. Como observación se recuerda que las medidas de los valores de los errores son en unidades de metro.

Finalmente en la figuras 4-21, 4-22, 4-23 y 4-24 se tienen las graficas velocidades $\tilde{x}$, $\tilde{y}$, $\tilde{z}$ y $\tilde{psi}$, respectivamente; Como se observa, estas velocidades se mantienen constantes.

Un punto importante a mencionar, es que en las gráficas mostradas contienen cierto -uido"que indica un comportamiento no deseado del drone durante la simulación, como se puede ver en la figura 4-24 que es la velocidad del angulo r_d, se observan algunos picos que indican que el dron quiere girar sobre su propio eje. Durante la simulación esto no es muy notorio ya que esta velocidad es cercana a cero es muy posible que esto suceda debido a la transmisión de datos entre MATLAB y ROS, pero aún así, el controlador diseñado cumple con el objetivo de llevar al cuatrirotor por la trayectoria circular deseada.

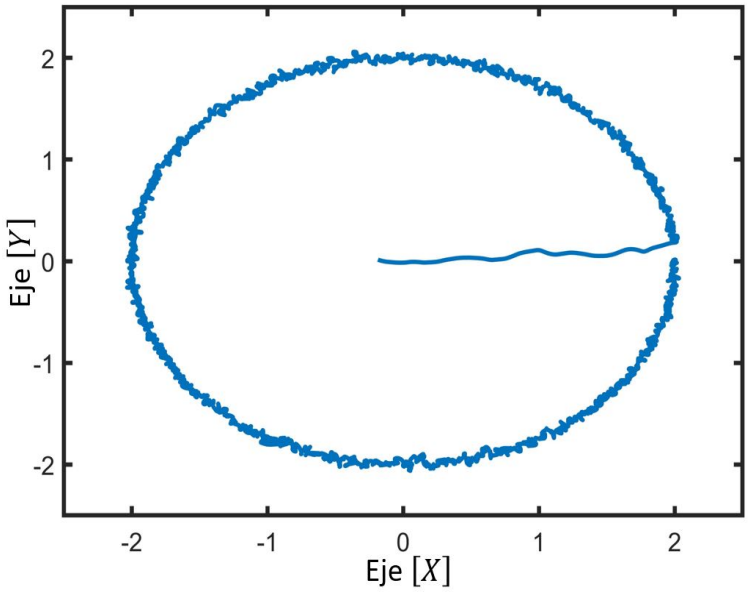

Figura 4-13: Recorrido circular de radio 2 metros del AR Drone en el plano $x - y$.

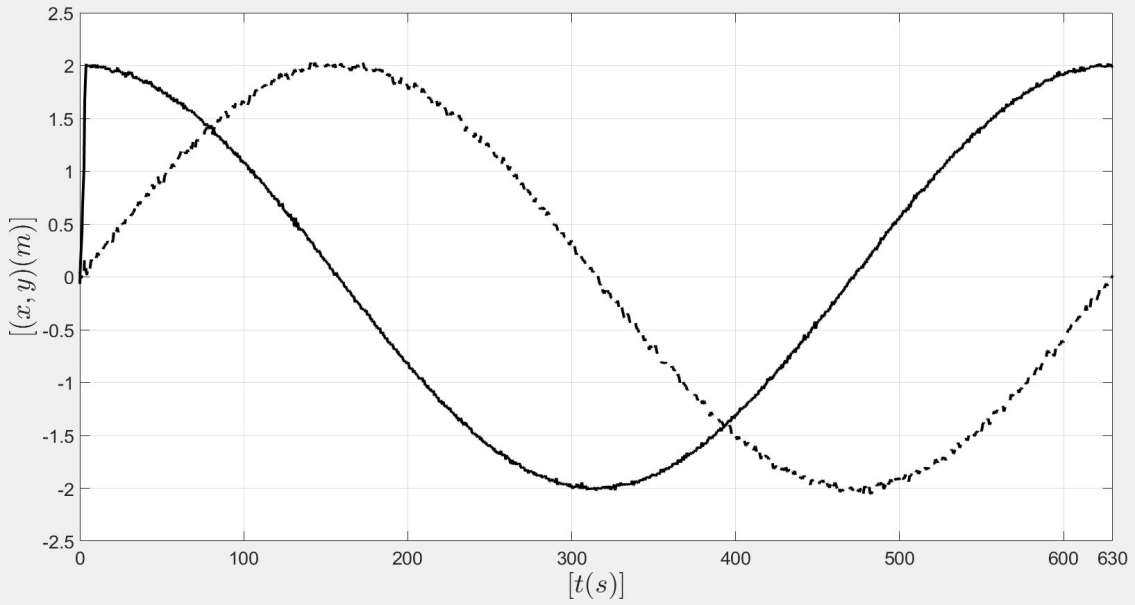

Figura 4-14: Recorrido en el eje x y el eje y del Ar Drone. Donde x es representado por la línea continua y y por la línea punteada

