Universidad de Sonora
División de Ciencias Exactas y Naturales
Departamento de investigación de física
Maestría en ciencias de la electrónica

“Control externo de cuadri-rotor mediante un sistema de visión de cámaras Optitrack”

Tesis que para obtener el título de
Maestro en Ciencias en electrónica

Presenta:
Mario Guadalupe Orocio Rubio

Director de tesis:
Luis Arturo García Delgado
Fecha

Universidad de Sonora

Repositorio Institucional UNISON

Excepto si se señala otra cosa, la licencia del ítem se describe como openAccess

Agradecimientos

A Dios por lar la salud y por todo lo que me a dado en la vida.

A mis padres y a mi hermana por todo el apoyo que me han dado, gracias a ustedes He podido alcanzar todas mi metas debido a la formación, valores y responsabilidades que me han enseñado y sobretodo en los momentos difíciles siempre han estado conmigo.

A todos mis compañeros de maestría por todas las enseñanzas, conocimientos, desveladas, momentos y sobretodo la amistad durante estos dos años y medio en la maestría en especial.

A mi ayuda y apoyo durante el desarrollo de este proyecto de la maestría.
A la CONACYT por el apoyo económico de la beca.

Resumen

En esta tesis se propone el control externo del cuadrirrotor 3DR Iris+ mediante un sistema de seis cámaras Optitrack, la utilización de una computadora a bordo del cuadrirrotor permite controlar externamente cualquier unidad de control de vuelo que tenga el software PX4 Autopilot. En el sistema de la OBC se utiliza ROS permitiendo el uso de varios sensores y actuadores que son compatibles con ROS.
El FCU utilizado por el 3DR Iris+ es el *Pixhawk*, La ventaja de esta unidad reside en que al ser un proyecto en código abierto permite modficar el firmware y adaptarlo como el desarrollador desee. El protocolo de comunicación que el Pixhawk utiliza con la estación de tierra QGC es llamada MAVLink, un protocolo que permite modificar cualquier parámetro del firmware PX4.
El prótocolo MAVLink adaptado en ROS es conocido como MAVROS, este paquete nos permite controlar sensores y actuadores, ademas de que se permite recibir y enviar datos, información o parámetros a la unidad Pixhawk como los que se muestran abajo:

- Puntos de ajuste de posición.
- Puntos de ajuste de velocidad.
- Puntos de ajuste de aceleración.
- Puntos de ajuste angulares.

Utilizando ROS y MAVROS se transmitieron los datos de visión de las cámaras Optitrack al Pixhawk, para los resultados se simulo en plataformas de ROS y se realizaron diferentes trayectorias al 3DR iris para corroborar que la transmisión se hizo correctamente.

Contenido

1 Introducción **1**
1.1 Objetivos de la tesis 1
1.1.1 Objetivo principal 1
1.1.2 Objetivos específicos 1
1.2 Antecedentes 2
1.3 Descripción del cuadrirrotor 2
1.4 Iris+ Hardware 3
1.5 Sistema de visión Optitrack 4
1.6 Esquema de tesis 5

2 Estado del arte **7**
2.1 Pixhawk autopilot 7
2.1.1 Arquitectura general del software Pixhawk Autopilot 8
2.1.2 PX4 Firmware 11
2.1.3 Módulos del PX4 firmware 14
2.2 ROS 21
2.2.1 Espacio de trabajos y paquetes ROS 23
2.2.2 Nodo maestro y nodos ROS 24
2.2.3 Tópicos y mensajes 28

3 Desarrollo Experimental 33
3.1 Modo no abordo y configuración del Pixhawk 33
3.1.1 Configuración del hardware de Pixhawk 33
3.1.2 Sintonización del estimador LPE para MOCAP o visión 36
3.1.3 Control no abordo 38
3.1.4 Configuración del hardware del control no abordo 39
3.2 MAVROS 41
3.2.1 MAVLink 42
3.2.2 nodo y plug-ins MAVROS 44
3.3 Transmisión de datos de sistema Optitrack a Pixhawk 47
3.3.1 Esquema de transmisión de datos Optitrack a Pixhawk 47
3.3.2 Configuración Motive Software y nodo vrpn_client_ros 48
3.3.3 Nodo MAVROS y transformación ENU→NED 49
3.4 Nodo de generación de trayectorias 52

4 Resultados experimentales 55
4.1 Resultados Experimentales 55
4.1.1 Prueba de altitud 55
4.1.2 Prueba de trayectoria circular 57
4.1.3 Prueba de trayectoria en forma de 8 58

5 Conclusión 61
5.1 Trabajo a futuro 62

Bibliografía 63

Palabras clave

- **FCU:** Unidad de control de vuelo
- **OBC:** Computadora a bordo.
- **QGCS:** Estación de tierra.
- **ROS:** Robot Operating System
- **MAVLink:** Micro Air Vehicle Communication Protocol
- **MAVROS:** Extensión de protocolo MAVLink-ROS
- **Pixhawk:** módulo de vuelo.
- **Motive Software:** Programa de las cámaras empresa Optitrack.
- **UAV:** Vehículo Aéreo no Tripulado
- **FPS:** Cuadros Por Segundo.

