INSTITUTO TECONOLOGICO DE TUXTLA GUTIERREZ

LABORATORIO DE INGENIERÍA ELECTRÓNICA Y MECATRÓNICA DEL ITTG,

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

REPORTE DE RESIDENCIA PROFESIONAL

PROYECTO:

DISEÑO Y CONTROL DE UN CUADROTOR UTILIZANDO SOFTWARE Y HARDWARE LIBRE.
ETAPA SOFTWARE

PRESENTA:

LÓPEZ AGUILAR RODOLFO

ASESOR INTERNO

Dr. FRANCISCO RONAY LÓPEZ ESTRADA

TUXTLA GUTIÉRREZ, CHIAPAS; DICIEMBRE DEL 2015

Diseño y control de un cuadrotor utilizando software y hardware libre. Etapa Software.

RODOLFO LÓPEZ AGUILAR

18 de diciembre de 2015

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

En el siguiente proyecto se desarrolla el vuelo autónomo de un vehículo aéreo no tripulado (VANT), con un cerebro basado en Linux, y corriendo un sistema operativo especial para robots, el Robot Operating System (ROS), donde con ayuda de sus paqueterías y características propias proporciona grandes ventajas al utilizar este software.

Los VANT durante los últimos años han recibido considerable atención por parte de investigadores, profesores y estudiantes como un gran avance tecnológico, los últimos avances en la tecnología han impulsado el desarrollo de diferentes estrategias de control a si como la operación de este tipo de vehículos, una parte importante es el modelo dinámico de un VANT que es el punto de partida para todos los estudios basados en la navegación aérea, también haciendo uso de muchos métodos de control en el cual son implementados en los VANT para un control autónomo, algunos de estos métodos es el controlador PID, control no lineal y controladores LQR entre otros, métodos que requieren información precisa para su aplicación, a si como igual se necesita información de las mediciones de posición y altitud que se pueden obtener haciendo uso de diferentes aparatos de medición como un giroscopio, un acelerómetro, magnetómetro y otros aparatos, como el GPS, sensores ultrasónicos y sensores de presión barométrica.

El propósito de este proyecto es presentar el control autónomo de trayectorias y proporcionar conceptos importantes sobre la navegación autónoma y el control de vehículos no tripulados, para formar una base de investigación innovadora dentro del Instituto Tecnológico de Tuxtla Gutiérrez. Esté se persigue con dos objetivos principales. El primer objetivo es estudiar el modelo matemático de la dinámica de los VANT, el segundo objetivo es el desarrollo de métodos adecuados para la estabilización y control de trayectoria.

En el proyecto se abordará el modelado básico para la navegación autónoma, se hace uso de los ángulos de navegación son un tipo de ángulos de Euler usados para describir la orientación de un objeto en tres dimensiones, si se tiene un sistema de coordenadas móvil respecto de uno fijo, en tres dimensiones, y se desea dar la posición del sistema móvil en un momento dado, hay varias posibilidades de hacerlo, una de ellas son los ángulos de navegación, llamados en matemáticas ángulos de Tait-Bryan, son tres coordenadas angulares que definen un triedro rotado desde otro que se considera el sistema de referencia se definen matemáticamente de forma similar a los ángulos de Euler, pero en vez de usar como línea de nodos el corte entre dos planos homólogos (por ejemplo el xy es el homólogo del xy), se utilizan dos planos no homólogos (por ejemplo xy e yz).

Para el control y manipulación de nuestro VANT, así como su control en un entorno tridimensional,

como se menciono anteriormente se hará uso de ROS ya que este sistema operativo de código abierto mantenido por la Open Source Robotics Foundation (OSRF), proporciona los servicios que caben esperar de un sistema operativo incluyendo las abstracciones de hardware, control de dispositivos a bajo nivel,implementación de herramientas y librerías para obtener, compilar, escribir y ejecutar código a través de múltiples ordenadores, ROS se compone de un numero de nodos independientes, cada uno se comunica con el resto de nodos utilizando el modelo publicador/suscriptor.

ROS es definitivo para el desarrollo de aplicaciones de robots proporciona una arquitectura distribuida y contiene algoritmos implementarios del estado del arte, mantenidos por expertos en el campo todo con una licencia permisiva que en unos pocos años cambiara el panorama de la robótica y es ampliamente adoptado por la investigación, centros educativos y la industria.

Otras de la ventajas que el uso de ROS supone para este proyecto es que contiene paquetes específicos para el control y manipulación de vehículos aéreos no tripulados, lo cual facilitará el trabajo en este sentido. ROS esta liberada bajo los términos de licencia BSD (Berkeley Software Distribution) y un software open source.

ROS promueve la reutilización de código así los desarrolladores y científicos lograran autonomía sobre los VANT y no tienen que reinventar la rueda todo el tiempo, con ROS podemos hacer esto y mucho mas, se puede tomar el código de los repositorios, mejorarlo y compartirlo de nuevo, es de esta manera que se realiza dicho proyecto y poder generar interés dentro del campo de la investigación.

Con ayuda ayuda de los protocolos de comunicación remota como lo es SSH nosotros podemos accesar de forma remota al sistema operativo del VANT y con eso podemos saber posición en el espacio, trayectoria, orientación, etc. desde una computadora que funciona como base de control a través de una conexión WIFI.

1.1. Objetivo general

Diseñar y controlar un vehículo aéreo no tripulado tipo cuadrotor utilizando software libre enlazado a una PC mediante el sistema operativo ROS (Robotic Operating System) con el protocolo de comunicación WIFI y SSH para accesar a maquinas remotas.

1.2. Justificación

Se aprovechara una importante característica de nuestro drone que es la forma de comunicación, la cual hace posible una integración con otros sistemas operativos en particular una computadora en la cual sera nuestro centro de mando y a la vez recibir los datos de navegación en tiempo real a través de la misma, todo esto se logrará por medio de una conexión WIFI y con el protocolo SSH. La idea principal es la de dar compatibilidad a través de otro sistema operativo. En este caso, elegimos Linux en concreto Ubuntu, ya que nos permite la compatibilidad mediante el uso de un software específico para sistemas robóticos, en concreto es el meta-sistema operativo ROS (Robot Operating System), formado por un conjunto de programas, herramientas y librerías de programación, todos bajo la licencia de software libre y gratuito todo esto nos llevara a obtener nuestro control adecuado para nuestro VANT.

1.3. Hardware: Erle-copter

Teniendo en cuenta que nuestro vehículo está conformado por diferentes sensores, microprocesadores y sistemas de propulsión que son más pequeños, ligeros y mas capaces que nunca, llevando a niveles de resistencia, eficiencia y autonomía que sobrepasan las capacidades humanas el cual son de mucha importancia para la navegación y estabilización de los VANT, cabe decir que es un hardware muy completo que genera gran cantidad de aplicaciones ante la sociedad y una de las características más importantes es la parte de como recibir los datos de navegación por medio de WIFI a través del protocolo de comunicación de maquinas remotas SSH, la cual nos permite la comunicación entre el vehículo aéreo y una computadora que es el punto de control para nuestro vehículo. Además otras de las ventajas, respecto a otros modelos de quadrotores, es que es de software libre por lo que podemos entrar a todos sus parámetros, leer todos sus sensores y sobre ello basarnos para crear un nuevo programa o bien modificar el que ya trae, para ejecutar otras funciones, como bien puede ser graficar su trayectoria a través de a lectura del sensor giroscópico y acelerómetro a través de cálculos matemáticos, por lo que lo hace muy factible para apoyar al estado del arte existente y a la investigación [8]. figura 1.1.

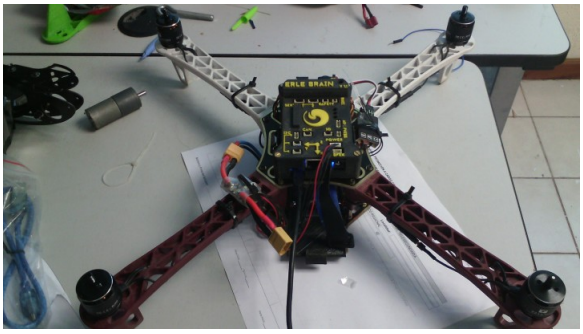

Figura 1.1: Erle-Copter

Capítulo 2

Estado del arte

2.1. Vehículo aéreo no tripulado, UAV.

Los drones o UAV tienen un gran potencial en áreas muy diversas, ya que puede desplazarse rápidamente sobre un terreno irregular o accidentado y superar cualquier tipo de obstáculo ofreciendo imágenes a vista de pájaro y otro tipo de información recogida por diferentes sensores[3].

Un sistema con múltiples robots UAV es más robusto aún, debido a la redundancia que esto ofrece, permite la cooperación en paralelo entre los drones, ayudándose unos a otros para, por ejemplo, cubrir grandes áreas en exteriores o crear redes de sensores móviles. Estos enjambres de vehículos aéreos no tripulados pueden desplegarse para realizar tareas de búsqueda ante cualquier tipo de desastre natural, como terremotos o ataques terroristas, ayudando a localizar a personas que puedan necesitar ayuda.

Las siglas UAV corresponde en ingles a Unmanned Aerial Vehicle, es decir, vehículo aéreo no tripulado. El origen del desarrollo de estos vehículos pertenece a fines militares, ya que dan la posibilidad de realizar operaciones de alto riesgo, o incluso la de sobrevolar una zona en conflicto para la vigilancia o recogida de información.

Vista la configuración aerodinámica, de apariencia novedosa, que se lleva trabajando en estos particulares sistemas desde hace años, sin embargo, en estos últimos años es cuando su pendiente desarrollo ha sido más exponencial, llegando incluso algunos de ellos a utilizarse de forma recreativa. Existen una gran variedad de UAVs, desde vehículos de grandes dimensiones para altos vuelos y/o grandes distancias de vuelos, hasta pequeñas dimensiones llegando incluso a pocos centímetros.

La investigación a aumentado considerablemente en los últimos años esa es la razón por la cual existen algunos modelos matemáticos referenciales de este cuadrotor. El desarrollo de un UAV se basa en el tipo de acción que va a llevar a cabo, y la distancia que tendrá que recorrer. A demás de dimensiones específicas, necesita de una tecnología de transmisión y vuelo necesaria para realizar la tarea. En este punto es necesario aclarar que vehículo no tripulado, UAV, no tiene un significado autónomo, sino que estará comunicado desde un operador de tierra, sean pilotos, controladores o cualquier otro tipo de operario relacionado con la monitorización de la aeronave.

Una vez superado el reto de la creación de vehículos no tripulados, se investigó en otro nivel los llamados UAS, Unmanned Aircraft System . Un UAS se trata de la evolución directa de un UAV. Los UAVs pueden estar controlados remotamente desde una estación de tierra por un operador, en cambio,

los UAS son autónomos y seguirán una trayectoria ya predefinida, o un vuelo con los recursos de los propios sensores. Existen dos estaciones que pueden manejar información del UAV, la estación de tierra y la estación a bordo del UAV. Dependiendo de cuan autónomo sea el UAV, la estación de tierra realizará más o menos funciones de forma habitual.

2.1.1. Aviones UAV con motor a reacción

En uso militar predominan los UAV de tipo avión, con motores de propulsión a reacción o de pistón. Dentro de los UAV, existe otra denominación para un uso en combate, que es UCAV, Unmmaned Combat Air Vehicle, denominados vehículos aéreos no tripulados de combate.

Un ejemplo de UCAV propulsado con reactores, es el Barracuda proyectado conjuntamente entre España y Alemania, a través de EADS. Figura 2.1

Figura 2.1: Barracuda

Con el mismo fin de desarrollo se encuentra el X-45, creado por la empresa Americana Boeing fue parte del proyecto J-UCAS de DARPA. Figura 2.2

Figura 2.2: X-45

2.1.2. Vehículos UAV, con motor de pistón

UAV con propulsión de tipo pistón se encuentra Predator, sirve principalmente en misiones de reconocimiento, pero además, tiene capacidad ofensiva con la posibilidad de incorporarle dos misiles. En la siguiente imagen además del propio avión podemos ver parte de la cabina de control de tierra, compuesta en su totalidad por una plantilla de 55 personas. figura. 2.3

Figura 2.3: Predator

2.1.3. Vehículos no tripulados de peso reducido

El más novedoso es el Honeywell RQ-16A T-Hawk, desarrollado por Honeywell, es un pequeño UAV con un motor de gasolina propulsado por un único ventilador colocado en su centro. Su fin es la vigilancia y reconocimiento, este modelo no dispone de armas de defensa u ofensiva, pero tiene un largo alcance en proporción a su peso de solo 8.3 Kg. Su reducido peso y tamaño hace que entre dentro de la denominación MAV, del inglés, Micro Aerial Vehicle.

Su creación surgió gracias a la agencia DARPA, responsable de la investigación en el ámbito militar, y actualmente se sigue utilizando en Afganistán por el ejército de los EEUU. Al ser un vehículo no tripulado, no supone riesgo en labores de reconocimiento. Figura. 2.4

Figura 2.4: T-Hawk

En el contexto del desarrollo de vehículos no tripulados, en su mayoría para un fin militar, tanto de grandes dimensiones, o de pequeñas dimensiones como el RQ- 16 T-Hawk, aparecen numerosos modelos para un uso recreativo. Se distribuyen multitud de modelos que se pueden calificar, al igual que los anteriores, en su propulsión o configuración aerodinámica.