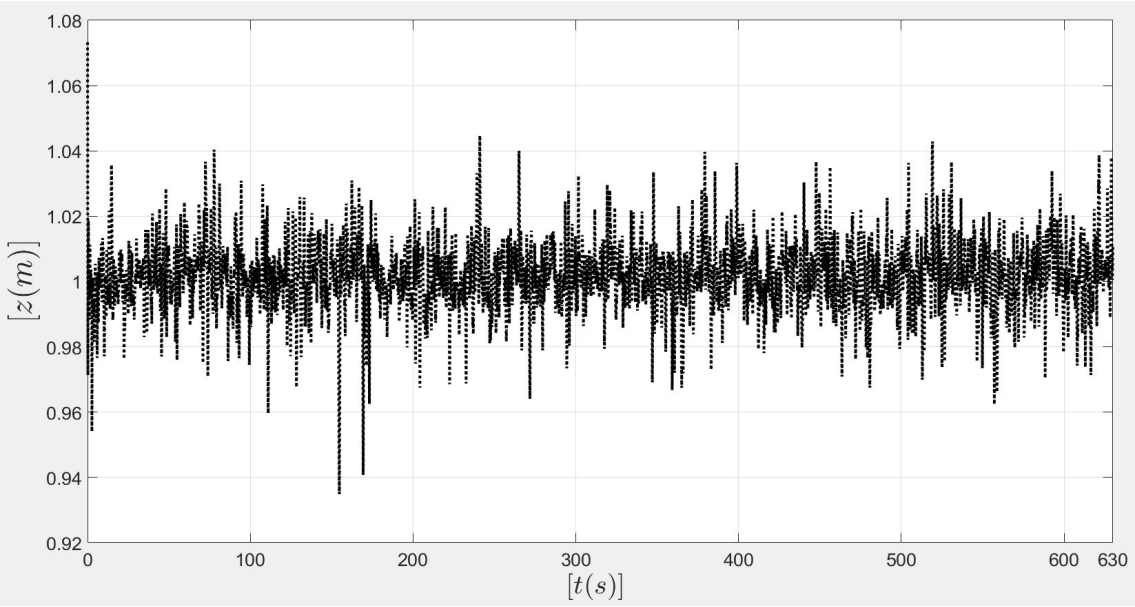

Figura 4-15: Recorrido en el eje z del AR Drone.

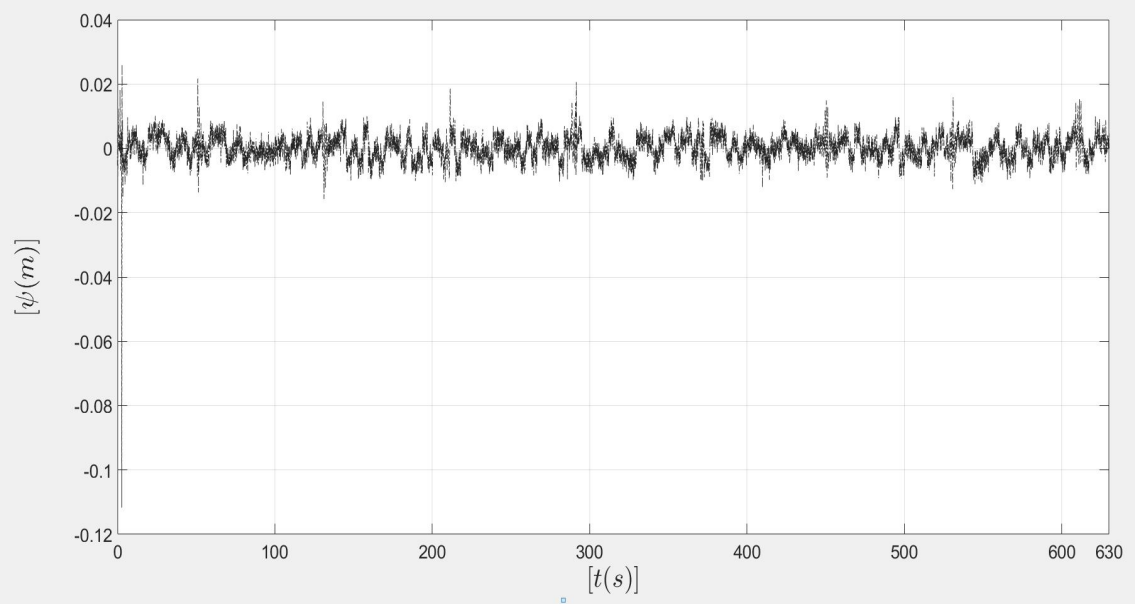

Figura 4-16: Giro del ángulo de guiñada ψ del AR Drone.

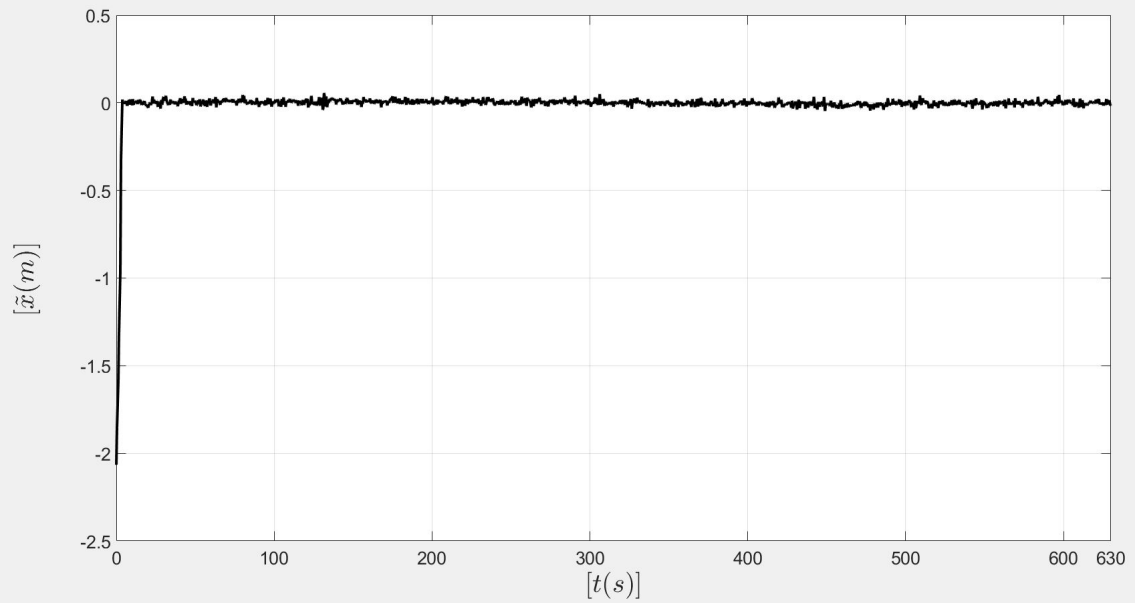

Figura 4-17: Error de trayectoria $\tilde{x}$ del AR Drone.