1 Introducción

En este capítulo se presentan los objetivos de la tesis así como los conceptos básicos del uso del cuadrirrotor y de las cámaras Optitrack.

1.1. Objetivos de la tesis

1.1.1. Objetivo principal

Implementar un sistema de posicionamiento del dron basado en un sistema de visión y un control externo para un cuadrirrotor.

1.1.2. Objetivos específicos

- Estudiar la arquitectura del FCU Pixhawk.
- Modificar librerías y módulos del Pixhawk.
- Estudiar e implementar ROS y MAVROS.

1.2. Antecedentes

Un cuadrirrotor es un UAV que tiene dos juegos de hélices idénticas alimentadas por motores de DC brushless que proporcionan una fuerza de empuje y que es capaz de realizar maniobras cuando está en el vuelo. Debido a su estabilidad y características, los cuadrirrotores tienen una mejor maniobrabilidad de vuelo que los aviones de ala fija.

En la actualidad, el auge de los UAV se ha expandido a 3 sectores de la población: el primero es para el uso militar, donde son utilizados para misiones de inteligencia, vigilancia y de reconocimiento; el segundo sector es en lo comercial, donde los cuadrirrotores son equipados con pequeñas o grandes cámaras para el área de supervisión de edificios y las áreas de fotografía y cine; por último, es el sector de investigación, debido a que se pueden manejar y operar los cuadrirrotores en interiores. En los últimos años se han ido creando diversos tipos de cuadrirrotores para el área de investigación, como el AR Drone Parrot, 3DR Iris+, etc. debido a que su software es de código abierto, por lo que el lenguaje de programación es de bajo nivel y se puede modificar la programación funciones básicas de vuelo o modificar el código embebido del hardware para realizar controladores de vuelo complejos.

1.3. Descripción del cuadrirrotor

La estructura del cuadrirrotor generalmente consiste en cuatro rotores, donde cada par de rotores está colocado a la misma distancia de su centro de masa. Un par de rotores giran en el sentido de las manecillas del reloj y el otro par en sentido inverso (figura **1-1**). La rotación en sentido opuesto es para contrarrestar el torque giroscópico. El cuadrirrotor realiza las maniobras de vuelo enviando señales PWM a los motores brushless para variar la velocidad de rotación de las hélices.

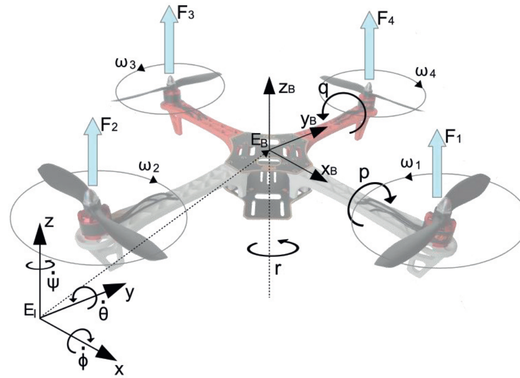

Figura 1-1: Marcos y sentido de giro de los rotores de un cuadrirrotor.

Otra característica importante del funcionamiento consiste en que el cuadrirrotor se mueve en base a un marco inercial y aunque también se considera un marco unido a su estructura llamado marco del cuerpo. El cuadrirrotor es un UAV que tiene un movimiento de 6 grados de libertad producidos por un entrada de cuatro fuerzas que vienen de los rotores, por lo que se considera un vehículo subactuado.
En el diseño de un sistema de control se tienen en cuenta cuatro variables de control:

- **Traslación, x, y, z:** Es el desplazamiento lineal sobre cada uno de los tres ejes.
- **Aceleración, u:** Es el total de los empujes como resultado de la suma del empuje de cada motor.
- **Alabeo, ϕ:** Produce un giro sobre el eje x del cuerpo..
- **Cabeceo, θ:** Produce un giro sobre el eje y del cuerpo.
- **Guiñada, ψ:** Produce un giro sobre el eje z del cuerpo.

1.4. Iris+ Hardware

El iris+ (Figura **1-2**) es un cuadri-rotor construido por la empresa 3D Robotics (3DR), este vehículo aéreo fue diseñado para introducir a las personas al manejo de drones y para utilizarlo en el ámbito de la investigación. Lo acompaña una batería LiPo de 5100 mAh con 12v y cuenta con cuatro motores MN2213 950kV DC que dan un tiempo de vuelo de aproximadamente 22 min. La velocidad rotacional de los cuatro motores proporciona una

fuerza de empuje para que el iris+ pueda levantar una carga que no supere los 400 gr, ademas cuenta con un modulo GPS y otro de telemetría para establecer una comunicación inálambrica con la estación de tierra para recibir datos de vuelo en tiempo real.

Figura 1-2: Iris+ quadri-rotor [?, ?].

1.5. Sistema de visión Optitrack

Las cámaras Optitrack es un sistema óptico capturador de movimiento, teniendo como resultado la posición y orientación de un objeto en un espacio tridimensional. Esto se logra gracias a las cámaras infrarojas y los reflectores de luz infraroja que se encuentran en el objeto. Se recomienda tener un mínimo de cuatro marcadores reflectores para identificar un cuerpo.