Pero el más innovador, fue el Ardrone 1.0 de Parrot, empresa francesa que lo desarrolló y distribuye, con una configuración de cuadricóptero con motores eléctricos.

Se diferencia de otros modelos ya que posee un microprocesador competente para el procesamiento de las señales recibidas a través de una serie de sensores. Además incluye dos cámaras que le permiten captar lo que ocurre a su alrededor. Incorpora un cambió en la estación de control, ya que necesita de un Smartphone para el control del mismo.

La llegada de un nuevo modelo, el Ardrone 2.0, con diferentes cámaras y sensores, ha sido necesario un nuevo modelo de sistema de control para este novedoso modelo. Figura. 2.5

Figura 2.5: AR.Drone 2.0

La versión 2.0 es más potente en todas sus características importantes para su navegación y estabilización, pero que aun mantiene la esencia de su predecesor con la conexión Wi-Fi, la distribución de sus dos cámaras y las pequeñas dimensiones.

El reto de este proyecto es hacer el control y estabilización en diferentes trayectorias a través de modelos matemáticos ya establecidos a nivel de investigación y principalmente haciendo uso de un software en código abierto, en este caso ROS el cual nos aporta diferentes paquetes de navegación y control, y nos permitirá hacer la navegación y el control de nuestro vehículo aéreo no tripulado.Erle-copter

Figura 2.6: Erle-copter

2.2. Normativa Mexicana CO-AV-23/10

Con base en la circular emitida por la Dirección General de Aeronáutica Civil No. CO-AV-23/10 del estado mexicano, que se puede ver en [?], se ha definido esta normativa de carácter nacional para impulsar un sector tecnológicamente puntero y emergente. El espacio aéreo está regulado y a veces prohibido para el vuelo Drone en determinados lugares y/o momentos lo que se debe entender, es que no se puede permitir ningún incidente entre paracaídas, ala delta, globo aerostático, planeador, helicóptero, avión de pasajeros o Drone ya que las consecuencias serían trágicas para todos los vehículos y personas involucradas, el respeto de esta normativa significa preservar la libertad que tenemos en México para volar Drones.

Reciben este nombre los vehículos aéreos no tripulados o aeronaves controladas de forma remota, conocidas según sus diversos acrónimos en inglés: UAV, Unmanned Aerial Vehicle y UAS, Unmanned Aerial System. La Organización de Aviación Civil Internacional (OACI/ICAO) a través de la Circular

328 AN/190 del 2011, emitió un pronunciamiento en el cual estableció que RPAS (Remotely Piloted Aircraft Systems) sería la sigla internacional para identificar estas aeronaves. En español son conocidos por el acrónimo VANT (Vehículo Aéreo No Tripulado) o simplemente DRONE, palabra que puede conconsiderarse una adaptación válida al español del sustantivo inglés drone (literalmente abejorro, zángano o zumbador por el ruido que emiten en su operación).

2.2.1. Clasificación

A-AD-S1/S2 Drone recreativo

B-AD-S3/S4 Drone civil equipado con cámara profesional, FPV y/o GPS

Ambos, aparatos de despegue vertical, ala rotativa y multirotor que no sean helicópteros: pueden ser tricópteros (3 rotores), cuadricópteros (4 rotores), hexacópteros (6 rotores) y octocópteros (8 rotores) controlables en sus tres ejes. Esta clasificación incluye, de manera general, los elementos individuales del sistema en su conjunto tales como la estación de control en tierra y cualquier otro elemento necesario para facilitar su vuelo, como es el enlace de comunicación (GPS, FPV -FIRST PERSON VIEW-) y el o los dispositivos necesarios para su recuperación. Hay los que se manejan manualmente desde una ubicación remota así como los que vuelan de forma autónoma sobre la base de planes de vuelo preprogramados usando sistemas más complejos de automatización dinámica (despegue, vuelo y aterrizaje automático).

Drone recreativo (Clasificación A-AD-S1/S2)

Vehículo aéreo no tripulado utilizado solo para fines recreativos, con o sin cámara, (Genérica, GoPro, Gear Pro, etc.) que pese menos de 20 kilogramos, exento de bitácora de vuelo, documentos de aeronavegabilidad y autorizaciones de vuelo por los fines que persigue.

Reglas

Este equipo despega, vuela y aterriza solo al alcance de la vista del piloto, por debajo de los 122 metros, en zonas pobladas (siempre que esto no suponga un riesgo claro de daño a otros). El vehículo aéreo deberá portar una etiqueta/placa de 10x5 cm que muestre nombre, dirección y número de teléfono del propietario. Dadas sus características, tiene prohibido volar de noche. Durante el día tiene estrictamente prohibido volar a menos de 3 millas náuticas o 5.5 km de un aeródromo o infraestructura para el despegue o aterrizaje de helicópteros y aviones (espacio aéreo controlado o regulado considerado restringido, peligroso o prohibido).

En esta clasificación entran los vehículos cuyos propietarios no comercializan las fotos o vídeos que toman, por lo que tampoco deberán entregarlas en forma gratuita como parte de una actividad comercial. Todas las fotos y vídeos generadas pueden ser compartidas en redes sociales y blogs siempre y cuando se adjunten los datos de identificación del piloto, mencionando que fueron tomadas para fines recreativos, de investigación o académicos.

Drone civil (Clasificación B-AD-S3/S4)

Vehículo aéreo no tripulado utilizado para actividades especiales, comerciales o multimision (vigilancia, medición, fotografía o vídeo aéreo), que pesa mas de 20 kilogramos y sujeto a la autorización de operación,

aprobación de tipo, certificado de aeronave habilidad especial, categoría experimental, matriculación y registro así como a responsabilidad civil.

Reglas

Este equipo puede despegar, volar y aterrizar lejos del alcance de la vista usando una cámara FPV (que retransmite la imagen a tierra, al Piloto en tiempo real), adicional a la que hace la toma fotográfica o de vídeo (Sony, Canon, Nikon, Black Magic, RED, Kinemini, etc.) solo por debajo de los 122 metros de altura a 30 metros de distancia de personas, vehículos y edificios.

Este vehículo aéreo deberá estar equipado con medios para poder conocer su posición exacta, un sensor barométrico para determinar la altitud y automáticamente evitar exceder la altura permitida así como con un dispositivo capaz de forzar el retorno al punto de partida, en el caso de pérdida de control del piloto. Al igual que el Drone de clasificación A-AD-2014, este deberá portar una etiqueta/placa de 10x5 cm que muestre el nombre, dirección y número de teléfono del piloto.

Tiene estrictamente prohibido volar a menos de 3 millas náuticas o 5.5 km de un aeródromo o infraestructura para el despegue o aterrizaje de helicópteros y aviones (espacio aéreo controlado o regulado, restringido, peligroso o prohibido).

Todas las vías de desarrollo que se han investigado en el presente proyecto en el apartado trabajos futuros, necesitan de una normativa para integrar de manera segura los UAV en el espacio aéreo civil. Estas aplicaciones pueden interesar incluso a los propios gobiernos ya que podrían ser misiones de vigilancia de fronteras, recogida datos o patrulla marítima.

Los pioneros en la regulación sobre el espacio aéreo son Suiza, Reino Unido o Australia, legislan las operaciones aéreas mediante normas nacionales. En otros países las operaciones de cualquier UAV requiere de una autorización específica por parte de las autoridades aeronáuticas.

Los principales organismos de gestión y control del tráfico aéreo son, FAA (Estados Unidos) y EASA (Europa) están realizando estudios sobre cómo integrar las aeronaves no tripuladas en sus respectivos espacios aéreos, y por tanto establecer unos mínimos sobre los que certificar estas aeronaves.La FAA, en conjunción con la Universidad del MIT, estudia dirigir la regulación de las aeronaves no tripuladas realizando estudios de tamaño, masa, probabilidad de fallo o probabilidad de impacto contra aeronaves tripuladas. Según estos estudios, las aeronaves inferiores en masa 14kg, donde estaría el Erle-Copter, serían reguladas mediante una serie de normas para el uso civil. Estas normas, serían similares a las reglas seguidas por los aficionados de R/C, no registradas como leyes, y sí como normas de clubes de R/C, basadas en consejos de la FAA. Estas normas, establecen cotas de operación muy bajas, mantener siempre el contacto visual con la aeronave y se restringe su uso fuera del espacio aéreo controlado, lejos de zonas "problemáticas" como sendas de aproximación y despegue de aeropuertos.

Para los UAV de peso superior establecen unas normas homogéneas a las ya existentes para el control del espacio aéreo, aplicando las mismas normas que para aeronaves tripuladas que operan en VFR, como son los ultraligeros. Otra opción sería la de operaciones de UAS del tipo HALE, que vuelan a una cota superior al espacio aéreo que usan los vuelos comerciales.

2.3. Normativa Europea

Por otro lado en Europa, se está trabajando para que UAS de gran tamaño puedan utilizar el espacio civil en 2016. La Unión Europea trabaja para el desarrollo de las aplicaciones civiles de los Sistemas Aéreos Pilotados Remotamente.

Las única regulación existente son las normas de conductas emitidas por las asociación AUVSI, Asociación Internacional de Vehículos no Tripulados. Estas normas buscan realizar unas directrices para concebir seguridad, y acelerará la confianza pública en estos sistemas. Las normas emitidas por la AUVSI son las siguientes.

Seguridad

- No operaremos UAS de forma que represente riesgo para las personas o propiedades en la superficie o en el aire.
- •Aseguraremos que los UAS serán pilotados por personas que están debidamente entrenadas y tienen competencias para operar vehículos o sus sistemas.
- Aseguraremos que los vuelos de los UAS serán realizados solo después de una rigurosa valoración de los riesgos asociados con la actividad. Esta valoración de riesgos, incluirá, pero no limitará.
- Condiciones climáticas en relación con la capacidad de los sistemas.
- Identificación de normalidad de modos de fallo anticipado (perdida de enlace, fallos de potencia, pérdida de control, etc.) y las consecuencias de los fallos.
- Condiciones físicas de la tripulación para operar el vuelo.
- Cumplimiento de las regulaciones de aviación y de la apropiada operación en el espacio aéreo y procedimientos no nominales.
- Comunicación, comando, control y requisitos de espectro de frecuencia del palead (carga útil).
- Fiabilidad, rendimiento y aeronavegabilidad mediante estándares establecidos.

Profesionalidad

- Cumpliremos con las leyes nacionales, estatales y locales, ordenanzas, acuerdos y restricciones relativos a las operaciones de UAS.
- Operaremos nuestros sistemas como miembros responsables de la comunidad aeronáutica.
- Seremos sensibles a las necesidades de las personas.
- Cooperaremos totalmente con las autoridades nacionales, regionales y locales en el despliegue en respuesta a emergencias, investigación de accidentes y relaciones con los medios.
- Estableceremos planes de contingencia para todos los eventos anticipados y los compartiremos abiertamente con todas las autoridades pertinentes.

Respeto

- Respetaremos los derechos de otros usuarios en el espacio aéreo.
- Respetaremos la privacidad de las personas.
- Respetaremos los asuntos del público así como los relativos a las operaciones de los UAS.
- Apoyaremos la mejora del conocimiento y educación pública sobre las operaciones de los UAS.

2.4. Modelado de un VANT

Para obtener el modelo dinámico basado en ecuaciones que describan la posición y orientación, se supone el cuadrotor como un cuerpo rígido en el espacio, sujeto a una fuerza principal (empuje) y tres momentos (pares).

A continuación, de una forma más detallada, se describen las fuerzas y pares actuantes sobre la plataforma.

El par para generar un movimiento de balanceo o de roll (ángulo φ) se realiza mediante un desequilibrio entre las fuerzas f 2 y f 4. Para el cabeceo o pitch (ángulo θ), el desequilibrio se realizará entre las fuerzas f 1 y f 3 . El movimiento en el ángulo de guiñada o de yaw (ángulo Ψ) se realizará por el desequilibrio entre los conjuntos de fuerzas (f 1 y f 3.) y (f 2 y f 4.). Se debe a que los rotores 1 y 3 giran en sentido contrario a los rotores 2 y 4. El empuje total produce que el cuadrotor se desplace perpendicularmente al plano de los rotores, y se obtiene como suma de las cuatro fuerzas que ejercen los rotores.

Los cuadrotor suelen ser sistemas de vuelo de estructura ligera, por lo que el modelo dinámico debe incluir los efectos giroscópicos resultantes tanto del cuerpo rígido rotando en el espacio, como de la rotación de las cuatro hélices. En el cuadro 2.1 se describen tales efectos, donde C representan términos constantes, Ω es la velocidad del rotor, JR es el momento de inercia rotacional del rotor alrededor de su eje, l es la distancia del centro de masa a los rotores, J es el momento de inercia del cuerpo rígido y Φ, θ, Ψ son los denominados ángulos de Tait-Bryan.