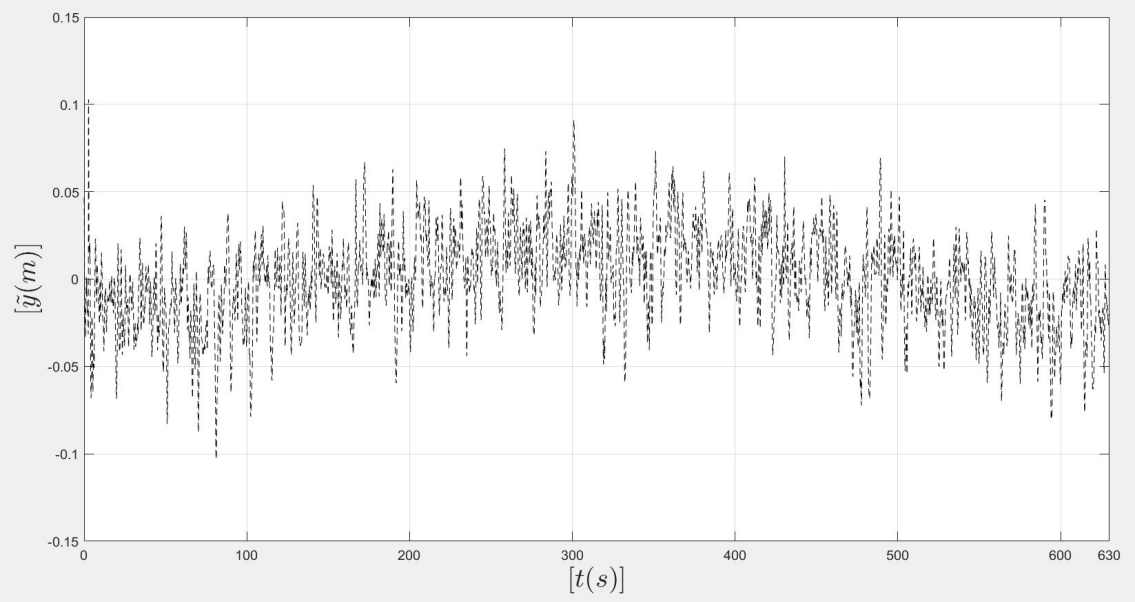

Figura 4-18: Error de trayectoria $\tilde{y}$ del AR Drone.

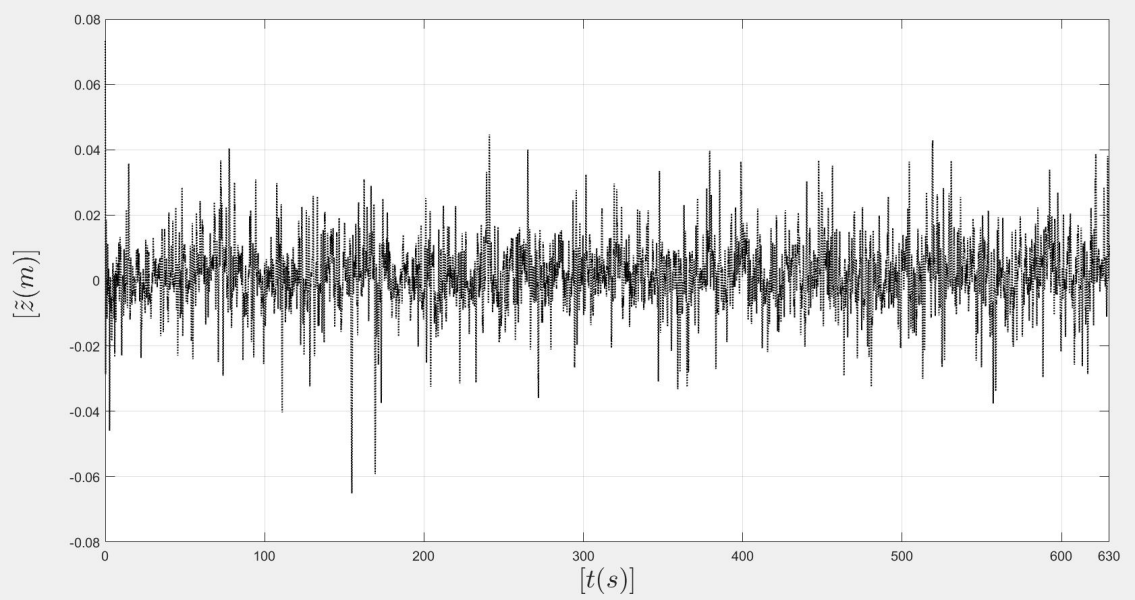

Figura 4-19: Error de trayectoria $\tilde{z}$ del AR Drone.

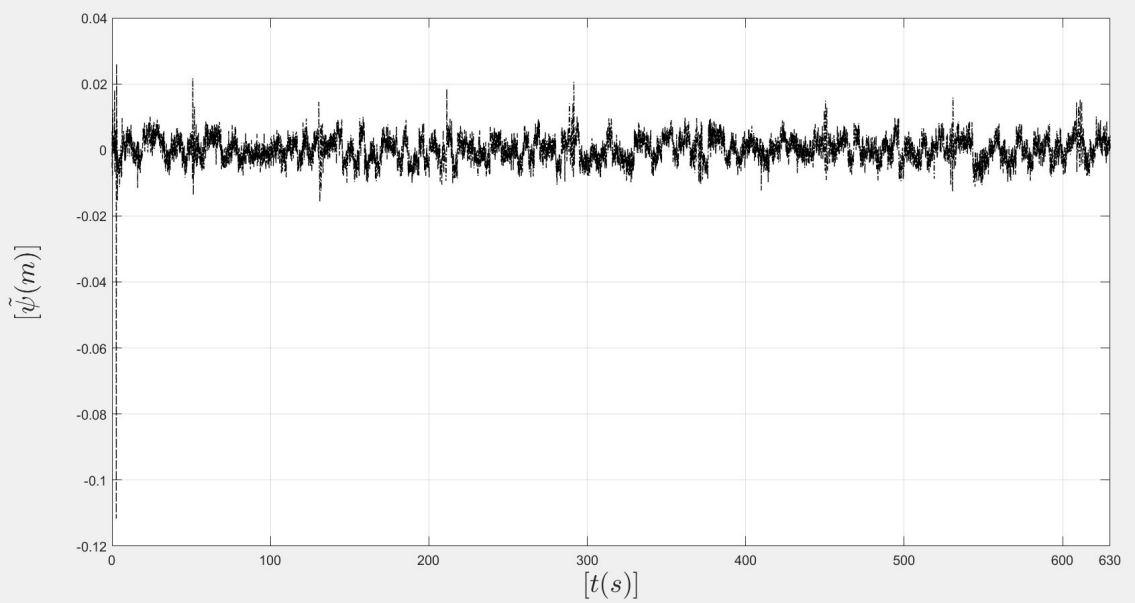

Figura 4-20: Error de trayectoria del ángulo $\tilde{\psi}$ del AR. Drone.