El sistema Optitrack utiliza Motive Software [7], este programa es el que se dedica a crear el cuerpo rígido llamado "tracker"(Captura de movimiento y seguimiento de objetos 6 DOF), de igual manera este programa tiene la capacidad de crear varios cuerpos rígidos y obtener los datos de posición de cada uno, además de que nos permite modificar parámetros de la cámara como los FPS, la exposición, el límite de brillo mínimo, iluminación LED, etc.

Figura 1-3: Cámara Optitrack Flex 13

El laboratorio cuenta con un sistema de 6 cámaras Optitrack del tipo Flex 13 (ver Figura **1-3**). En la Tabla **2-2** se muestran las especificaciones del sistema de cámaras.

Tabla 1-1: Especificaciones de las cámaras FLEX 13

Resolución	1.3 MP(1280x1024)
Cuadros por segundo	120 FPS
Horizontal FOV	42,56
Interfaz	USB 2.0
Número de LEDs	28
Latencia	8.3 ms

1.6. Esquema de tesis

El esquema de esta tesis se resume de esta manera:

Capítulo 1, proporciona una breve introducción a la tecnología del cuadrirrotor, explica de manera general la descripción y las variables de control que intervienen en el cuadrirrotor y por último proporciona información del cuadrirrotor 3DR Iris+ y del sistema de visión, que son los equipos que se emplearon para el desarrollo de esta investigación.

Capítulo 2, presenta un análisis detallado de la unidad de control de vuelo Pixhawk, proporcionando información del software Autopilot, las diferencia de las pilas de vuelo PX4

y Ardupilot, un análisis extenso del firmware PX4 que contiene la tarjeta Pixhawk y por último una introducción a lo que es el lenguaje de programación ROS.

Capítulo 3, contiene las configuraciones de hardware y firmware, el desarrollo del procedimiento que se empleó para la transmisión de datos del sistema de cámaras Optitrack y el uso de MAVLink, junto con el paquete MAVROS.

Capítulo 4, presenta el análisis de los datos de vuelo registrados desde el 3DR Iris+ proporcionados por las pruebas de vuelo, mostrando el desempeño del cuadrirrotor con la recepción de datos del sistema de cámaras Optitrack y por último la conclusión para esta y las posibles áreas que se pueden estudiar para futuros trabajos de investigación.

2 Estado del arte

En este capitulo se presenta el equipo y el lenguaje de programaciòn que se empleo para enviar los datos de posición al 3dr iris.

2.1. Pixhawk autopilot

El Pixhawk autopilot ver figura (**2-1**) es diseñado por PX4 open-hardware project [2]. Este proyecto combina en un solo componente los módulos de unidad de gestión de vuelo (PX4FMU) y unidad de entrada/salida (PX4IO). Este componente tiene integrado un microprocesador M4 que ejecuta el sistema operativo Nuttx RTOS, el cual permite una programación en un entorno Linux. Gracias esto Nuttx permite a los desarrolladores programar, construir, cargar aplicaciones o firmware dentro del Pixhawk y, al ser un sistema operativo ligero y eficiente proporciona un entorno de programación del tipo POSIX (Portable Operating System Interface). En la Tabla **2-1** se presenta un resumen de las especificaciones del Pixhawk autopilot.Los módulos PX4FMU y PX4IO son:

- **PX4FMU:** La FMU es la unidad de manejo de vuelo, es la combinación de piloto automático y la IMU. Esta tarjeta se dedica a gestionar diferentes tipos de sensores o dispositivos conectados al bus de pines. Utiliza buses I2c, UART, CAN y pines de entrada/salida. Cuenta con cuatro salidas PWM para el manejo de motores y entrada GPS.

- **PX4IO:** La placa IO del PX4 se compone de una con una fuente de alimentación y un modulo de expansión para la unidad de manejo de vuelo PX4FMU. También posee

salida para servos, dos relevadores de estado solido, conectores de entrada/salidas y salidas e potencia de 5V con limitación de corriente.

Figura 2-1: PX4 Pixhawk Autopilot

Tabla 2-1: Especificaciones del Pixhawk

Procesador	Sensores	Interfaces
32bit STM32F427 Cortex M4	ST Micro 16 bit gyroscope	5x UART (puerto serial)
168 MHz	ST Micro 14 bit accelerometer/ magnetometer	2x CAN
256 KB RAM	MPU 6000 3-axis accelerometer/ gyroscope	spektrum DSM satellite
2 MB Flash	MEAS MS5611 barometer	Futaba Bus input/output
32 bit STM32F103 co-procesador		PPM sum signal input
		RSSI (PWM or Voltage)
		I2C
		SPI
		3.3v and 6.6v ADC inputs

2.1.1. Arquitectura general del software Pixhawk Autopilot

El Pixahwk tiene soporte de dos grandes familias de sofware de control, el primero de ellos es PX4 flight stack y el segundo es APM flight stack, ambos paquetes son de código abierto. La arquitectura de ambos se detalla a continuación:

1. **PX4 Flight stack**

   Este software [6] se puede dividir en dos principales bloques, el primero es el *PX4 flight control* que consiste en un conjunto de aplicaciones que componen los dos controladores de vuelo. Hay aplicaciones que son para cualquier tipo de UAV y otras aplicaciones que son para un determinado tipo de aeronave. El segundo llamado PX4 Middware es el encargado de proporcionar los drivers necesarios para sensores o dispositivos externos, así como una comunicación asíncrona entre aplicaciones independientes, en la figura **2-2** se muestra un diagrama más detallado de la arquitectura . Una breve explicación de cada bloque es:

   - **Aplicaciones:** El primer bloque es el que contiene las aplicaciones como el controlador de vuelo, estimador de estados o aplicaciones individuales e independientes *(apps)* que el mismo usuario realice. Generalmente las aplicaciones se deben empezar con la función *main()* y debe publicar o subscribirse a un tópico.