C = Representa términos constantes

Ω= Velocidad del rotor

JR= El monto de inercia rotacional del rotor alrededor de su eje

L= Es la distancia del centro de masa a los rotores

J= Es el momento de inercia del cuerpo rígido

Φ, θ, Ψ = Ángulos de Tait-Bryan.

Cuadro 2.1: Efectos físicos actuales

EFECTOS	FUENTES	FORMULACIÓN
Efectos Aerodinámicos	Rotación de los rotores - giro de hélices	$C\Omega^2$
pares inerciales opuestos	cambio en la velocidad de rotación de los rotores	JRΩ
efecto de la gravedad	posición del centro de masa	L
efectos giroscópicos	cambio en la orientación del cuerpo rígido	JR$\Omega\theta$, φ
fricción	todos los movimientos del helicóptero	C(θ, φ, Ψ)

El cuadrotor es un sistema mecánico subactuado de 6 grados de libertad y 4 entradas de control. A continuación, de una forma mas detallada, se describen la fuerzas y pares actuantes sobre la plataforma.

En esta sección se presenta la estimación de la orientación y posición inercial del vehículo. El cuadrotor, como solido rígido, esta caracterizado por un sistema de coordenadas ligado a él y con origen en su centro de masas. El sistema de ejes ligado a él está determinado por B=$\left\{\overrightarrow{xL}, \overrightarrow{yL}, \overrightarrow{zL}\right\}$,donde ele eje $\overrightarrow{xL}$ es la dirección normal de ataque del helicóptero, $\overrightarrow{yL}$ es ortogonal a $\overrightarrow{xL}$ y es positivo hacia la derecha mirando en el sentido positivo de éste, mientras que $\overrightarrow{zL}$ está orientado en sentido ascendente y ortogonal al plano $\overrightarrow{xL}$ O $\overrightarrow{yL}$. El sistema de coordenadas inercial I =$\{\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{z}\}$se considera fijo con respecto a la tierra. Se determina $\xi = \{x, y, z\}$como la posición del centro de masa del helicóptero con respecto al sistema inercial I, y la orientación del vehículo se supondrá dada o por una matriz de rotación R_I: B I , donde R_I ϵSO es una matriz de rotación ortogonal.

La rotación de un cuerpo rígido puede ser obtenida mediante ángulos de Euler, o cuaterniones, por ejemplo. Existen varias definiciones de los ángulos de Euler para representar la orientación relativa de dos sistemas de coordenadas. La más popular en ingeniería aeroespacial es la convención x, y, z (giro alrededor de x, y' , z") y es conocida como ángulos de Tait-Bryan, también conocidos por ángulos "cardano".

Los ángulos de Tait-Bryan se utilizan para describir una rotación general en el espacio Eucledio tridimensional a través de tres rotaciones sucesivas en torno de ejes del sistema móvil en el cual están definidos. Así, podremos utilizar los ángulos de Tait-Bryan para describir la orientación del cuadrotor.

La rotación de un cuerpo rígido en el espacio se realiza mediante tres rotaciones sucesivas.

1. Rotación según $\overrightarrow{x}$ de φ: el primer giro es el correspondiente al ángulo de roll o de balanceo, φ, y se realiza alrededor del eje $\overrightarrow{x}$.

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} XL \\ yL \\ zL \end{bmatrix} \tag{2.1}$$

2. Rotación según $\overrightarrow{y}$ de θ: el segundo giro se realiza alrededor del eje $\overrightarrow{y}$ y a partir del nuevo eje , $\overrightarrow{yL}$ con el ángulo pitch o de cabeceo, θ, para dejar el eje $\overrightarrow{zL}$ en su posición final.

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \tag{2.2}$$

3. Rotación según $\overrightarrow{z}$ de Ψ: el tercer giro y la ultima rotación corresponde al ángulo yaw o de guiñada, Ψ, y se realiza al rededor del eje $\overrightarrow{z}$ a partir del nuevo eje $\overrightarrow{zL}$ para dejar el cuadrotor en su posición final.

$$\begin{bmatrix} x_3 \\ y_3 \\ z_3 \end{bmatrix} = \begin{bmatrix} x_3 \\ y_3 \\ z_3 \end{bmatrix} \begin{bmatrix} \cos\Psi & -\sin\Psi & 0 \\ \sin\Psi & \cos\Psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} \tag{2.3}$$

Las matrices de rotación definen algebraicamente lo que es una rotación en un espacio 3D considerando un ángulo en el que está girando, las matrices de rotación tienen unas propiedades que son importantes de notar:

- Sus ejes de coordenadas son vectores ortogonales (forman un ángulo de 90 grados entre ellos).

- Su determinante es 1

- Si se saca la normal de cualquier vector perteneciente a la matriz el resultado es 1 por lo que es una matriz unitaria

- Al ser una matriz ortogonal su transpuesta es igual a su inversa Pero ¿Para qué usarlo en la robótica?, la respuesta a esta pregunta es que la matriz de rotación define los movimientos de objetos rígidos, por lo que es ideal para su uso en la robótica. Dentro de la robótica podemos pensar que un objeto puede girar sobre diferentes ejes, prácticamente siempre en 3D, por lo que podemos introducir dos nuevos conceptos el de marco global y el de marco de referencia.

El marco global es el punto en donde comienza o está su robot y el marco de referencia es con respecto a que esta girando de manera que se puede obtener coordenadas con respecto a cualquiera de estos dos marcos, en pocas palabras dan posición del robot. Para obtener estas coordenadas se usan transformaciones que involucran a la matriz de rotación.

Las matrices de rotación que representan la orientación del cuerpo rígido rotando alrededor de cada eje y queda expresada como:

$$R\left(x,\phi\right)=\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\phi & -\sin\phi\\ 0 & \sin\phi & \cos\phi\end{bmatrix} \tag{2.4}$$

$$R\left(x,\theta\right)=\begin{bmatrix}\cos\theta & 0 & \sin\theta\\ 0 & 1 & 0\\ -\sin\theta & 0 & \cos\theta\end{bmatrix} \tag{2.5}$$

$$R\left(x,\Psi\right)=\begin{bmatrix}\cos\Psi & -\sin\Psi & 0\\ \sin\Psi & \cos\Psi & 0\\ 0 & 0 & 1\end{bmatrix} \tag{2.6}$$

La matriz de rotación completa de B respecto a I, llamada matriz de cosenos directores, viene dada por. R I = R(z, Ψ).R(y,θ).R(x, ϕ)

$$R_I=\begin{bmatrix}\cos\Psi & -\sin\Psi & 0\\ \sin\Psi & \cos\Psi & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\cos\theta & 0 & \sin\theta\\ 0 & 1 & 0\\ -\sin\theta & 0 & \cos\theta\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\phi & -\sin\phi\\ 0 & \sin\phi & \cos\phi\end{bmatrix} \tag{2.7}$$

$$R_I=\begin{bmatrix}\cos\Psi\cos\theta & \cos\Psi\sin\theta\sin\phi-\sin\Psi\cos\theta & \cos\Psi\sin\theta\cos\phi+\sin\Psi\sin\phi\\ \sin\Psi\cos\theta & \sin\Psi\sin\phi\sin\theta+\cos\Psi\cos\phi & \sin\Psi\sin\theta\cos\phi-\cos\Psi\sin\phi\\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi\end{bmatrix} \tag{2.8}$$

La matriz de rotación expresada en el sistema de coordenadas B es la traspuesta de R_I , por su propiedad ortogonal, y viene dada por :

$$R_B=\begin{bmatrix}\Psi\cos\theta & \sin\Psi\cos\theta & -\sin\theta\\ \cos\Psi\sin\theta\sin\phi-\sin\Psi\cos\phi & \sin\Psi\sin\theta\sin\phi+\cos\Psi\cos\phi & \cos\theta\sin\phi\\ \cos\boldsymbol{\Psi}\sin\theta\cos\phi+\sin\Psi\sin\theta & \sin\Psi\sin\theta\cos\phi-\cos\Psi\sin\phi & \cos\theta\cos\phi\end{bmatrix} \tag{2.9}$$

A partir de la matriz de rotación, se pueden obtener las ecuaciones cinemáticas de rotación del cuadrotor que establecen las relaciones entre las velocidades angulares.

Se una matriz ortogonal R,donde:

$$R^T R = I_n \tag{2.10}$$

y su derivada respecto al tiempo es:

$$\dot{R}^T R + R^T \dot{R} = 0_n \tag{2.11}$$

definiendo:

$$S = R^T \dot{R} \tag{2.12}$$

se obtiene a partir de la ec.anterior que:

$$S^T + S = 0_n \tag{2.13}$$

Donde S es anti-simétrica. La relación entre la derivada de la matriz ortogonal y la matriz anti-simétrica es la siguiente:

$$S = R^{-1} \dot{R} \tag{2.14}$$

Por lo tanto, las ecuaciones cinemáticas para determinar la orientación del helicóptero suponiendo la matriz de rotación, vienen dadas por:

$$\dot{R}_I = R_I . S(\omega) \tag{2.15}$$

Donde $\omega = [p.q.r]^T$son las velocidades angulares en el sistema de coordenadas ligado al cuerpo rígido y $\mathbf{S}(\omega)$ ($\mathbf{S}(\omega)(\mathbf{.}) = \omega x \mathbf{.}$)es la siguiente matriz anti-simetrica.

$$\mathbf{S}(\omega) = \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} \tag{2.16}$$

Manipulando matemáticamente se obtiene:

$$\begin{bmatrix} \phi \\ \theta \\ \Psi \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\theta \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{2.17}$$

La variación de los ángulos de Tait-Bryan(ϕ, θ, Ψ) es una función discontinua. Hay que distinguirla de las velocidades angulares en el sistema de referencia asociado al cuerpo rígido (p, q, r) las cuales pueden medirse con giróscopos. Normalmente, se utiliza la IMU, de inglés Inertial Measurement Unit o Unidad de Medida Inercial, para medir las rotaciones y calcular los ángulos de Tait-Bryan.

La relación entre las velocidades angulares en el sistema fijado al cuerpo y la variación en el tiempo de los ángulos de Tait-Bryan se obtiene a traces de la inversión del Jacobiano de la ecuación (2.14) y viene dada por:

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\sin\phi \\ 0 & \cos\phi & \sin\phi\cos\theta \\ 0 & -\sin\phi & \cos\phi\cos\theta \end{bmatrix} \begin{bmatrix} \phi \\ \theta \\ \Psi \end{bmatrix} \tag{2.18}$$

El movimiento rotacional del cuadrotor viene dado por las componentes de las velocidades angulares: velocidad angular de balanceo (p) , velocidad angular de cabeceo (q), y velocidad angular de guiñada (r), sobre los ejes $\overrightarrow{x_L}, \overrightarrow{y_L}$y $\overrightarrow{z_L}$ respectivamente. Las velocidades angulares son debidas a los pares ligados al cuerpo rígido producidos por las fuerzas externas, las cuales definen momentos en los tres ejes: momento de balanceo (L), momento de cabeceo (M), y momento de guiñada (N) sobre los ejes $\overrightarrow{x_L}, \overrightarrow{y_L}$ y $\overrightarrow{z_L}$ respectivamente.

El movimiento de traslación viene dado por la velocidad $\mathbf{v}=[u_0 v_0 w_0]^T$n ejes inerciales, y relacionada con la velocidad absoluta del helicóptero expresada en **B,** $\mathbf{V}=[u_L v_L w_L]$mediante la expresión: Con base a las fuerzas y los momentos que actúan en un cuadrotor, se realizo una estimación del modelo dinámico del cuadrotor Figura. 2.7.