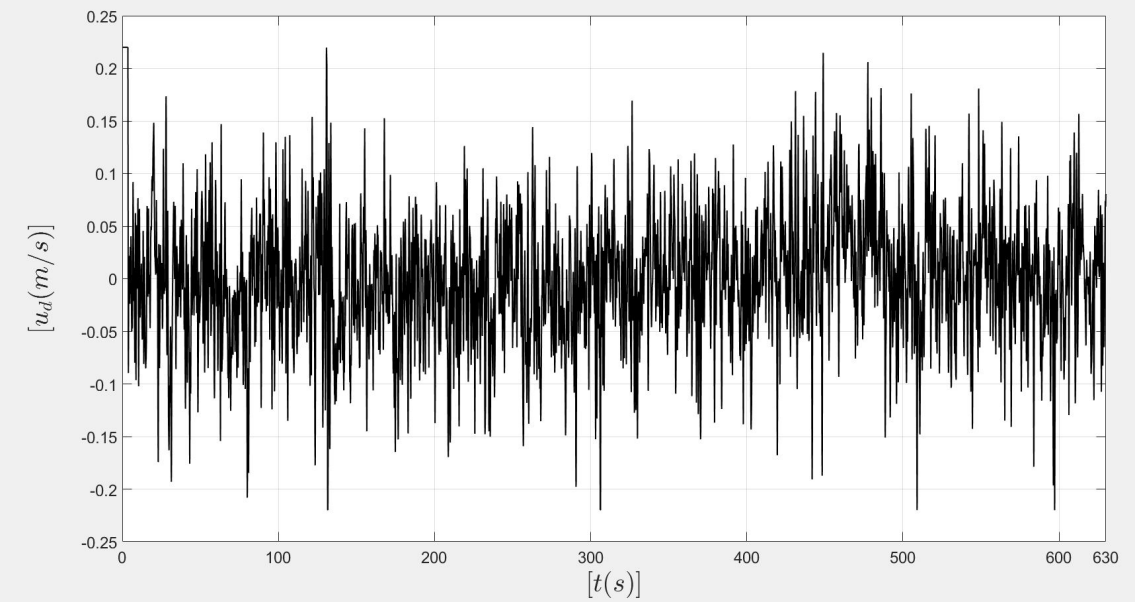

Figura 4-21: Velocidad u_d.

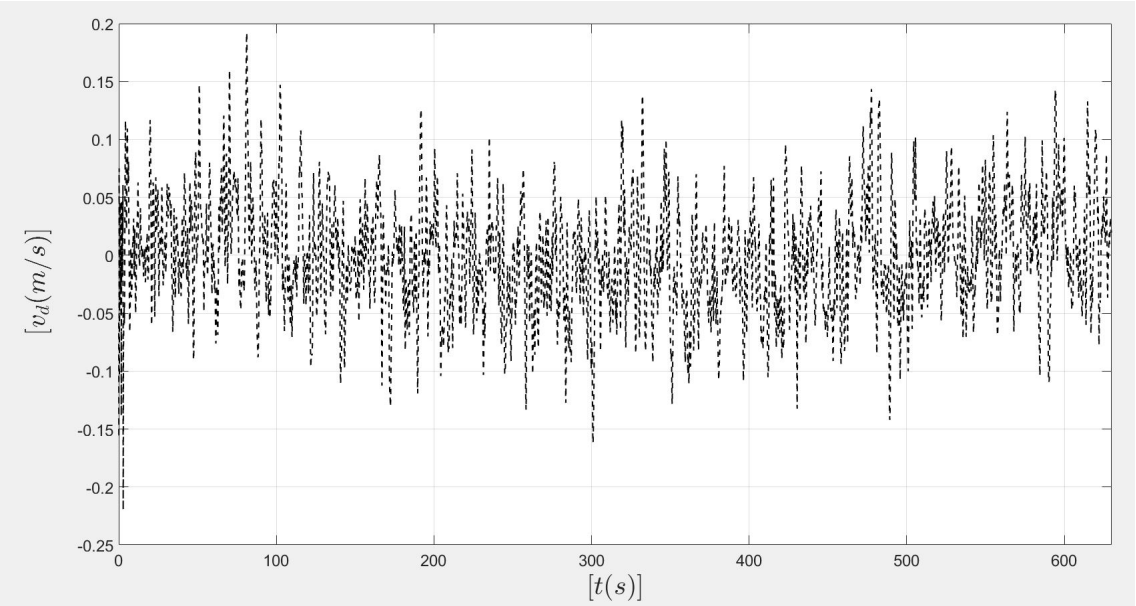

Figura 4-22: Velocidad v_d.