   - **uORB:** Esta es una estructura de datos. Consiste en un publicador y un subscriptor, donde el publicador espera compartir información mediante un tema (tópico), entonces con este tema puede subscribirse un subscriptor y así leer la información del tema, también se puede dar caso en el que un proceso puede ser un publicador y al mismo tiempo un subscriptor como se muestra en la figura **2-3**, gracias a este tipo de estructura de datos las aplicaciones pueden conectarse a diferentes procesos y drivers, haciendo que la comunicación entre éstos sea de manera eficiente y simple.

   - **Drivers y Hardware embebido:** El ultimo bloque es donde se encuentran los driver con código específico y los drivers que sirven como una interfaz para el sistema. Estos elementos mencionados forman parte del sistema operativo Nuttx.

   Por lo tanto, esto hace que los códigos generados por el Pixhawk sean altamente portables lo que permite que el Pixhawk autopilot pueda ser usado para una variedad de UAV (cuadri-rotores, vehículos de ala fija, vehículos terrestres no tripulados, etc.).

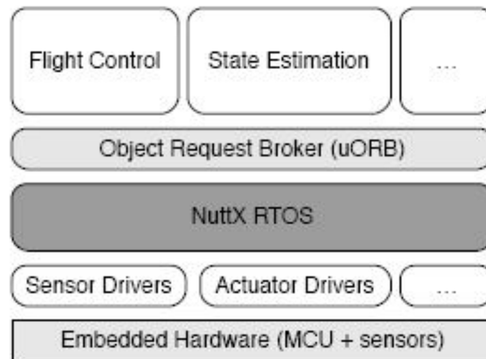

Figura 2-2: Partes de la estructura del sistema PX4 Flight stack

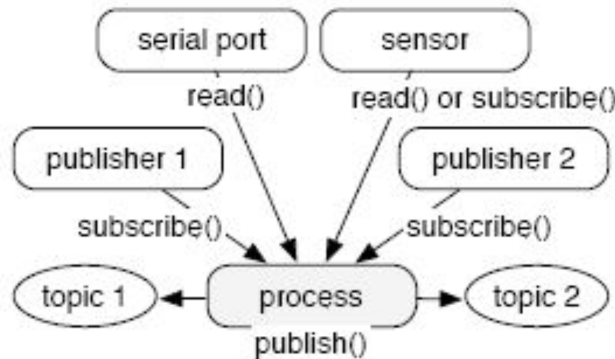

Figura 2-3: Proceso del sistema PX4 de publicador y subscriptor

2. **APM Flight stack**

El software APM Flight [10] es diseñado por *Ardupilot Autopilo*t y fue hecho como una aplicación única para la arquitectura PX4 Flight. Esta aplicación se ejecuta en las tarjetas PX4 Control (PX4FMU, Pixhawk, etc.), y se ejecuta a través del PX4 Middware (ver figura **2-4**) dentro del PX4 Framework, donde el APM se ejecuta para reemplazar al PX4 como el principal controlador de los drivers, haciendo que esta aplicación sea monolítica, es decir, una única aplicación en donde los niveles de interfaz de usuario y acceso de datos se combinan en un solo programa.

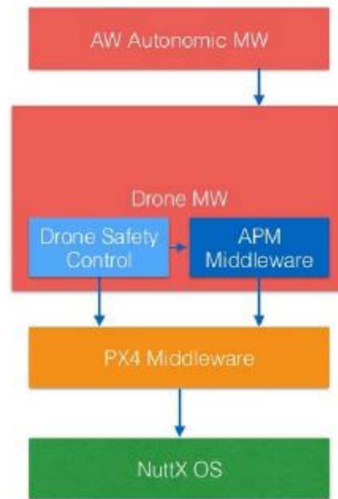

Figura 2-4: Arquitectura de la aplicación APM

2.1.2. PX4 Firmware

Como se mencionó anteriormente, el Pixhawk es la combinación del software de los módulos PX4FMU y PX4IO, teniendo en cuenta que el PX4FMU contiene la mayoría de las aplicaciones de vuelo como es el caso de los controladores de posición y de los ángulos. Por su parte, el PX4IO contiene la comunicación con los actuadores de los motores y los receptores de los mandos RC ubicados en la parte trasera del pixhawk exceptuando los pines de auxiliares que son parte del módulo PX4FMU.