Partiendo de la matriz de rotación R I mencionada anteriormente se tiene que la dinámica de traslación es :

$$\sum F_1 = mV_1 \tag{2.19}$$

donde: $V_1 = (u_x u_y u_z)$
de igual manera la dinámica de rotación es expresada de la siguiente manera:

$$\sum M_B = j\Omega + \Omega x j\Omega \tag{2.20}$$

donde j=momento de inercia, $\Omega = (p, q, r)$, vector de velocidad de rotación
Matriz de inercia:

$$j = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix} \tag{2.21}$$

velocidad de rotación:

$$\Omega = \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \sin\phi\cos\theta \\ 0 & -\sin\phi & \cos\phi\cos\theta \end{bmatrix} \tag{2.22}$$

entradas T $= f_1 + f_2 + f_3 + f_4$ peso = fuerza de empuje
momentos de los cuadro motores:
$\tau\phi = i\,(f_2 - f_4)$
$\tau\theta = i\,(f_1 - f_3)$
$\tau\Psi = -\tau_{R1} + \tau_{R2} - \tau_{R3} + \tau_{R4}$
equilibrio de fuerzas y momentos:

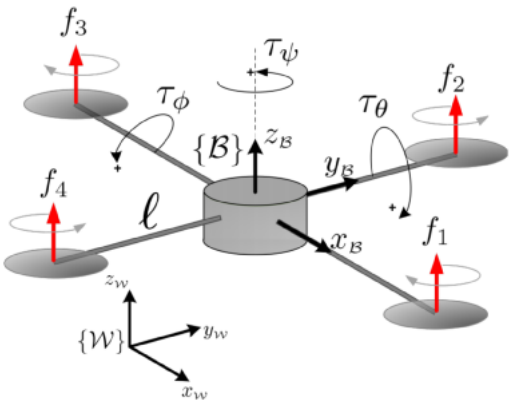

Figura 2.7: fuerzas y momentos

$$\sum F_1 = \begin{pmatrix} 0 \\ 0 \\ mg \end{pmatrix} + R_B \begin{pmatrix} 0 \\ 0 \\ -T \end{pmatrix} + F_A + F_D$$

$$\sum M_B = \begin{pmatrix} L_\phi \\ L\theta \\ L\Psi \end{pmatrix} + \begin{pmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\Psi \end{pmatrix} + \tau_A + \tau_D$$

donde:

$$R_B \begin{pmatrix} 0 \\ 0 \\ -T \end{pmatrix} = \begin{pmatrix} (-\cos\Psi \sin\theta \cos\phi - \sin\Psi \sin\phi)\, T \\ (\cos\Psi \sin\phi - \sin\Psi \sin\theta \cos\phi)\, T \\ (-\cos\theta \cos\phi)\, T \end{pmatrix}$$

de la dinámica de traslación $V_I = \begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix}$

$$\begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix} m = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} + \begin{bmatrix} (-\cos\Psi \sin\theta \cos\phi - \sin\Psi \sin\phi)\, T \\ (\cos\Psi \sin\phi - \sin\Psi \sin\theta \cos\phi)\, T \\ (-\cos\theta \cos\phi)\, T \end{bmatrix} + \begin{bmatrix} F_{Ax} \\ F_{Ay} \\ F_{Az} \end{bmatrix}$$

de donde obtenemos:

$$V_x = F_{Ax} - [\cos\Psi \sin\theta \cos\phi - \sin\Psi \sin\phi + \sin\Psi \sin\phi] \frac{T}{m} \tag{2.23}$$

$$V_y = F_{Ay} + [\cos\Psi \sin\phi - \sin\Psi \sin\theta \cos\phi] \frac{T}{m} \tag{2.24}$$

$$V_z = F_{Az} + g - \cos\theta \cos\phi \left[\frac{T}{m}\right] \tag{2.25}$$

de la dinámica de rotación:

$\sum M_B = j\Omega + \Omega x j\Omega$

$$\mathrm{j}\Omega = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} p & I_x \\ q & I_y \\ r & I_z \end{bmatrix}$$

$$\Omega x j \Omega = \begin{bmatrix} i & -j & k \\ p & q & r \\ pI_x & qI_y & rI_z \end{bmatrix} = q_r\left(I_z - I_y\right) + p_r\left(I_x - I_z\right) + p_q\left(I_y - I_x\right)$$

$\therefore$obtenemos lo siguiente.

$$\dot{p} = \frac{\tau_\phi}{I_x} + \frac{[I_y - I_z]}{I_X} q_r + \tau_{Ax} + \frac{I_r}{I_x} q\Omega r \tag{2.26}$$

$$\dot{q} = \frac{\tau_\phi}{I_x} + \frac{[I_y I_z]}{I_X} q_r + \tau_{Ay} + \frac{I_r}{I_y} q\Omega_r \tag{2.27}$$

$$\dot{r} = \frac{\tau_\phi}{I_x} + \frac{[I_y - I_z]}{I_X} q_p + \tau_{Az} \tag{2.28}$$

de la matriz de velocidad de rotación Ω se obtiene un sistema de 3 ecuaciones y 3 incógnitas (p,q,r) el cual obtenemos (ϕ,θ, Ψ).

$$\Omega = \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \sin\phi\cos\theta \\ 0 & -\sin\phi & \cos\phi\cos\theta \end{bmatrix} \begin{bmatrix} \phi \\ \theta \\ \Psi \end{bmatrix}$$

se obtiene $p = \phi - \Psi\sin\theta$,$q = \theta\cos\phi + \Psi\cos\phi,\ r = -\theta\sin\phi + \phi\cos\theta\cos\phi$

obtenemos:

$$\phi = r\tan\theta\cos\phi + p + q\tan\theta\sec\phi \tag{2.29}$$

$$\theta = q\cos\phi - r\sin\phi \tag{2.30}$$

$$\Psi = r\sec\theta\cos\phi + q\sec\theta\sec\phi \tag{2.31}$$

Capítulo 3

Hardware y software del sistema

3.1. Erle-copter

El Erle-copter,(figura 3.1) de la compañía española erle robotics es un vehículo aéreo no tripulado radiocontrolado de software libre para uso recreativo civil e investigación, funciona propulsado por cuatro motores eléctricos en configuración cuadricóptero y es similar en su estructura básica y aerodinámica a otros modelos de radiocontrolados, pero se diferencia de todos ellos en que cuenta con un microprocesador y una serie de sensores, como lo es un conector WI-FI integrado que le permite vincularse a un ordenador con sistema operativo Linux en especifico Ubuntu. Esto permite manipular al cuadrotor directamente desde el ordenador, así mismo permite accesar a telemetría de los sensores que trae integrado.

El Erle-copter es el primer VANT inteligente basado en Linux Snappy Ubuntu Core para ser mas específicos, utiliza un sistema robotico como ROS y el software APM autopilot para cambiar a diferentes modos de vuelo.

El Erle-copter es ideal para operaciones en exteriores y esta diseñado para extender el tiempo de vuelo para un despegue con una carga de 2 kilogramos. debido que es de software libre se puede acceder.

Figura 3.1: Erle-copter.

3.2. Especificaciones técnicas del hardware

El núcleo de este Erle-copter es el erle-brain esta compuesta por un pixhawk-fire v1.6 y un beaglebone black (BBB),(figura 3.2)cuenta con procesador Sitara AM3358BZCZ100 de 1GHz, 2000 MIPS. también dispone de una memoria RAM DDR3L 800MHz de 512 Mb, así como una expansión de una microSD de 8Gb de almacenamiento. Cuenta con un HS USB2.0 con acceso por usb0 como cliente vía usb0 por SSH, una conexión WI-FI ac. con una frecuencia de 5Hz ajustable a 2.4Hz para conectar vía SSH. un puerto Ethernet 10/100, RJ45, conector microSD de 3,3V y un motor gráfico SGX530 3D 20M polygons/S.

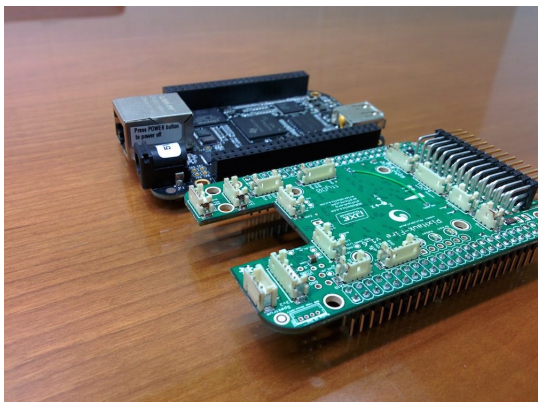

Figura 3.2: Pixhawk v1.6 y beaglebone black (erle-brain)

3.2.1. Sistema de propulsión

El Erle-copter es un vehículo aéreo con movimiento físico propulsado por cuatro motores eléctricos en una estructura aerodinámica de cuadricóptero.figura 3.3.

Figura 3.3: Tiger motors brushless

Los cuatro motores de rotor interno sin escobillas(figura 3.4) hacen que el dispositivo tenga alta velocidad de reacción en el aire y pueda, dentro de su categoría de vuelo, ascender a alta distancia. la fuente de alimentación de la que dispone es una batería de 3 celdas de Litio-Polímero con capacidad de 5500 mAh a 11.1 V y capacidad de descarga de 35C, cumple con la normativa de seguridad UL2054. el cumplimiento con la norma UL2054 hace viable una evolución del aparato hacia un vuelo de mayor longitud y tiempo.

Figura 3.4: Motor brushless

Estos cuatro motores son los responsables, a nivel de hardware, para que se mantenga en el aire, con la ayuda de los comandos a nivel de software emitidas por el sistema de estabilización.

La cantidad de KV caracteriza la propiedad de un motor de transformar la potencia eléctrica que

recibe en velocidad, o en par. a mayor kV mayor velocidad y menos par; a menor kV mayor par y menor velocidad.

3.2.2. Sensores

Para que el Erle-copter pueda volar se necesita controlar la dinámica rotacional y translacional. los sensores giroscópicos y acelerómetros se utilizan para estabilizar la dinámica rotacional y para controlar el vehículo en el espacio se emplea el GPS y el compass. con estos sensores se puede hacer un control por seguimiento de trayectoria.

También cuenta con un sensor barométrico el cual sirve para tener una altura relativa del vehículo con una precisión de ±10 Pa.

El acelerómetro digital de 3 ejes con precisión de ±50mg para monitorizar los movimientos de posición

giroscopio de 2 de 2 ejes y giroscopio piezoeléctrico de precisión de 2000o/s para movimientos de Roll, Pitch y Yaw.figura 3.4.

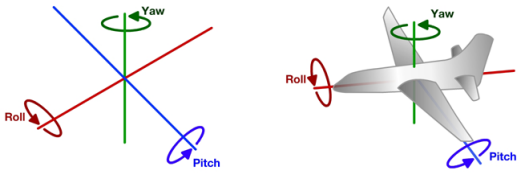

Figura 3.5: roll,pitch, yaw

El GPS es un NEO-M8N GPS Module Built-in Electronic Compass For APM Pixhawk con alta precisión y bajo consumo de energía, la presión es de 0,6 metros y la prueba más práctica en 0,9 metros más o menos. alta velocidad de búsqueda soporta Europe's Galileo ,Russia's GLONASS ,japan's small sun, China's big dipper, tiene una taza de muestreo de 10 hz.

El conjunto de estos sensores proporcionan toda la información al Erle-copter para la navegación. la fusión de los datos de los sensores proporciona los ángulos de Euler[3] figura 3.5, utilizados para la estabilización rotacional y translacional.

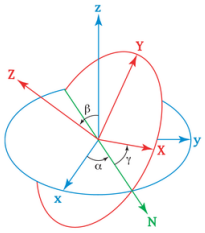

Figura 3.6: ángulos de Euler

3.3. ROBOT OPERATION SYSTEM (ROS)

ROS fue originalmente desarrollado en 2007 por el Laboratorio de Inteligencia artificial de Stanford (SAIL) con el soporte del proyecto Standfor AI Robot. Es 2008 se continuó el desarrollo en Willow 19Ga-

rage, una institución de investigación robótica, con más de 20 instituciones colaborando en el desatollo. En febrero de 2013, ROS se transfirió a la Open Source Robotics Foundation.

ROS está liberada bajo los términos de la licencia BSD (Berkeley Software Distribution) y un software open source. Este software gratuito para el uso comercial y de investigación y se podrá consultar en [9].

ROS promueve la reutilización de código, así los desarrolladores y científicos no tienen que reinventar la rueda todo el tiempo. Con ROS, puedes hacer esto y mucho más. Puedes coger el código de los repositorios, mejorarlo y compartirlo de nuevo.

Permite introducir librerías y herramientas para ayudar al desarrollo del software de aplicaciones en el campo de la robótica. Gracias, entre otras cosas, a la abstracción de hardware, los controladores de dispositivo, bibliotecas, visualizadores, paso de mensajes, o gestión de paquetes son algunas de las funciones que nos permite.

Debido a la falta de normativa para el vuelo de estos aparatos, micro UAV, al cual pertenece el Ardrone, véase Normativa UAV. Hemos centrado la elección de la vía de desarrollo del sistema, en base a la licencia libre del software.

Actualmente ROS está bajo licencia de código abierto, la licencia BSD, la licencia BSD tiene menos restricciones en comparación con otras como GPL, estando muy cercana al dominio público. Permite el uso del código fuente en software no libre.

3.3.1. ¿Qué es ROS?

ROS es un meta-sistema operativo de código abierto para el desarrollo en el sector de la Robótica. Provee diferentes servicios que se esperarían de un sistema operativo, como abstracción de hardware, control de dispositivos en bajo nivel, funciones que se usan comúnmente, comunicación de procesos mediante mensajes y administración por paquetes.

Pero decimos que ROS es un meta-sistema operativo porque además, también comparte características con algunos sistemas middleware (acoplamiento clasificación y envío por el paso de mensajes) y frameworks (callbacks).

Comparando ROS con la mayoría de sistemas middleware y frameworks, ROS impone una leve política restrictiva en el desarrollador, tanto en términos de API, como en problemas de licencia.

La comunidad de este meta-sistema operativo aumenta dado que la curva de aprendizaje no es demasiado alta, y la mayoría del código puede ser fácilmente trasladado, una vez que se entiende lo básico ROS. Fig. X

ROS nos proporciona herramientas y librerías que permiten obtener, construir, escribir y ejecutar programas entre varios computadores, como se ha definido en el apartado anterior, es similar en algunas características con otros frameworks usados en robótica, tales como Player, Orocos, Yarp, Orca, Microsoft Robotics Studio, entre otros.