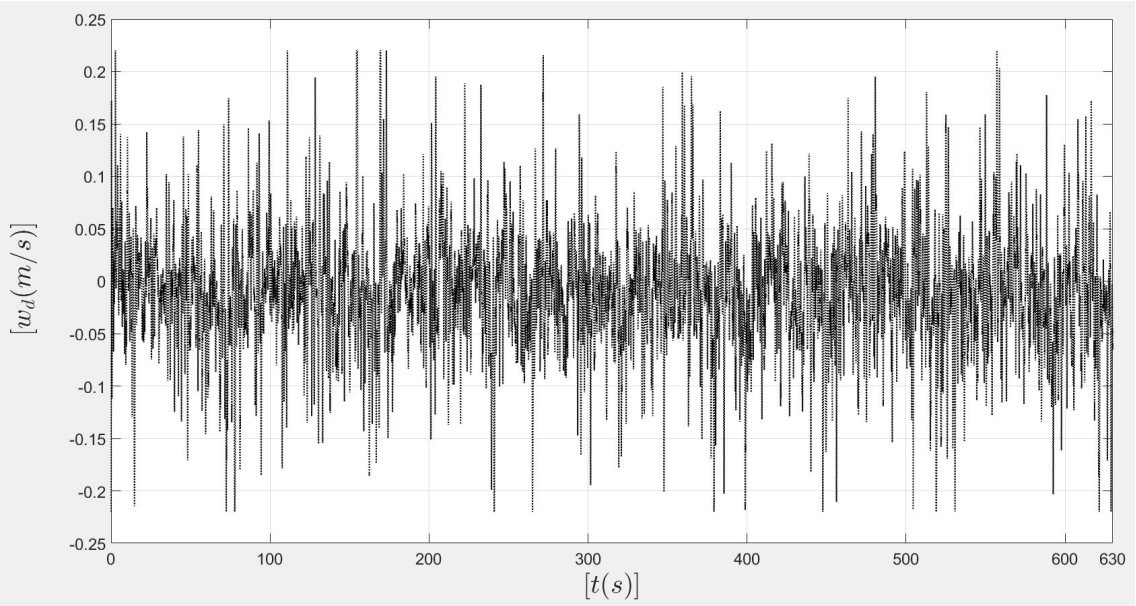

Figura 4-23: Velocidad w_d.

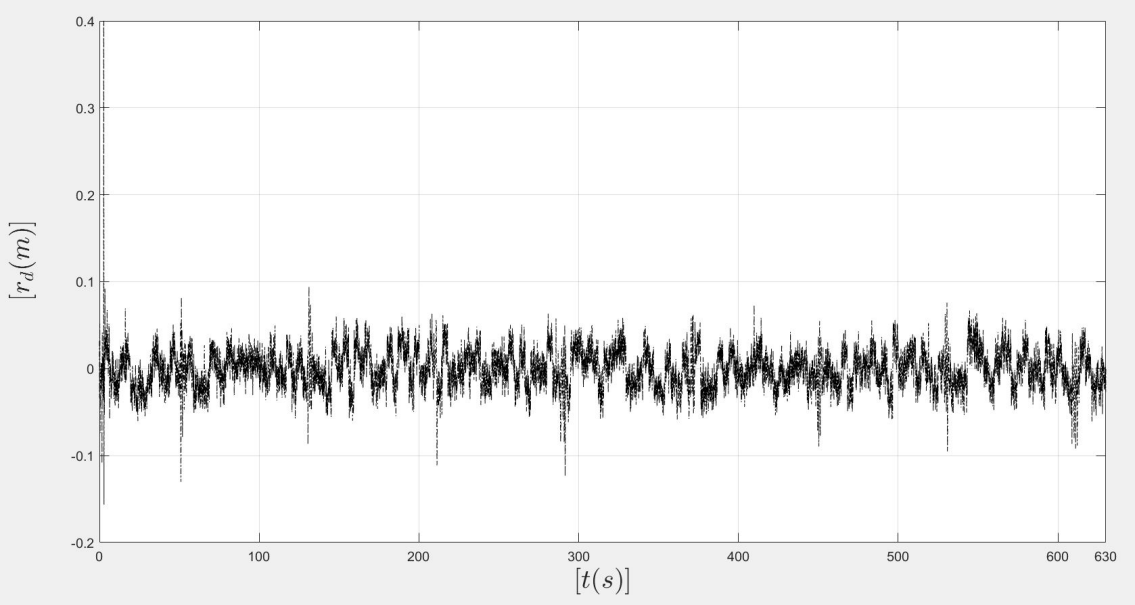

Figura 4-24: Velocidad r_d.

Capítulo 5

Conclusiones y trabajo a futuro

En este capítulo se presentan las concluciones de este trabajo, asi como el trabajo a futuro que se puede realizar con el AR Drone 2.0 utilizando el tópico que permite utilizar su camara.

5.1. Conclusiones

En este trabajo de tesis se presenta el desarrollo de un control para un seguimiento de trayectoria de un cuatrirotor, este controlador se diseñó al considerar como entradas de control las velocidades traslacionales en ejes cuerpo y la velocidad rotacional alrededor del eje vertical para el cuatrirotor AR-Drone 2.0. Aunque existen ligeras desviaciones durante la simulación el cuatrirotor es capaz de seguir la trayectoria circular propuesta; estas desviaciones se pueden deber a ruido existente durante la transmisión de datos entre MATLAB y ROS.

Finalmente, el controlador fue programado en MATLAB/Simulink para correr una simulación cooperativa con el simulador Gazebo a través del software ROS (Robot Operating System), para así observar el desempeño del controlador y la trayectoria propuesta, durante esta simulación se observó como los errores de las trayectorias convergian a cero de manera exponencial en un tiempo finito, logrando completar la trayectoria circular y así el objetivo general y los objetivos particulares establecidos al

principio de este trabajo.

5.2. Trabajo a futuro

Como trabajo a futuro se plantea utlizar la cámara del AR-Drone para visión y procesamiento de imagenes con la ayuda del simulador Gazebo a través del tópico de la camara inferior con la que cuenta el AR-Drone 2.0 y de esta manera proponer un control para la navegación de este con la ayuda de la matriz de homografia. Una vez realizado esto se podran realizar pruebas físicas.

Apéndice

En esta sección se reporta los pasos a seguir para la instalación del sistema operativo Ubuntu versión 16 y los programas de ROS kinetic y Gazebo 7. También se muestran los pasos a seguir para descargar las paqueterias del AR Drone simulator y por último se describen una lista de comandos necesarios para hacer navegar al AR Drone desde el terminal de Ubuntu.