Código fuente y organización de directorios

Después de instalar el firmware PX4 toolchain como en la figura **2-5** se muestra los tres principales directorio y la explicación de cada uno de estos:

- */cmake:* Archivos tipo make.
- */msg:* Son mensajes del tipo uORB, se encargan de publicar o subscribir.
- */ROMFS:* Este directorio contiene los scripts del proceso de arranque del sistema y la configuración de diferentes mixers.
- */src:* Es uno de los directorios más importantes del pixhawk, es el directorio o carpeta de trabajo donde se ubican todas las aplicaciones, drivers, controladores de vuelo, etc.

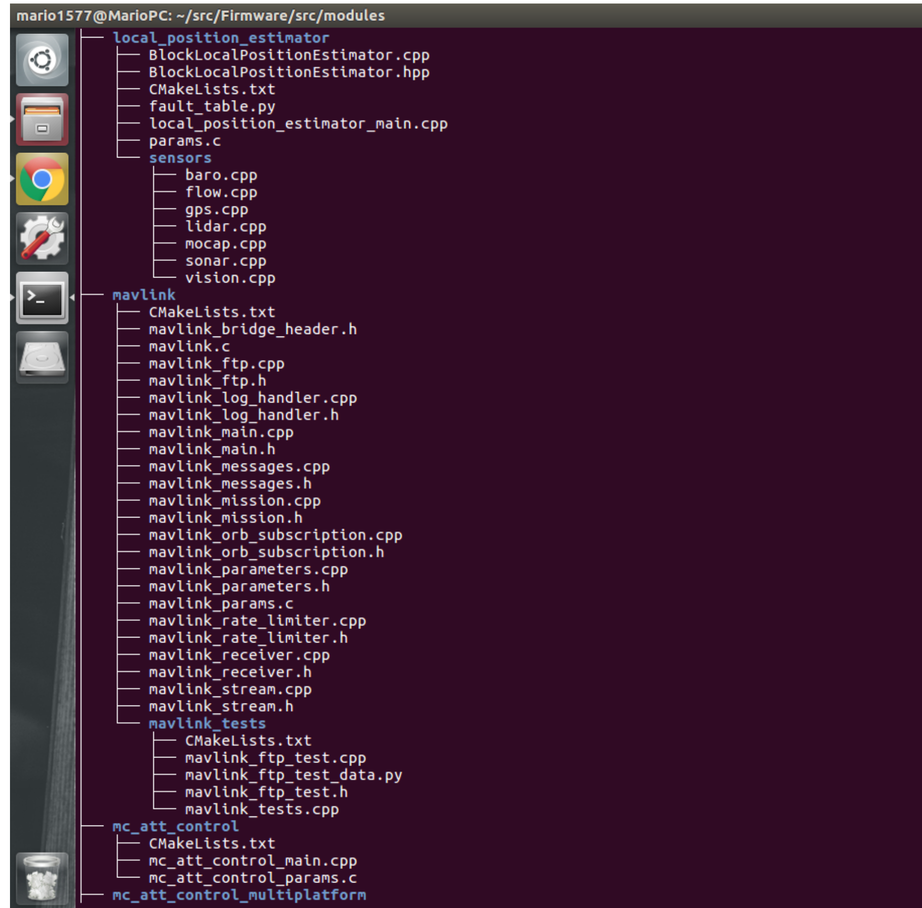

Figura 2-5: Diagrama de árbol de los directorios del firmware PX4.

- *src/drivers:* Este directorio contiene todos los drivers de los sensores interiores del pixhawk, este directorio se tiene que modificar si se desea agregar un sensor externo al pixhawk.
- *src/examples:* Contiene ejemplos simples para entender el código.
- *src/modules:* Este directorio contiene todos los estimadores o filtros de los sensores y los diferentes controladores de vuelo para aviones de ala fija, cuadrirrotores o VTOL.
- *src/systemcmds:* Comandos prácticos tipo *cmd* que pueden ser usados en Nuttx shell.

Arranque del sistema PX4

El bootloader es el primer programa del sistema en ejecutarse, la función de éste es en servir como gestor de arranque del sistema operativo Nuttx. Al ejecutarse el sistema operativo, Nuttx ejecuta el archivo script llamado *init.d/rcS*.
El script de arranque está ubicado en el directorio *ROMFS/px4fmu_common/init.d/rcS*, en éste se encuentran los comandos de arranque *uorb, mtd, param* ubicados en los archivos *uORBMain.cpp, mtd.c, param.c*; El comando uORB proporciona el servicio de comunicación, y carga el parámetro mtd_params que contiene las estructuras de los UAV.
El proceso de arranque en sistema PX4 se resume en lo siguiente:

1. Leer el archivo de parámetros.
2. Iniciar los drivers de los sensores internos y externos del Pixhawk (script *rc.sensors*).
3. Configurar y cargar el mixer correspondiente al parámetro *SYS_AUTOSTAR*T, configurar el canal pwm (script *rc.autostart*, este archivo es generado después de compilar el código).
4. Iniciar las tareas de vuelo correspondientes al parámetro airframe *SYS_AUTOSTART* (*script rc.fw_apps, rc.mc_apps, etc.*)

Arquitectura del PX4 Firmware:

Para el control de un UAV es necesario guiarse por puntos de referencia, estimar y controlar la posición y la orientación, en la (figura **2-6**).