3.3.2. ¿PORQUE ROS?

ROS es el SDK definitivo para el desarrollo de aplicaciones de robots. Proporciona una arquitectura distribuida y contiene algoritmos implementados del estado del arte, mantenidos por expertos en el campo. Todo con una licencia permisiva que en unos pocos años cambiará el panorama de la robótica y es ampliamente adoptado por la investigación, centros educativos y la industria.

Tenga en cuenta que los nodos de ROS no tienen que estar en el mismo sistema (varios ordenadores) o incluso ser de la misma arquitectura!. Se puede tener un Erle-Brain publicando mensajes y un ordenador portátil suscribirse a ellos. ROS también es de código abierto y es mantenido por muchas personas. Esto hace a ROS muy flexible y adaptable a las necesidades del usuario.

3.3.3. Objetivo de ROS

El objetivo de ROS es el de implementar todas las conexiones a nivel bajo, para acelerar el proceso de desarrollo, e incluso poder portar las aplicaciones a otros robots gracias a su sistema general de mensajes y nodos. La posibilidad de portar las aplicaciones es una gran ventaja, ya que avances que se realicen para el Ardrone, pueden funcionar con otros dispositivos.

3.3.4. Conceptos generales de ROS

Para entender el sistema ROS debemos conocer unos sencillos conceptos propios de este sistema. Aun- que ROS actualiza anualmente la versión del software estos conceptos no cambian, manteniendo su estructura.

ROS posee tres niveles de conceptos:

- Nivel del sistema de archivos.
- Nivel de Computación Gráfica.
- Nivel comunitario

3.3.5. ROS nivel del sistema de archivos

En este nivel nos referimos a la totalidad de archivos necesarios para el funcionamiento de ROS, no nos referimos al sistema de conexiones o envío de mensajes a nivel interno.

Paquetes: unidad principal en ROS, puede contener procesos de ejecución (nodos), una biblioteca ROS dependiente, conjuntos de datos, archivos de configuración, o cualquier otro archivo que se útil para la organización de ROS.

Manifiestos: residentes en manifesté.xml proporcionan datos sobre un paquete, incluyendo su información de licencia y dependencias, así como información específica del idioma.

Pilas: Pilas o stacks son colecciones de paquetes que proporcionan funcionalidad agregada, como una "stack de navegación".

Manifiestos de pila o stack: se trata del archivo stack.xml, proporcionan datos sobre una pila, incluyendo su información de licencia y sus dependencias en otras pilas.

3.3.6. Nivel de computación gráfica

El gráfico de la computación es la red peer-to-peer de los procesos de ROS que están procesando datos conjuntamente. Se trata de un gráfico bastante útil en el desarrollo, ya que de una forma sencilla, podemos observar las conexiones internas en ROS. Los conceptos básicos de computación gráfica en ROS, nos proporcionan los datos de la gráfica de diferentes maneras.

3.3.6.1. ROS máster

ROS empieza con el ROS MÁSTER. el máster permite todas las demás piezas del software ROS para encontrar y comunicarse el uno al otro. fuera del máster, los nodos no serian capaces de encontrar cada mensaje de otra, de cambio o invocar servicios. de esta manera no tiene que indicar siempre específicamente el estado "enviar datos de este sensor al equipo 127.0.0.1" para eso solo podemos decir nodo1 para enviar mensajes al nodo 2.

el ROS máster proporciona el registro de nombres y de consulta para el resto de la computación gráfica.Fig. 3.6

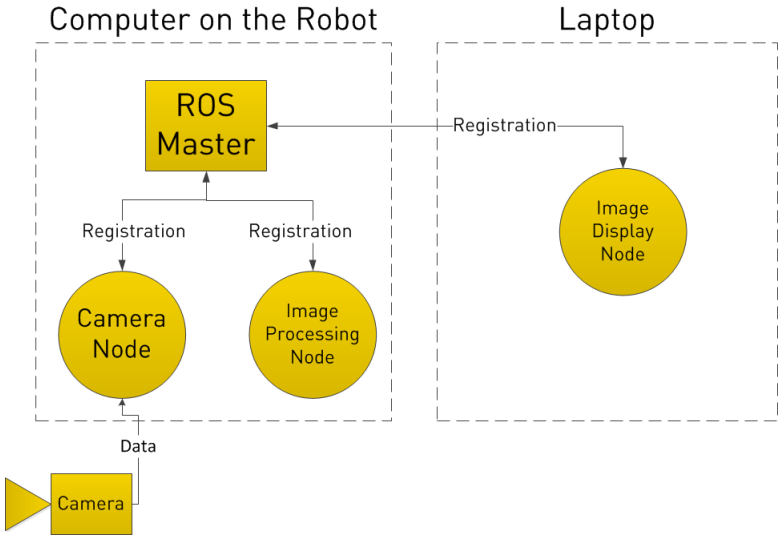

Figura 3.7: ROS MÁSTER

3.3.6.2. Nodo

Los nodos son procesos que llevan a cabo la computación. ROS diseñado para ser moduladora gran escala; usualmente el sistema de control robotico comprende, por lo general muchos nodos. Por ejemplo, nodo controla una telemetría por láser, un nodo controla la velocidad de un motor, un nodo lleva a cabo la localización, un nodo proporciona una vista gráfica del sistema, un nodo lleva a cabo la planificación de la trayectoria y así sucesivamente.

Un nodo ROS se escribe con el uso de una biblioteca de cliente ROS, tales como roscpp o rospy.

Los nodos son ejecutables que pueden comunicarse con otros procesos utilizados temas,servicios o el servidor de parámetros. El uso de nodos en ROS nos proporciona tolerancia a fallas y separa código y funcionalidades que el sistema sea mas simple. Un nodo debe tener un nombre único en el sistema. Este nombre se utiliza para permitir que el nodo para comunicarse con otro nodo utilizando su nombre sin ambigüedad. Un nodo puede ser escrito utilizando diferentes bibliotecas como roscpp para C++ y rospy para Python.

ROS tiene herramientas para manejar los nodos y nos dará información sobre el mismo, como rosnode. Algunos comandos con rosnode son:

- Rosnode info node. Esto imprime información sobre el nodo

- Rosnode kill node. Esto detiene el nodo o el envío de la señal
- Rosnode list. Esto enlista los nodos activos.
- Rosnode machine hostname. Esto enlista los nodos que se están ejecutando sobre una maquina en particular o enlista las maquinas.
- Rosnode ping node. Esto pone a prueba la conectividad con el nodo.
- Rosnode cleanup. Esto purga la información de registro de nodos inalcanzables

3.3.6.3. Servidor de parámetros

Los nodos se comunican unos con otros pasando mensajes. un mensaje es simplemente una estructura de datos que comprende los datos mecanografiadas. Se admiten tipos primitivos estándar (entero, punto flotante, booleano,etc), así como los arreglos de tipos primitivos. Los mensajes pueden incluir estructuras anidadas arbitrariamente y matrices (muy similares a estructuras C).

3.3.6.4. Topics o Temas

los mensajes se enrutan a través de un sistema de transporte con la publicación/suscripción semántica. Un nodo envía un mensaje mediante su publicación a un tema determinado. el tema es un nombre que es usado para identificar el contenido del mensaje. Un nodo que esta interesado en un determinado tipo de datos, suscribirá con el tema apropiado. Pueden haber múltiples publicaciones y subscripciones para un simple topic o tema. Y un simple nodo puede publicar y/o suscribir en múltiples topics. En general, los editores y suscriptores no son conscientes de la existencia de los demás. La idea es disociar la producción de su consumo. Lógicamente, se puede pensar en un tema como un bus de mensajes inflexible de tipos. Cada bus tiene un nombre y cualquiera puede conectarse al bus para enviar o recibir mensajes mientras son del tipo correcto.

Los topics son los buses utilizados por los nodos para transmitir datos. Los topics pueden ser transmitidos sin una conexión directa entre los nodos, es decir, la producción y el consumo de datos se desacopla. Un topic puede tener varios suscriptores. Cada topic es fuertemente tipado por el tipo de mensaje de ROS.

Los topics pueden ser transmitidos a través de TCP/IP y UPD. El transporte TCP/ basada en IP se conoce como TCPROS y utiliza la conexión TCP/IP persistente. Este es el transporte predeterminado utilizando ROS.

ROS tiene una herramienta para trabajar topics llamados rostopic. Es una herramienta de linea de comandos que nos da información sobre el tema o publica los datos directamente en la red.

- rostopic bw /topic. Esto muestra el ancho de banda utilizado por tema.
- rostopic echo /topic. Imprime los mensajes en la pantalla.
- rostopic hz /topic. Esto muestra la taza de publicación en el topic.
- rostopic info /topic. Imprime información sobre el topic activo, los topics publicados , los que está suscrito, y servicios.
- rostopic list. Imprime información sobre topics activos, los topics publicados

- rostopic pub /topic type args. Esto publica datos del topic. nos permite crear y publicar los datos en cualquier tema que queremos, directamente desde la linea de comandos.
- rostopic type /topic. Esto imprime el tipo de topic, es decir, el tipo de mensaje que publica.

3.3.6.5. Servicio

El modelo de editor y suscriptor es un paradigma de comunicación muy flexible, el transporte de un solo sentido no es apropiado para la interacción solicitud / respuesta que a menudo se requiere en un sistema distribuido. Solicitud / respuesta se realiza a través de servicios, que se definen por un par de estructuras de mensajes: uno para solicitud y uno par la respuesta. Un nodo ofrece un servicio con un nombre y el cliente utiliza el servicio mediante el envío del mensaje de solicitud y espera respuesta. Las librerías de cliente ROS presentan esta interacción para el programador como si fuera una llamada a procedimiento remoto.

Cuando usted necesita comunicarse con nodos y recibir una respuesta, no se puede hacer con los topics, usted necesita hacer con servicios. Los servicios son desarrollados por el usuario y los servicios estándar no existen para los nodos. Los archivos con el código fuente de los mensajes se almacenaran en la carpeta srv.

Con rosservice, podemos enumerar y consultar los servicios. Los comandos soportados son los siguientes:

- rosservices call /services args. Esto requiere el servicio con los argumentos proporcionados.
- rosservice find msg-type. Esto encuentra servicios por el tipo de servicios.
- rosservice info /service. Esto imprime información acerca del servicio.
- rosservice list. Esto enlista los servicios activos
- rosservice type /service. Esto imprime los tipos de servicios
- rosservice uri /service. Esto imprime los servicios ROSRPC URI.

3.3.6.6. Bags

Bags son un formato para guardar y ejecutar datos de mensajes ROS. Los bags son un mecanismo importante para el almacenamiento de datos, como los datos de un sensor, que pueden ser difíciles de recoger pero es necesario para desarrollar y probar algoritmos.

3.3.7. Áreas de aplicación de ROS

Algunas áreas de uso de ROS son:

- Publicación o subscripción de flujos de datos: imágenes, estéreo, láser, control, actuador, contacto, etc.
- Multiplexación de la información.
- Testeo de sistemas.

- Percepción
- Identificación de Objetos.
- Segmentación y reconocimiento.
- Reconocimiento facial.
- Reconocimiento de gestos
- Seguimiento de objetos.
- Comprensión de movimiento
- Visión estéreo: percepción de profundidad.
- Robots móviles.
- Agarre de objetos.

3.3.8. Instalación de ROS Índigo en Ubuntu(Linux)

Como bien se realizó la investigación detallada sobre ROS, en esta subsección se comprenderá de como instalar ROS en la versión de Ubuntu 14.04, suponiendo que ya se tiene instalado Ubuntu en esta versión, si utiliza normalmente un sistema operativo diferente y usted no tiene una partición libre en el disco duro, también puede instalar Ubuntu en una memoria USB (de arranque), en este caso, se necesitan al menos 4 GB de almacenamiento adicional [4].

Se comprende las etapas principales de la instalación de ROS índigo para Ubuntu. En primer lugar abrir desde el equipo una terminal y agregar las claves y los registros respectivos:

$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu precise main" > /etc/apt/sources.list.d/ros-latest.list'

$ wget http://packages.ros.org/ros.key -O - | sudo apt-key add -

$ sudo apt-get update

Ahora instale ROS y algunos paquetes de útiles:

$ sudo apt-get install ros-indigo-desktop-full ros-indigo-joystick-drivers python-rosinstall python-rosdep liblapack-dev libblas-dev daemontools libudev-dev libiw-dev

Crea tu carpeta de espacio de trabajo ROS

$ mkdir ~/workshop

Añadir comandos de ROS y la carpeta de espacio de trabajo a tu .bashrc : Abra en su editor de texto favorito (por ejemplo, utilizando gedit ~ / .bashrc), y anexar los siguientes dos líneas al final:

$ source /opt/ros/indigo/setup.bash

$ export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:~/workshop

Revise su instalación abriendo una nueva ventana de consola y empezar.

$ roscore

Todo está bien cuando vea la línea "iniciado servicio básico [/ rosout]" . A continuación, puede dejar de correr roscore pulsando CTRL-C .

Descargue el código fuente en su espacio de trabajo ROS.