5.3. Instalación del sistema operativo linux (ubuntu 16.04 LTS)

Debido a que el sistema operativo Linux presenta varios problemas cuando se cambia la la versión de los sotwares involucrados, en este proyecto es necesario instalar versiones especificas del sistema operativo Ubuntu, así como también versiones especificas para ROS y el simulador Gazebo, es importante mencionar que tanto ROS como Gazebo se instalan en el sistema operativo de ubuntu ya que como se mencióno anteriormente, Gazebo corre en linux.

Para la instalación del sistema operativo linux (ubuntu 16.04 LTS) se necesitan los siguientes requisitos mínimos, y su instalación es como la de cualquier sistema operativo.

- Intel o AMD 1 GHZ
- Ram 1.5 GB

- Disco duro 7 GB
- Tarjeta grafica VGA
- Tarjeta de red

5.4. Instalación de ROS Kinetic

En este apartado se muestran la lista de comandos necesarios para la instalación de ROS versión Kinetic

numero	Comandos
1	sudo sh-c 'echo deb http://packages.ros.org/ros/ubuntu /etc/apt/sources.list.d/ros-latest.list'
2	sudo apt-key adv –keyserver 'hkp://keyserver.ubuntu.com:80' –recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
3	sudo apt update
4	sudo apt install ros-kinetic-desktop-full
5	sudo apt install python-rosdep python-rosinstall python-rosinstall-generator python-wstool build-essential
6	sudo apt install python-rosdep
7	sudo rosdep init
8	rosdep update
9	echo "source /opt/ros/kinetic/setup.bash»> /.bashrc
10	source /.bashrc

Tabla 5.1: Comandos para instalación de ROS kinetic.

Una vez que se realizan los pasos mostrados en la tabla 5.1, ROS debe estar instalado correctamente, esto se verifica con el comando **rosversion** la cual debe mostar la versión de ROS instalada, como se muestra en la imagen 5-1

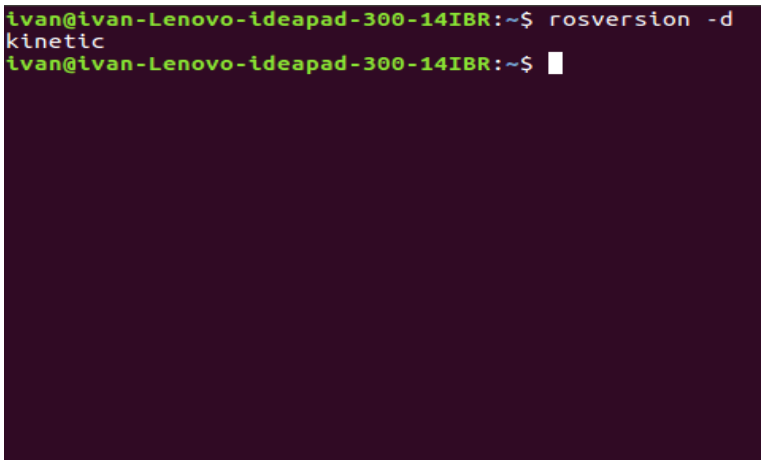

Figura 5-1: Inicialización del nodo ROS con el comando **roscore.**

Los comandos **"source /opt/ros/kinetic/setup.bash"** y **"source /catkin ws/devel/setup.bash"**, siempre deberán ejecutarse cada que se abre el terminal de ubuntu, esto para aplicar la configuración de la ventana de la terminal que se ejecuta en ese momento. Para evitar estos pasos se modifica el archivo **".bashrc"**, agregando al final de ese archivo, las lineas de la tabla 5.2.

número	Comandos
1	Set ROS Kinetic
2	source /opt/ros/kinetic/setup.bash
3	source /catkin ws/devel/setup.bash
4	Set ROS Network
5	export ROS HOSTNAME=xxx.xxx.xxx.xxx
6	export ROS MASTER URI=http://ROS HOSTNAME:11311
7	Set ROS alias command
8	alias cw=’cd /catkin ws’
9	alias cs=’cd /catkin ws/src’
10	alias cm=’cd /catkin ws catkin make’

Tabla 5.2: Comandos necesarios para modificar el archivo .bashc.

Donde el comando **export ROS HOSTNAME = xxx.xxx.xxx.xxx** se usa la dirección IP del equipo, esta se consulta con el comando **ifconfig**.

Para asegurar que el espacio de trabajo esta superpuesto de manera correcta por el scrip de configuración se corre el comando **echo ROS PACKAGE PATH**, el cuál manda el siguiente mensaje en la terminal:
/home/youruser/catkin ws/src:/opt/ros/kinetic/share, como se muestra en la figura 5-2.

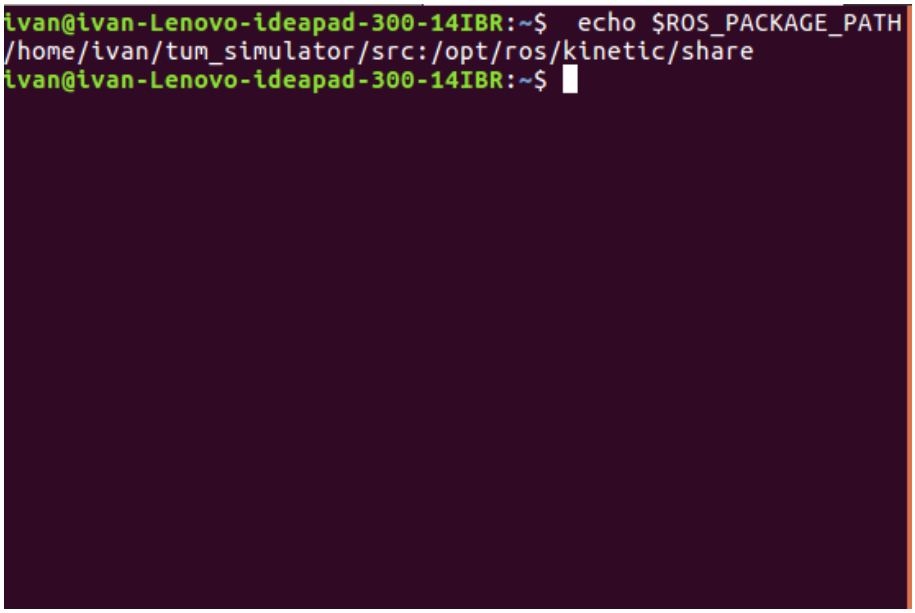

Figura 5-2: Mensaje de comprobación del espacio de trabajo catkin.