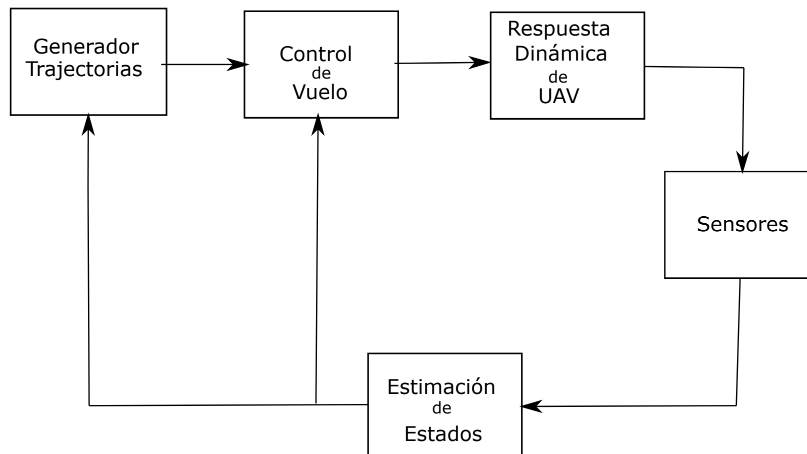

Figura 2-6: Arquitectura general del control de vuelo en el pixhawk

2.1.3. Módulos del PX4 firmware

Para representar de una manera más sencilla y sobre todo visual la arquitectura de los módulos del firmware, observe el diagrama de la figura **2-7**, donde cada cuadro representa una aplicación, mientras que las líneas que unen cada cuadro representan los tópicos, donde el sentido de cada flecha representa la dirección y la trayectoria del mensaje que sale del publicador tiene como destino el subscriptor. Dentro del diagrama los cuadros con linea continua representan las aplicaciones de la carpeta de trabajo (src) del firmware del PX4, mientras que el cuadro y flechas grises son los controladores de los de los sensores del módulo PX4FMU y por ultimo los cuadros con linea discontinua (—) representan los módulos encargados de los actuadores, en este caso el mixer, que pertenece al módulo PX4IO.
En general, el diagrama no representa toda la arquitectura del pixhawk, debido a que es esta es demasiado compleja por lo que no muestra la totalidad de aplicaciones y tópicos que intervienen en el control del UAV. El diagrama muestra las principales aplicaciones y tópicos que se emplearon para la realización del vuelo en interiores del Iris.
Los módulos y aplicaciones del PX4 firmware se encuentran en el directorio *Firmware/src/-modules*
Los módulos del PX4 Firmware:
Sensors
Esta aplicación se encarga de obtener los datos de todos los sensores internos que contiene el Pixhawk y sensores externos que se le agregan al FCU, además de que se encarga de transformar los valores de entradas obtenidos del radio control a valores de alabeo, cabeceo, guiñada y la aceleración (empuje), también de interpretar un cambio de modo de vuelo mediante la activación de un botón.

Mavlink
Esta aplicación se dedica a implementar el protocolo de comunicación para el telemetría, con el fin de enviar y recibir mensajes con el formato de MAVLINK, por medio de un puerto serie, a continuación se muestran los principales mensajes empleados por mavlink:

- mavlink_msg_heartbeat
- mavlik_msg_sys_status
- mavlink_msg_highres_imu
- mavlink_msg_attitude