$ cd ~/workshop

$ GIT_SSL_NO_VERIFY=1 git clone https://github.com/AutonomyLab/ardrone_autonomy.git -b indigo

$ GIT_SSL_NO_VERIFY=1 git clone https://github.com/tum-vision/autonavx_ardrone.git

3.3.9. Ejemplo práctico

Ahora que se ha instalado ROS, para empezar a usarlo se tiene el siguiente ejemplo práctico.

Usted debe proporcionar a su sistema la ruta donde está instalado ROS, abrir una nueva terminal desde su equipo y escriba el siguiente comando.

$ roscore

$ roscore: command not found

Si el comando no se ejecuta le marcara como error donde le dirá comando no encontrado, este mensaje es debido a que su sistema no sabe dónde buscar los comandos, para resolverlo, escriba el siguiente comando en una en una nueva terminal.

$ source /opt/ros/indigo/setup.bash

A continuación, escriba el comando roscore una vez más, y verá la siguiente salida:

```
... Press Ctrl-C to interrupt
started roslaunch server http://192.168.7.2:52131/
ros_comm version 1.11.10
SUMMARY
========
PARAMETERS
* /rosdistro: indigo
* /rosversion: 1.11.10
NODES
auto-starting new master
process[master]: started with pid [5437]
ROS_MASTER_URI=http://192.168.7.2:11311/
setting /run_id to 8e8dcaf5-fd39-11e4-bff2-6c400899ab38
process[rosout-1]: started with pid [5440]
started core service [/rosout]
```

Esto significa que su sistema sabe dónde encontrar los comandos para ejecutar ROS.

Una vez que hemos iniciado el ROS podemos acceder a los nodo, topics y/o servicios. muestra de ello haremos un programa en Python en el cual 2 programas estarán comunicados, uno imprimirá y publicara una secuencia de tiempo, mientras que el otro programa leerá esa secuencia y lo imprimirá en pantalla, para ello abriremos una nueva terminal y escribiremos:

$ touch publicar.py suscribir.py

$ gedit publicar.py

Al hacer esto abrirá el editor de textos y dentro copiaremos el siguiente programa en Python.

```
#!/usr/bin/env Python
```

```
import rospy
from std_msgs.msg import String
def talker():

    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():

        hello_str = "hello world%s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':

    try:

        talker()

    except rospy.ROSInterruptException:

        pass
```

Luego de haber copiado el programa lo guardaremos y regresamos a la consola, ahora haremos lo mismo con el programa suscribir.py, por lo que escribiremos

$ gedit suscribir.py

y copiaremos lo siguiente:

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import String
def callback(data):

    rospy.loginfo(rospy.get_caller_id() + "I heard%s", data.data)

def listener():

    rospy.init_node('listener', anonymous=True)

    rospy.Subscriber("chatter", String, callback)
    # "spin()" simplemente mantiene python hasta que el nodo se detiene

    rospy.spin()

if __name__ == '__main__':

    listener()
```

Guardamos el programa y regresamos a la terminal.

Ya una vez en la terminal para ejecutar los programas solo escribimos.

$ python suscribir.py

$ python publicar.py

3.4. Sercure shell (SSH)

SSH que en español significa interprete de ordenes seguro es el nombre de un protocolo y del programa que lo implementa y sirve para acceder a maquinas remotas a través de una red.permite manejar por completo la computadora mediante comandos y también puede redirigir el trafico de X para poder ejecutar programas gráficos si tenemos ejecutando un servidor X(en sistemas Unix y Windows).

Además de la conexión a otros dispositivos, SSH nos permite copiar datos de forma segura (tanto archivos sueltos como simular sesiones FTP cifradas), gestionar claves RSA para no escribir claves al conectar a los dispositivos y pasar los datos de cualquier otra aplicación por un canal seguro tunelizado mediante SSH.

3.4.1. Seguridad

SSH trabaja de forma similar a como se hace con telnet. La diferencia principal es que SSH usa técnicas de cifrado que hacen que la información que viaja por el medio de comunicación vaya de manera no legible, evitando que terceras personas puedan descubrir el usuario y contraseña de la conexión ni lo que se escribe durante toda la sesión; aunque es posible atacar este tipo de sistemas por medio de ataques de REPLAY y manipular así la información entre destinos.

3.4.2. Instalación de OpenSSH

OpenSSH es una herramienta libre de la conectividad SSH y para instarlo y ejecutarlo se necesita abrir un terminal y escribir los siguientes comandos:

$ sudo apt-get install openssh-server

con esto instalamos y ya que termine de instalar debemos iniciarlo.

$ sudo etc/initd./ssh start

Capítulo 4

Desarrollo e implementación de algoritmo autónomo.

En este capitulo se describen tipos de control para llegar a tener un control autónomo con el VANT pero para poder lograr una interfaz entre la computadora y el VANT se necesita un software que nos permite la recepción y envío de datos basada en Linux y lograr la navegación autónoma así como su su respectivo control de un punto de posición en un espacio tridimensional, y que a la vez el software trabaje con código abierto, conforme a lo investigado y visto en los capítulos anteriores fue que se aplico ROS, y es el entorno en el qué se ha finalizado el proyecto así como también se hará uso de un lenguaje de programación conocido como Python, en el cual nos permite crear nodos para la investigación autónoma y control del VANT.

Se utilizaron varios paquetes, el principal se trata del paquete mavros, que es el encargado del envío y recepción de mensajes del VANT con el ordenador en este caso una PC en el cual nosotros podemos observar todos los datos de navegación en tiempo real, de igual forma se tiene el paquete rospy que es el encargado de comunicar el programa con los nodos de ROS. Con esto podemos correr 2 programas a la vez y que estos se estén comunicando entre si.

4.1. Caracterización de los motores

Para el modelo y para saber cuanta fuerza tiene nuestro drone fue necesario caracterizar los motores para saber que torque, empuje y revoluciones por minuto tiene por lo que se realizo un programa en arduino para ir aumentando de manera controlada y gradual el PWM(modulación por ancho de pulso), y con ayuda de un sistema poder medir las RPM(revoluciones por minuto) y el empuje. dicho programa fue el siguiente:

```
 #include <Servo.h>
Servo myservo;
int val;
int state;
int bits = 63;
```

```
volatile int conteo = 0;
int rpm=0;
unsigned long lastmillis = 0;
void setup()  {
myservo.attach(9);
pinMode(2,INPUT);
pinMode(13,OUTPUT);
digitalWrite(2,LOW);
Serial.begin(9600);
attachInterrupt(0,pinISR,RISING); }
void loop()  {
myservo.write(bits);
if (millis() - lastmillis >= 1000)
{

    detachInterrupt(0);
    rpm = conteo * 30;
    Serial.print("RPM =\t");
    Serial.print(rpm);
    Serial.print("\t Hz=\t");
    Serial.print(conteo);
    Serial.print("\t Bits=\t");
    Serial.println(bits);
    conteo=0;
    lastmillis = millis();
    attachInterrupt(0, pinISR, RISING);

}
//Incrementar pwm por teclado c1+ c2-
if (Serial.available() > 0)
{

    if (Serial.peek() == 'c')
    {
        Serial.read();
        state = Serial.parseInt();
        if (state==1)
        {
           bits=bits+3;
        }
        if(state==2)
        {
           bits=bits-3;
        }
```

```
          if(state==3) {bits=63;}
        digitalWrite(13,state);
      }
      while (Serial.available() > 0)
      {        Serial.read();     }
      }
}
//interrupcion IR void pinISR()
{    digitalWrite(13,!digitalRead(13));
       conteo++;
}
```

Con este código pudimos obtener los siguientes resultados.

Cuadro 4.1: Datos para calcular RPM

BITS	RPM	GRAMOS
141	1800	15
146	1830	29
151	2280	44
156	2670	65
161	3030	82
166	3330	100
171	3690	119
176	4050	145
181	4350	178
186	4710	209
191	5070	238
196	5400	276
201	5790	318
206	6090	352
211	6420	390
216	6750	434
221	7020	468
226	7320	502
231	7590	536
236	7860	570
241	7860	570

Y Sobre esa información resulto las gráficas de las figura 4.1 y 4.2

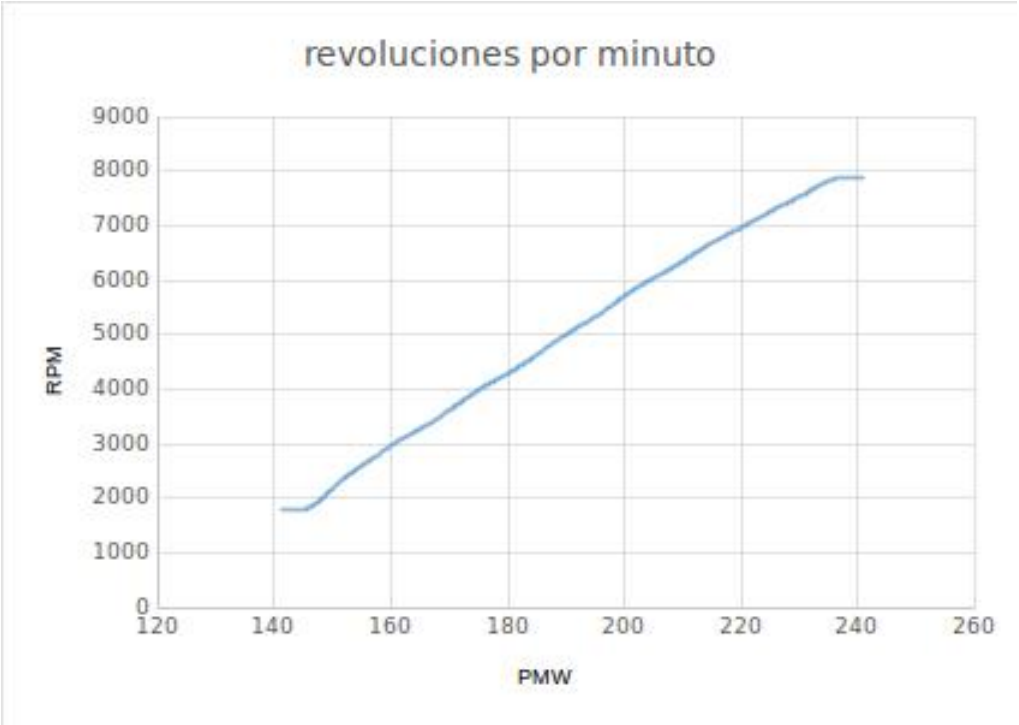

Figura 4.1: gráfica de RPM

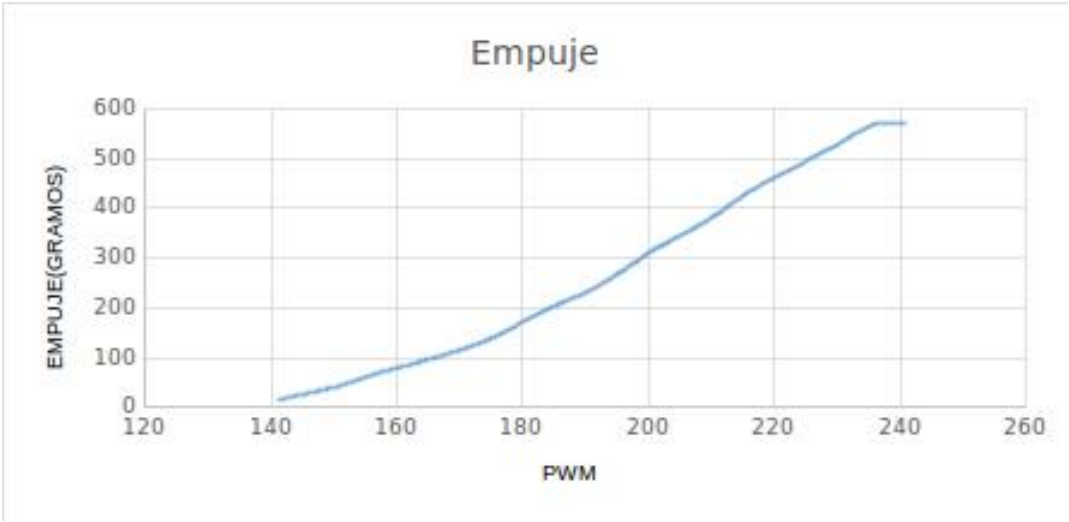

Figura 4.2: Trafica de empuje

Cada motor nos ofrece un empuje de 570 gramos, esto multiplicado por los 4 motores nos da una fuerza total de 2280 gramos, que en teoría es lo máximo que nuestro drone puede levantar.

4.2. Conectar el erle-brain a la computadora

Antes de comenzar tenemos que saber como manipular el Erle-copter, como vimos en capítulos anteriores, este cuenta con un cerebro llamado erle-brain, este cerebro es un sistema embebido por lo que para accesar tenemos que utilizar el protocolo SSH, para ello se conecta el cable usb a la computadora y al erle-brain, abrimos un terminal de ubuntu y para ver las interfases disponibles escribimos.

$ ipconfig

La que nos interesa es la USB0 con ella va a ser que nos vamos a comunicar con el Erle-copter. le asignamos una IP con.

$ sudo ifconfig usb0 192.168.7.1

NOTA: Si la conexión entre la computadora y el erle-brain se pierde se tiene que volver asignar la IP a la interfaz usb0, esto suele pasar con frecuencia.

Con el protocolo SSH entraremos al sistema del erle-brain, para ello se inicia el protocolo y luego se coloca la IP del erle-brain para accesar, el usuario es ubuntu y la contraseña es ubuntu, para ello escribimos.

$ sudo /etc/init.d/ssh start

$ ssh ubuntu@192.168.7.2

$ubuntu

Para saber que todo se realizo correctamente, nos apareció lo de la figura 4.1.

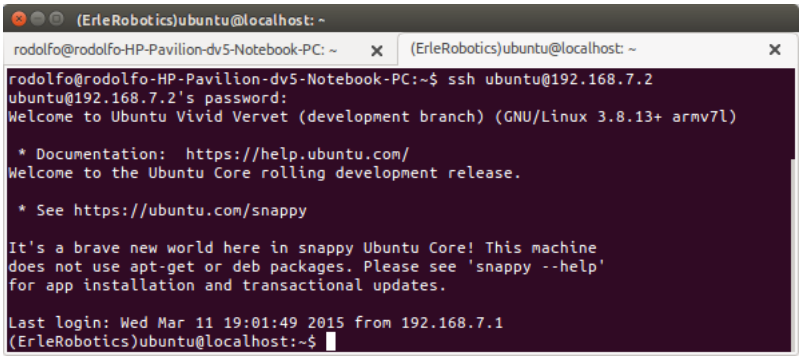

Figura 4.3: Interfaz erle-brain

Si nos aparece ese mensaje ya podemos manipular el erle-brain.

4.3. Modificar frecuencia del WI-FI y conectar

El erle-brain maneja una conexión WI-FI en el cual nos podemos conectar pero la frecuencia que maneja es de 5Ghz y aun son pocos los dispositivos que manejan esa frecuencia, por lo que tenemos la opción de bajar la frecuencia a una mas usual como es la de 2.4GHz, para eso, ya que estuvimos dentro del cerebro escribimos.

(ErleRobotics)ubuntu@localhost:~$sudo vi /etc/hostapd/hostapd.conf

Esto nos abrirá el editor vi, cabe recalcar que una vez que estamos manipulando el celebro del Erle-copter solo nos permite utilizar este editor,ahora en el archivo que nos abrió, se modificó los siguientes parámetros:

channel=40 -> channel=9

#hw_mode=g -> hw_mode=g

hw_mode=a -> #hw_mode=a

Y luego reseteamos el erle-brain, con esto pudimos bajar la frecuencia, y ahora cualquier dispositivo pudo encontrar el WI-FI.

Para conectarnos al WI-FI buscamos la red llamada erle_Snappy, la contraseña es holaerle, ahora para conectarnos hicimos los mismos pasos que con el cable, solo que cambiamos la IP.

$ ssh ubuntu@10.0.0.1

$ubuntu

Y nos apareció lo de la figura 4.1.

4.4. Acceso a la telemetría

Como mencionamos anteriormente debido a las características del erle-brain, nosotros pudimos accesar a todos los parámetros disponibles de los sensores, todo esto dentro del sistema operativo del Erle-copter, con ayuda de ROS, para dicho paso fue necesario ejecutar los siguientes comandos en una terminal de nuestro ordenador.

$ sudo etc/init.d/ssh start

$ ssh ubuntu@192.168.7.2

$ cd trusty

$ sudo sh chroor.sh

$ cd root

$ source /opt/ros/indigo/setup.bash

$ export | grep ROS

Para saber que parámetros teníamos disponibles utilizamos

$ rostopic list

Después de desplegarnos toda la lista de parámetros pudimos ejecutar cualquiera de ellos con:

$ rostopic echo “parámetro”

Por ejemplo.

$ rostopic echo /diagnostics

Este comando como tal nos muestra un diagnósticos de como funciona el cerebro del Erle-copter.

4.5. ROS en en Erle-brain

Como vimos en capítulos anteriores el erle-brain es una computadora embebida con un sistema operativo cargado llamado Snappy ubuntu core y sobre ese sistema operativo trae un meta-sistema operativo especial para robots como lo es ROS, eso quiere decir que tenemos disponible todas las virtudes de este meta-sistema y como prueba de ello se compilo el programa hecho anteriormente después de instalar el ROS en nuestro ordenador, para ello es fue necesario escribir en una terminal de ubuntu los siguientes comandos.

$ roscore

$ “ctrl”+ “shift” + “t”

$ touch publicar.py suscribir.py

$ gedit publicar.py

Al hacer esto abrirá el editor de textos y dentro copiaremos el siguiente programa en python.

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import String
def talker():

    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():

        hello_str = "hello world%s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':

    try:

        talker()

    except rospy.ROSInterruptException:

        pass
```