5.5. Instalación de Gazebo 7

Para la instalación del simulador Gazebo se utilizan los comandos de la tabla estos de igual manera se introducen en el terminal de ubuntu.

numero	Comandos
1	sudo sh -c ’echo "deb http://packages.osrfoundation.org/gazebo/ubuntu-stable ‘lsb release -cs‘ main"
2	wget https://packages.osrfoundation.org/gazebo.key -O - | sudo apt-key add -
3	sudo apt-get update
4	sudo apt-get install gazebo7

Tabla 5.3: Comandos para la instalación del simulador Gazebo 7.

Para correr el simulador simplemente se introduce el nombre **Gazebo** dentro de la terminal y asi se abre el ambiente.

5.6. Instalación del AR Drone simulator

Los paquetes que se utilizan para el AR Drone autonomy, funcionan con la versión de ROS Kinetic instalado anteriormente.
Como primer paso es necesario crear un espacio de trabajo llamado **catkin ws** , donde se instalan las paqueterías del AR Drone. Dentro del terminal de ubuntu se crean las carpetas **catkin ws** y la carpeta **src** con el comando **mkdir -p /catkin ws/src**. Despúes se entra a la carpeta **catkin ws** con el comando **cd /catkin ws/**. Finalmente dentro de esta carpeta se llama al **catkin make** donde se clonaran las librerias del AR Drone.

Para la instalación del AR Drone autonomy se necesitan las siguientes lineas de comandos:

- **mkdir -p /AR Drone simulator/src:** crea una carpeta con el nombre AR Drone simulator y dentro de esta carpeta, se crea otra llamada src.
- **catkin init workspace:** crea un catkin para el espacio de trabajo del AR Drone.

- **git clone https://github.com/iolyp/AR Drone simulator gazebo7:** clona el repositorio del AR Drone autonomy.
- **catkin make:** clona las librerias que utiliza el AR Drone autonomy

Despues de crear la carpeta entramos en ella con el comando **cd /AR Drone simulator/src**, como siguiente paso se crea un catkin para el espacio de trabajo del AR Drone con el comando catkin init workspace, como paso siguiente clonamos el repositorio del AR Drone autonomy con el comando **git clone https://github.com/iolyp/AR Drone simulator gazebo7**. Finalmente, se realiza nuevamente un catkin make donde se clonaran las librerias para el AR Drone.

Para correr el AR Drone autonomy se utiliza la linea de comando **cd /AR Drone simulator source devel/setup.bash roslaunch cvg sim gazebo AR Drone testworld.launch**. Una vez que se abre el programa de gazebo se puede visualizar el ambiente de Gazebo y el AR Drone como se muestra en la figura 5-3.

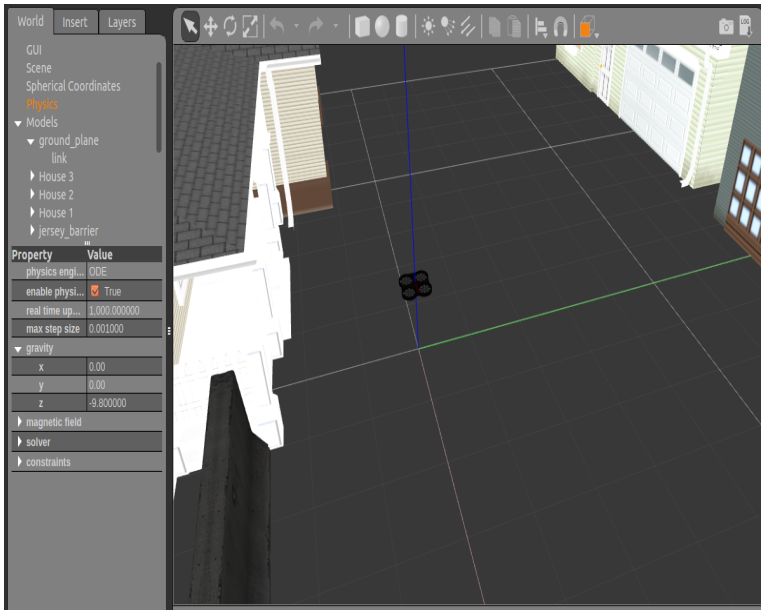

Figura 5-3: Simulador Gazebo con la paquetería del AR Drone.

5.7. Control de comando en terminal

Para despegar y aterrizar al AR Drone autonomy se describen los siguientes comandos mostrados en la tabla 5.4. Estos comandos deben ejecutarse en el terminal de ubuntu.

	Comandos
despegue	rostopic pub -1 /AR Drone/takeoff std msgs/Empty
Aterrizaje	rostopic pub -1 /AR Drone/land std msgs/Empty

Tabla 5.4: Control de comando de terminal.

También se enlista los siguientes comandos para el AR Drone autonomy que sirven para dar un movimiento específico en una direccción, de igual manera estos comandos se ejecuntan desde la terminal.