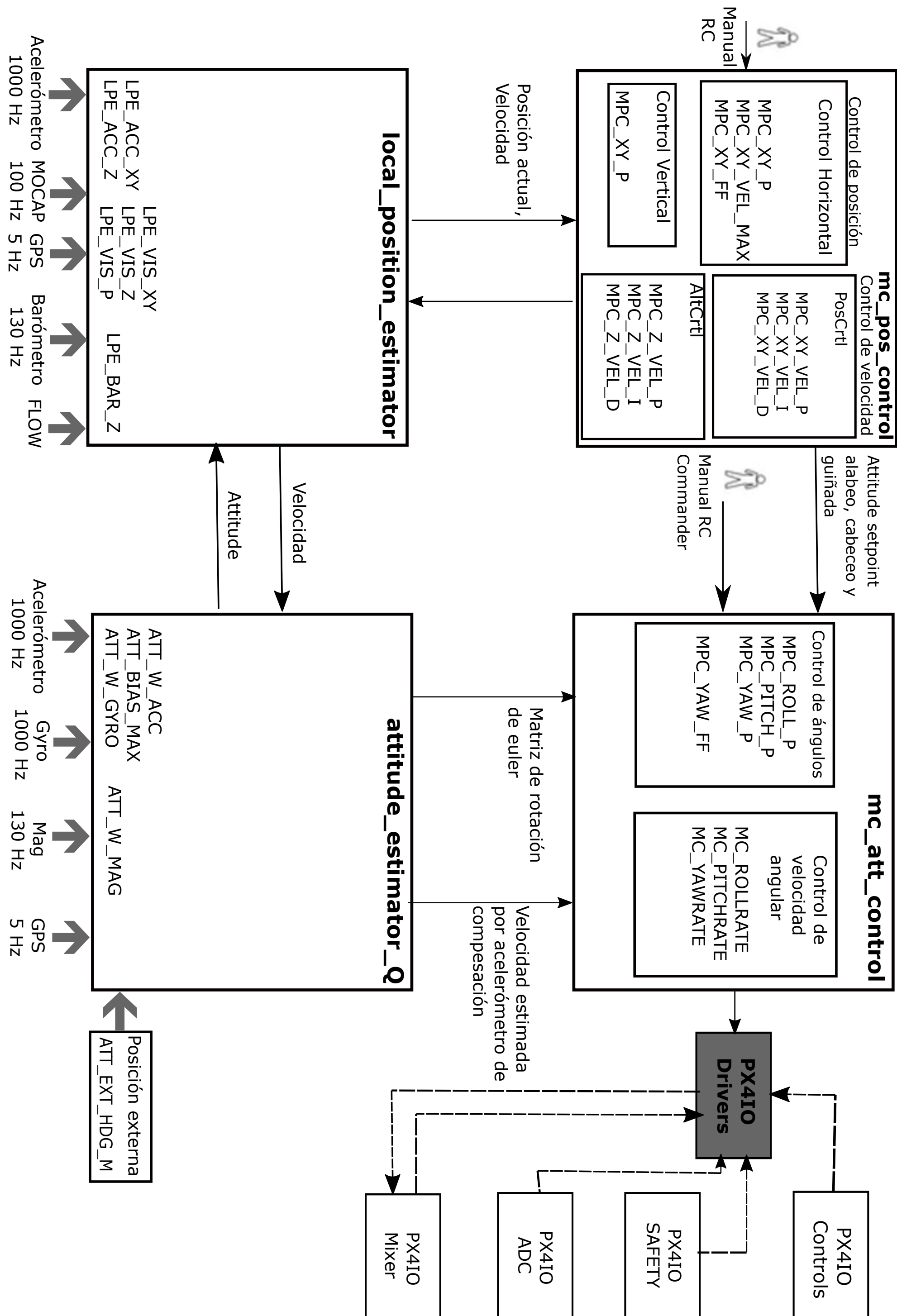

Figura 2-7: Arquitectura del PX4 Firmware.

- mavlin_msg_local_position_ned
- mavlink_msg_set_mode
- mavlink_msg_att_pos_mocap
- mavlink_msg_set_position_target_local_ned
- mavlink_msg_roll_pitch_yaw_thrust_setpoint

Drivers PX4

Son los controladores de los sensores que utiliza el cuadrirrotor, éstos son los que se encargan de publicar en diferentes tópicos la información obtenida de los sensores o dispositivos periféricos, a continuación se muestra los diferentes sensores internos y externos que contiene el Pixhawk:

- **hmc5883:** Es el controlador del magnetómetro, su interfaz de comunicación es i2c.
- **mpu6000:** Es el controlador del imu, contiene un acelerómetro y un giroscopio, utiliza una interfaz de comunicación SPI.
- **lsm303:** Contiene un acelerómetro y un magnetómetro, su interfaz de comunicación SPI.
- **l3gd2:** Es el giroscopio, contiene una interfaz de comunicación SPI.
- **meas_airspeed:** Es el controlador del sensor de aire, recibe los datos de presión diferencial y utiliza una interfaz de comunicación I2C.
- **ms5611:** Es el controlador del barómetro, este sensor permite obtener la presión atmosférica y tiene una interfaz de comunicación SPI.
- **GPS:** Este controlador proporciona una comunicación bidireccional entre el GPS y el pixhawk utilizando el protocolo de comunicación MAVLINK.
- **Driver PX4IO:** Se encarga de la comunicación entre los módulos PX4IO y PX4FMU. Como se muestra en la figura **2-7**, intercambian datos con lo controladores de los motores, del estado de la batería, canales ADC y de los controles RC.

- **Driver PX4fmu:** El módulo PX4fmu tiene un controlador de pines, donde los pines se pueden configurar a modo gpio o modo pwm, el primer modo sirve para controlar los pines como entrada/salidas digitales, el segundo modo sirve para configurar los pines como salidas PWM.

Commander
La aplicación commander se encarga de la máquina de estados global del sistema figura **2-8**. Esta aplicación maneja los led rojo, azul y el buzzer con el objetivo de indicar el estado actual del pixhawk, es decir, es la encargada de que las aplicaciones se ejecuten y realicen acciones de acuerdo al estado y sentido del sistema.

- **Armed State:** Es el estado armado del sistema, es decir, si los motores del cuadrirotor están listos para usarse. Cuenta con los estados: INIT, STANDBY, STANDBY_ERROR y ARMED_ERROR.

- **Main State_state:** Es el estado principal del sistema, se activa mediante el uso del control RC o desde la estación de tierra, esto indica que es el estado donde el usuario indica lo que suceda, los valores que puede tener este estado generalmente son los modos de vuelo manuales: MANUAL, ALTCTL, POSCTL, AUTO_MISSION, AUTO_RTL, AUTO_LOITER, ACRO, OFFBOARD.

- **Failsafe:** El sistema cuenta con una bandera para saber si el cuadrirotor se encuentra en estado a prueba de fallos.