Luego de haber copiado el programa lo guardaremos y regresamos a la consola, ahora haremos lo mismo con el programa suscribir.py, por lo que escribiremos

$ gedit suscribir.py

y copiaremos lo siguiente:

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import String
def callback(data):

    rospy.loginfo(rospy.get_caller_id() + "I heard%s", data.data)

def listener():

    rospy.init_node('listener', anonymous=True)

    rospy.Subscriber("chatter", String, callback)
    # "spin()" simplemente mantiene python hasta que el nodo se detiene

    rospy.spin()

if __name__ == '__main__':

    listener()
```

guardamos el programa y regresamos a la terminal.

ya una vez en la terminal para ejecutar los programas solo escribimos.

$ python suscribir.py

$ python publicar.py

Y al igual que se ejecutó en nuestro ordenador, también lo realizo dentro del sistema operativo del VANT.Figura4.2.

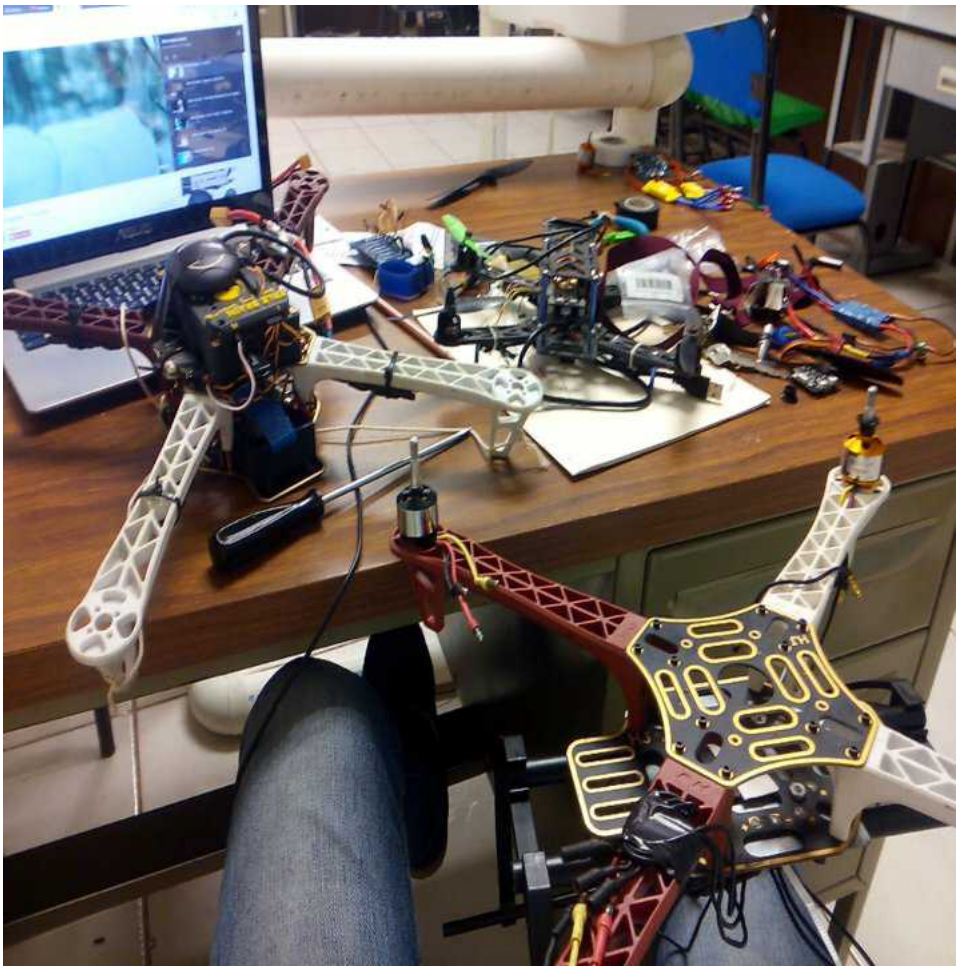

Figura 4.4: Erle-brain

4.6. Despegue y aterrizaje

Una vez sintonizado el PID y estabilizado el sistema se continuo con el vuelo autónomo y como primeros pasos se hizo un programa en C++ para que el drone despegara, se elevara a 10 metros y luego de esto se mantuviera 20 segundos para luego aterrizar, mas adelante veremos porque usamos Python y no C++;

Antes que nada como vimos en capitulo anteriores el celebro del erle es una computadora embebida, por lo tanto tenemos un sistema operativo Snappy Ubuntu Core y sobre eso tiene ejecutando el ROS, por lo tanto primeramente tenemos que entrar dentro del cerebro para poder modificar, crear y ejecutar nuestros programas, para ello conectamos el erle-brain a la computadora con los pasos anteriores.

Ya dentro del drone ejecutamos los comandos:

$ cd trusty

$ sudo sh chroot.sh

$ cd root

$ source /opt/ros/indigo/setup.bash

$export | grep ROS

$ mkdir catkin_ws

$ mkdir src

$ catkin_init_workspace

$ mkdir ros_erle_takeoff_land

$ touch CMakeLists.txt package.xml

$ mkdir src

$ main.cpp

Con esos comandos preparamos el drone para ejecutar el programa, se hicieron varias pruebas, y este fue un programa básico para solo elevar el drone 20 metros se mantenía 20 segundos y luego aterrizaba y el programa es el siguiente.

```
#include <cstdlib>
#include <ros/ros.h>
#include <mavros/CommandBool.h>
#include <mavros/CommandTOL.h>
#include <mavros/SetMode.h>
int main(int argc, char **argv)
{
    int rate = 10;

    ros::init(argc, argv, "mavros_takeoff");
    ros::NodeHandle n;

    ros::Rate r(rate);
 ////////////////////////////////////////
////////////////GUIDED//////////////////
////////////////////////////////////////
ros::ServiceClient cl = n.serviceClient<mavros::SetMode>("/mavros/set_mode");
mavros::SetMode srv_setMode;
srv_setMode.request.base_mode = 0;
srv_setMode.request.custom_mode = "GUIDED";
if(cl.call(srv_setMode))
 {
        ROS_ERROR("setmode send ok%d value:", srv_setMode.response.success);
}
else
{
       ROS_ERROR("Failed SetMode");
```

```
       return -1;
}
 ////////////////////////////////////////
//////////////////ARM/////////////////////
////////////////////////////////////////
ros::ServiceClient arming_cl = n.serviceClient<mavros::CommandBool>("/mavros/cmd/arming");
mavros::CommandBool srv;
srv.request.value = true;
if(arming_cl.call(srv))
{
        ROS_ERROR("ARM send ok%d",srv.response.success);
}
else
{
        ROS_ERROR("Failed arming or disarming");
}
   ////////////////////////////////////////
  /////////////////TAKEOFF////////////////////
 ////////////////////////////////////////
ros::ServiceClient takeoff_cl = n.serviceClient<mavros::CommandTOL>("/mavros/cmd/takeoff");
mavros::CommandTOL srv_takeoff;
srv_takeoff.request.altitude = 10;
srv_takeoff.request.latitude = 0;
srv_takeoff.request.longitude = 0;
srv_takeoff.request.min_pitch = 0;
srv_takeoff.request.yaw = 0;
if(takeoff_cl.call(srv_takeoff))
{
    ROS_ERROR("srv_takeoff send ok%d", srv_takeoff.response.success);
}
else
{
        ROS_ERROR("Failed Takeoff");
}
 ////////////////////////////////////////
////////////////DO STUFF//////////////////
////////////////////////////////////////
sleep(10);
  ////////////////////////////////////////
  /////////////////LAND////////////////////
  ////////////////////////////////////////
 ros::ServiceClient land_cl = n.serviceClient<mavros::CommandTOL>("/mavros/cmd/land");
```