	Comandos
Volar hacia adelante	rostopic pub -r 10 /cmd vel geometric msgs/Twist' lineal:x: 1,0, y: 0,0, z: 0,0, angular: x: 0,0, y: 0,0, z: 0,0'
Volar hacia atrás	rostopic pub -r 10 /cmd vel geometric msgs/Twist 'lineal: x: -1.0, y: 0.0, z: 0.0, angular: x: 0.0,y: 0.0,z: 0.0'
volar a la izquierda	rostopic pub -r 10 /cmd vel geometric msgs/Twist 'lineal: x: 0.0, y: 1.0, z: 0.0, angular: x: 0.0,y: 0.0,z: 0.0'
volar a la derecha	rostopic pub -r 10 /cmd vel geometric msgs/Twist 'lineal: x: 0.0, y: -1.0, z: 0.0, angular: x: 0.0,y: 0.0,z: 0.0'
volar hacia arriba	rostopic pub -r 10 /cmd vel geometric msgs/Twist 'lineal: x: 0.0, y: 0.0, z: 1.0, angular: x: 0.0,y: 0.0,z: 0.0'
volar hacia abajo	rostopic pub -r 10 /cmd vel geometric msgs/Twist 'lineal: x: 0.0, y: 0.0, z: 0.0, angular: x: 0.0,y: 0.0,z: -1.0'

Tabla 5.5: Control de movimiento del AR Drone.

Bibliografía

[1] Thomas Lagkas Ioannis Moscholios Panagiotis Radoglou-Grammatikis, Panagiotis Sarigiannidis. Applications of unmanned aerial vehicle (UAV) in road safety, traffic and highway infrastructure management: Recent advances and challenges. *Transportation Research Part A: Policy and Practice*, 141:116–129, 2020.

[2] I. M. Kroo, Michael Shantz, Peter Kunz, Gary Fay, Shelley J. Cheng, and Tibor Fábián. The mesicopter: A miniature rotorcraft concept phase ii interim report. 2000.

[3] José Fermi Guerrero Castellanos. Control y navegación de vehículos aéreos no tripulados: un enfoque teórico-experimental. *Komputer Sapiens*, 3, 10. Septiembre-diciembre 2018.

[4] Uav assistance paradigm: State-of-the-art in applications and challenges. *Journal of Network and Computer Applications*, 166:102706, 2020.

[5] Dianxiong Liu, Yuhua Xu, Jinlong Wang, Jin Chen, Kailing Yao, and Qihui Wu. Opportunistic uav utilization in wireless networks: Motivations, applications, and challenges. *IEEE Communications Magazine*, 58:62–68, 05 2020.

[6] Fatma Outay, Hanan Abdullah Mengash, and Muhammad Adnan. A compilation of uav applications for precision agriculture. *Computer Networks*, 172:107148, 2020.

[7] Quan. Q. *Introduction to Multicopter Design and Contro*. Springer Singapore, 2017.

[8] Pierre Eline Frederic D'Haeyer StephanePiskorski, Nicolas Brulez. Ar.drone developer guide. *Parrot*, 2012.

[9] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, 2004.

[10] O. Alvarez fres y S. Marcos Pablos P. Gómez del Torno. *Service Robotics within the Digital Home: Applications and Future Prospects*. Robotic Development de Service Robotic within the Digital Home. Springer New York, 2011.

[11] Lentin Joseph. *Robot Operating System (ROS) for Absolute Beginners.* Springer 2018., 2018.

[12] Rodríguez-Cortés Hugo Corona-Sánchez, José J. Trajectory tracking control for a rotary wing vehicle powered by four rotors. *Journal of Intelligent Robotic Systems*, 70:107148, 2013.

[13] M. Fecko. *Differential Geometry and Lie Groups for Physicists*. (Cambridge University Press. 2006.

[14] F. L. Lewis B. L. Stevens and E. N. Johnson. *Aircraft Control and Simulation: Dynamics, Controls Design, and Autonomous Systems*. (John Wiley Sons, 2015. 2015.

[15] C.-A. Arteaga-Escamilla, R. Castro-Linares, and J. Álvarez-Gallegos. Trajectory tracking of multiple quadrotors while maintaining time-varying spatial formations via synchronization. In Eusebio E. Hernandez, Sajjad Keshtkar, and S. Ivvan Valdez, editors, *Industrial and Robotic Systems*, pages 97–108, Cham, 2020. Springer International Publishing.

[16] Parrot AR-Drone 2.0. `http://ardrone2.parrot.com/`, 2022.

[17] M.A. Serrano E. Aranda. Modelado y control de un prototipo comercial de aeronave tipo quadrirotor. *Congreso Nacional de Control Automático 2014*, 2014.

[18] Documentación y API's oficiales de ROS. `http://wiki.ros.org/es`, 2022.

[19] Cristian Camilo Cuevas Castañeda. Ros-gazebo. una valiosa herramienta de vanguardia para el desarrollo de la robótica. 2016.

[20] M. Eaton. *Evolutionary Humanoid Robotics*. Springer, 2015.

[21] Documentación oficial de Gazebo; Docs: Get Started (gazebosim.org). `https://gazebosim.org/docs`, 2022.

[22] M. Monajjemi. Ardrone autonomy: A ROS Driver for ARDrone 2.0. `https://github.com/AutonomyLab/ardrone autonomy`, 2019.

[23] En linea. `http://wiki.ros.org/tum simulator`, 2019.

[24] Documentación oficial de Matlab (ROS). `https://la.mathworks.com/help/ros`.

[25] Documentación oficial de Matlab (Robot System Toolbox). `https://la.mathworks.com/help/robotics/getting-started-with-robotics-system-toolbox.html`.

[26] I. Cruz-Vega J. Martinez-Carranza A. López-Luna, H. Rodríguez-Cortés. , an immersion and invariance controller for aerial manipulation. *Unmanned Systems*, 10, 2022.

[27] H. K. Khalil. *Nonlinear Systems*. Pearson Education. Prentice Hall, 2002.

[28] Wei Ren and Randy W Beard. Trajectory tracking for unmanned air vehicles with velocity and heading rate constraints. *IEEE Transactions on Control Systems Technology*, 12(5):706–716, 2004.

[29] Mehdi Golestani and Iman Mohammadzaman. Pid guidance law design using short time stability approach. *Aerospace Science and Technology*, 43:71–76, 2015.

[30] UAV Systems International. `https://uavsystemsinternational.com/blogs/drone-guides/what-is-a-fixed-wing-drone-advantages-and-uses-of-fixed-wing-drones`, 2018.