```
mavros::CommandTOL srv_land;
srv_land.request.altitude = 10;
srv_land.request.latitude = 0;
srv_land.request.longitude = 0;
srv_land.request.min_pitch = 0;
srv_land.request.yaw = 0;
if(land_cl.call(srv_land))
{
    ROS_INFO("srv_land send ok%d", srv_land.response.success);
 }
else
{
        ROS_ERROR("Failed Land");
}
```

Se compilo el programa con los siguientes comandos, en la carpeta de src de catkin_ws.

$ catkin_make –pkg ros_erle_takeoff_land

$ cd devel

$ source setup.bash

$cd ../..

$ rosrun ros_erle_takeoff_land ros_erle_takeoff_land

Primeramente se hicieron pruebas en el laboratorio, en un cubo especial para probar drones, ahí se sujeto y se ejecutó con ROS, al comenzar a funcionar los motores observamos su comportamiento para luego probarlo al aire libre. Notamos que al ejecutar el programa, el drone tenia que estar alineado correctamente ya que los sensores se re-calibran y tomaban esa posición como su referencia.

Ya que se había hecho varias pruebas en el laboratorio y luego de observar un comportamiento estable se continuo probándolo en espacio abierto y como seguridad se amarro un lazo para evitar perder el control totalmente, luego de elevarse notamos que había cierta desviación debido al viento pero se pudo corregir con el GPS, por lo que el resultado fue satisfactorio. Fig. 4.2.

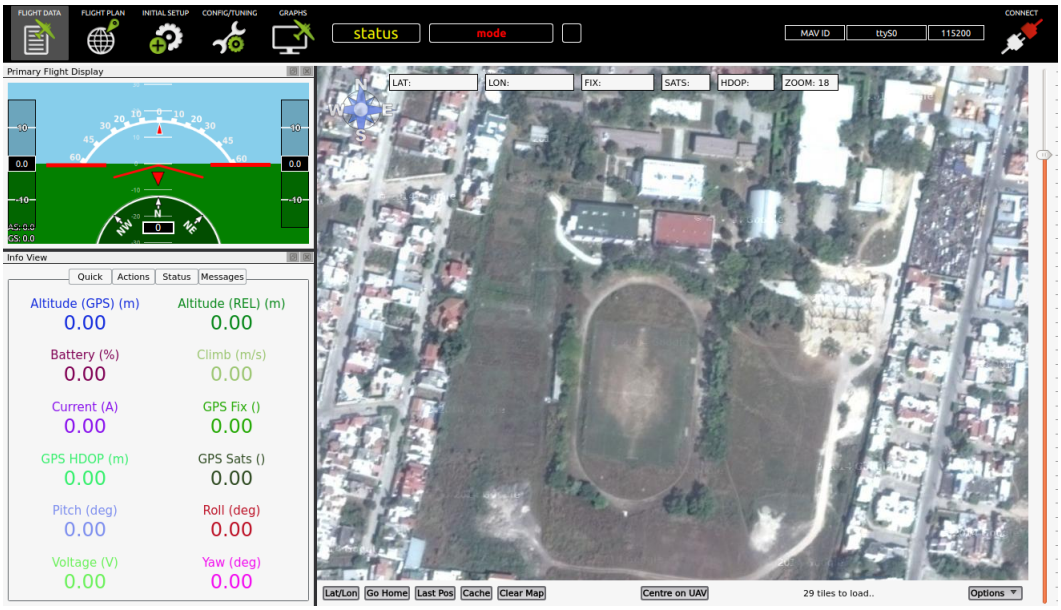

Figura 4.6: APM PLANNER

Figura 4.5: Erle-copter autónomo

4.7. Set point y trayectoria

El beaglebone black y el pixhawk es compatible con autopilot y este tiene a su vez un software llamado APM PLANNER dicho programa tiene la capacidad de leer y graficar los valores que el erle-brain esta obteniendo con los sensores y estos valores son mandados a este programa para ser interpretados a través de su interfaz (Fig. 4.3), con ayuda del GPS nosotros fuimos capaces de establecer rutas y dichas rutas poder ser ejecutadas por el Erle-copter,

En el mapa se establecían los puntos o las coordenadas para el erle, se establecía el punto de partida, el punto final, lo que corresponde al takeoff y al landing (despegue y aterrizaje) y la altura. Luego de

esto se inicio con el control, dándole cierta potencia a los motores, al elevar, con el APM PLANNER se le cambiaba el modo de vuelo, de estabilize a auto, al hacer el cambio comenzaba a realizar las trayectorias establecidas en el mapa, en ella misma trazaba una trayectoria de la posición actual del VANT y luego de ejecutar los puntos establecidos comenzaba a descender, las pruebas que se hicieron tuvieron un margen de error de aproximadamente 30 cm. con respecto al punto de inicio al punto final.

Capítulo 5

Resultados

En este capitulo se analizara los resultados que se obtuvieron durante la realización del proyecto, y se determinará si cumplió con los objetivos propuestos al inicio de este trabajo y cuales fueron los puntos clave a resaltar.

5.1. logaritmo autónomo aplicado al Erle-Copter

Luego de haber ejecutado el logaritmo desarrollado en C++ y luego en Python dentro del VANT y de observar los resultados dentro y fuera del laboratorio se llevo a la conclusión de que fue lo que se esperaba, y fue un avance importante que servirá como punto de partida para ir encontrando y desarrollando algoritmos mas complejos para que pueda realizar más acciones a la vez, así como se pudo observar que realizar un código en Python resulta mas simple de ejecutar para el VANT que uno hecho en C++ ya que la eficiencia del código y la buena distribución que caracteriza a la programación en Python facilita la ejecución y simplifica el código. También se observó que el meta-sistema operativo ROS resulta una herramienta importante, ya que la ideología de publicador / suscriptor simplifica el código, ya que al tener un código muy extenso, este se puede dividir en 2 y estar comunicados entre si a través de los nodos de ROS.

Asi mismo con todo este trabajo se pudo aprovechar otras herramientas necesarios para la elaboración del proyecto, una de ellas fue el protocolo de comunicación entre maquinas remota llamado SSH el cual nos ayudó a accesar al erle-brain como si ese fuera nuestro ordenador nativo y a través de ahí se enviaba y ejecutaba los comandos para el VANT.Figura 5.1.

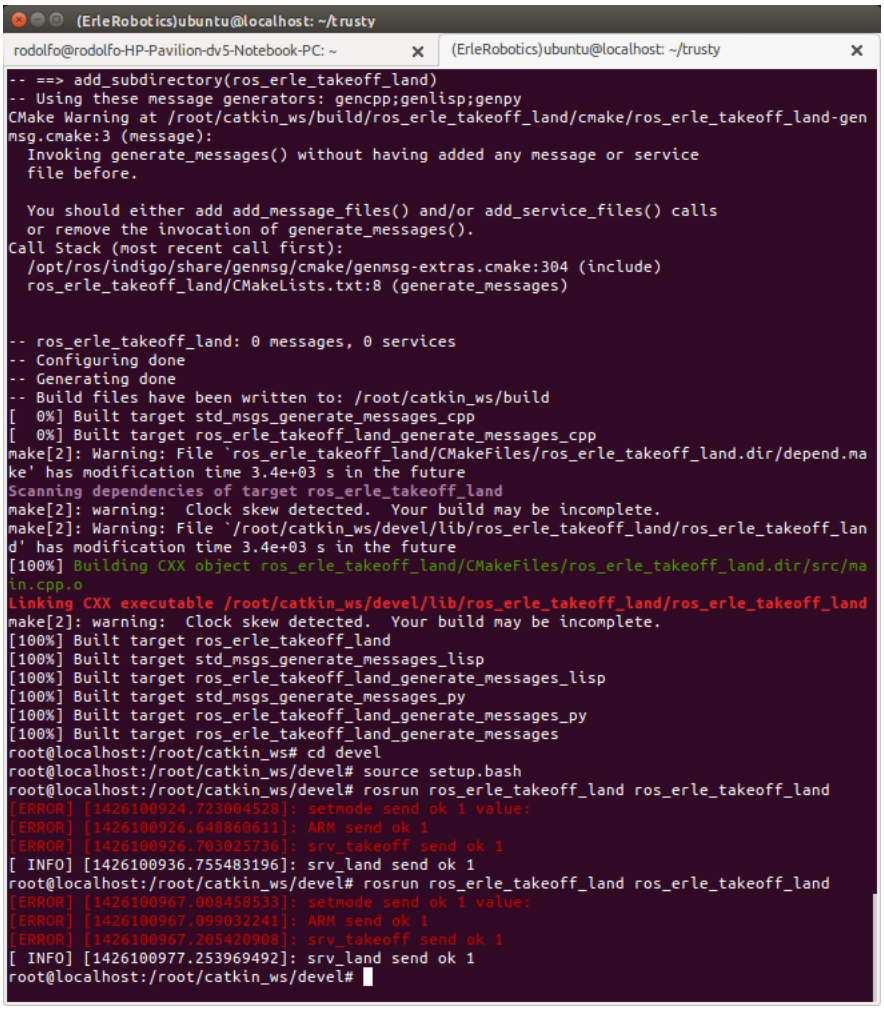

Figura 5.1: Ejecución de un programa en el VANT

De igual forma el GPS tomó un valor importante al encontrar la forma de establecer coordenadas a través de una interfaz como lo hizo el software APM PLANNER 2.0 ya que con este programa se encontró otra forma de manipular el VANT de manera autónoma y de una forma mas sencilla a la que fue por código, y con una precisión muy aceptable, mejor a la que tiene cualquier GPS convencional, debido a su precisión lo hace apto para otra forma de control autónomo por seguimiento de trayectoria.

Capítulo 6

Conclusión y trabajos futuros

Se concluyó que hacer un buen control de un vehículo aéreo nos deja una gran experiencia, demostrando la capacidad del control y de vuelvo autónomo de nuestro Erle-copter de cuatro motores, se encuentra disponible de forma absoluta la investigación en estos pequeños aparatos, logramos hacer un buen control de posición, cabe mencionar que los resultados obtenidos son de gran eficiencia y de gran relevancia para la educación, y de un gran impacto tecnológico, visto el nivel de reacción y la estabilidad gracias a su configuración de cuatro motores, se pueden sugerir una multitud de campos de desarrollo.

En el desarrollo del proyecto se ha logrado el vuelo autónomo y la manera de publicar y recibir información referida a la navegación o a la telemetría del cuadricóptero. Se obtuvo un punto de inflexión, ya que al conocer el funcionamiento del Erle-Copter y de ROS con el Rospy y los diferentes métodos de configuración, obtenemos multitud de vías de investigación.

Una vez resuelto el principal objetivo del proyecto, de dotar al Erle-Copter de un software capaz de realizar un vuelo autónomo, se ha investigado en los diferentes caminos de ampliación.

La principal ampliación se centra en la navegación, podríamos dotar al VANT de una navegación autónoma siguiendo una línea recta, al traer integrada una cámara vertical podemos observar el suelo, y con ella la línea guía para realizar ese vuelo. Además, descubiertas las ventajas de este pequeño vehículo no tripulado se puede adaptar una cámara, en el cual podemos realizar mapeado 3D a través de una única cámara, podemos llegar a conocer las coordenadas de posición del VANT en el momento de la toma de imagen. Cercana a la aplicación obtenida en este proyecto podemos realizar una detección de caras, que puede ser muy útil en tareas de vigilancia o desastres naturales. Sería posible encontrar personas en zonas de difícil acceso, o a modo de vigilante para el reconocimiento de intrusos. Los cuadricópteros son vehículos aéreos de corta historia, pero con una curva de alto desarrollo en estos momentos. Son innumerables los modelos utilizados en el ámbito militar, mayoritariamente como elemento de vigilancia, pero la mayoría no disponen de vuelo autónomo. Dotando a estos aparatos de un pilotaje autónomo se podrían ganar en seguridad, y en rapidez de reacción, por ejemplo, en la búsqueda de artefactos como minas, e incluso, aparatos autónomos suficientemente potentes podrían rescatar a personas una vez encontradas sin esperar a la llegada de un equipo humano, el muestreo de zonas terrestres de alto riesgo etc. No se puede estimar la multitud de campos donde estos aparatos pueden avanzar sin embargo el desarrollo de los cuadricópteros es una apuesta segura para el avance en numerosos campos, dando máxima importancia a su característica principal de estabilidad, y posibilidad de reconocimiento del entorno gracias a sensores, entre ellos, las

cámaras de alta definición.

Y las aplicaciones no paran ahí se puede equipar al VANT con tubos al vacío con el objetivo de medir la calidad del aire a diferentes alturas.

Bibliografía

[1] Master's thesis, INSTITUTO POLITECNICO NACIONAL, 2010.

[2] colaboradores de Wikipedia. https://es.wikipedia.org/w/index.php?title=6 de agosto del 2015, 16:42 UTC.

[3] A. B. I. Gonzalo Ferrer Minguez. integracion kalman de sensores inerciales ins con gps en un uav. Master's thesis, 2009.

[4] ROS instalacion. http://wiki.ros.org/ros/installation.

[5] Reglamento Mexicano. http://www.aeropuertodrone.com/reglamento.htm.

[6] Katsuhiko Ogata. *Ingenieria de control moderna*. 2003.

[7] Katsuhiko Ogata and Yanjuan Yang. Modern control engineering. 1970.

[8] Erle robotics. http://erlerobotics.com/blog/home-creative/.

[9] ROS. http://www.ros.org/.

[10] Oscar Vila Rovira. Modelizacion de aeronaves no tripuladas con simulink. Master's thesis, escuela universitaria de ingenieria tecnica aeronautica, 2011.