- **Navigation State:** El estado de navegación contrario al estado principal indica lo que el UAV tiene qué hacer, cambia en función al estado principal y generalmente por cuestiones de seguridad como por ejemplo el estado de comunicación por el UAV. Los modos de vuelo en este estado son: ALTCTL, POSCTL, AUTO_MISSION, AUTO_RTL, AUTO_LOITER, ACRO, LAND, DESCEND, TERMINATION, OFFBOARD.

En la (figura **2-8**) se puede ver el diagrama total de los estados del sistema, básicamente cuando inicia el sistema de la unidad de control de vuelo éste entra en estado INIT, este primer estado el sistema verifica si todos los sensores del Pixhawk funcionan correctamente, después pasa al estado STANDBY si hay suficiente cantidad de batería de lo contrario si es inferior a 10 % pasa al estado STANDBY_ERROR, después se procede a verificar el estado de armado de los motores, de igual manera como el estado anterior, el sistema verificara si hay suficiente batería entra a los estados ARMED o ARMED_ERROR. Como se mencionó anteriormente, el estado principal coincide con los modos de vuelo del UAV. Los modos se

controlan mediante los interruptores del control RC, por lo tanto, sucede los mismo con el estado principal.
Si hay problemas de comunicación con el UAV, éste entra a modo de prueba de fallos y el estado de navegación se modifica de acuerdo a los datos que tenga el sistema. Por ejemplo, si el UAV se encuentra en modo manual y se pierde la señal del control RC, el sistema pasará al modo de navegación RTL (Return To Launch) donde utilizará las señales GPS para calcular el punto donde despegó, si dispone información sobre la posición local éste aterrizará controladamente, se puede dar el caso que no disponga toda la información para entrar a los estados anteriores por lo que en este caso el sistema entra en el estado de navegación TERMINATED y automáticamente se apagarán los motores.

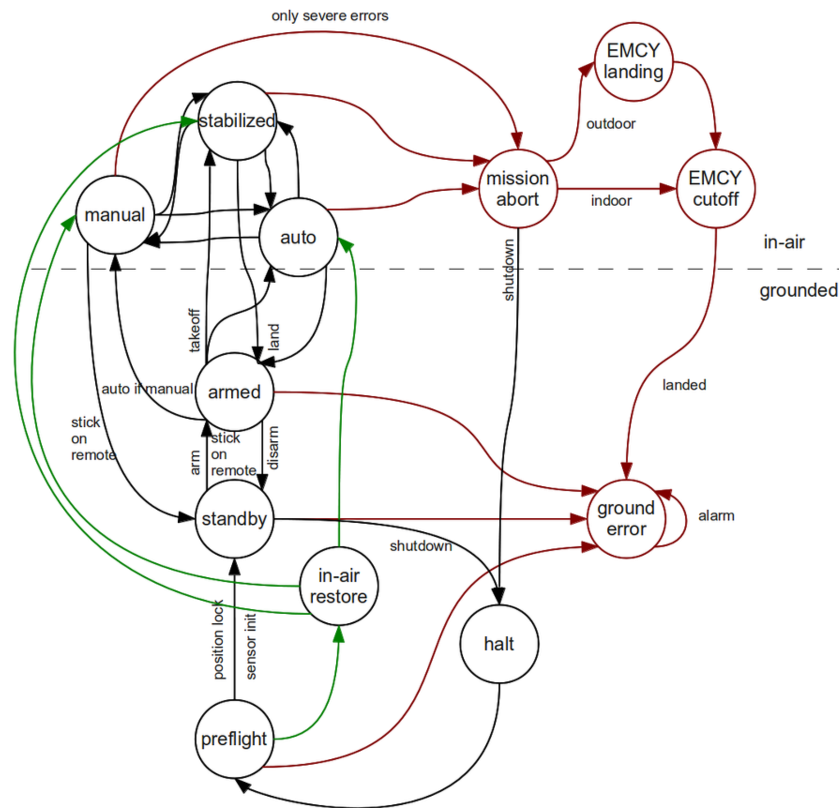

Figura 2-8: Estados del PX4 firmware

local_position_estimator

El estimador de posición LPE es un filtro de kalman extendido. Su función consiste en esti-

mar la posición en 3D y los estados de velocidad. Utiliza los datos medidos por la mayoría de los sensores internos (acelerómetro y barómetro) del Pixhawk, incluyendo los sensores externos, como el GPS y datos tipo mocap o visión que son obtenidos por medio de cámaras.

attitude_estimator_q

El estimador de los ángulos es un simple filtro complementario basado en un cuaternión, hace una estimación de la orientación del UAV, es decir, los valores de los ángulos de alabeo, cabeceo y guiñada por medio de los datos recibidos de los sensores (acelerómetro, magnetómetro y giroscopio).

mc_attitude_control

En la (figura **2-9**) se muestra el sistema de control de orientación utilizado por el Pixhawk. El control es implementado usando un controlador P para convertir el ángulo de error dentro de un valor de rotación deseado seguido por un controlador PID para convertir el error de velocidad de rotación deseada en un comando de motor de alto nivel [1].

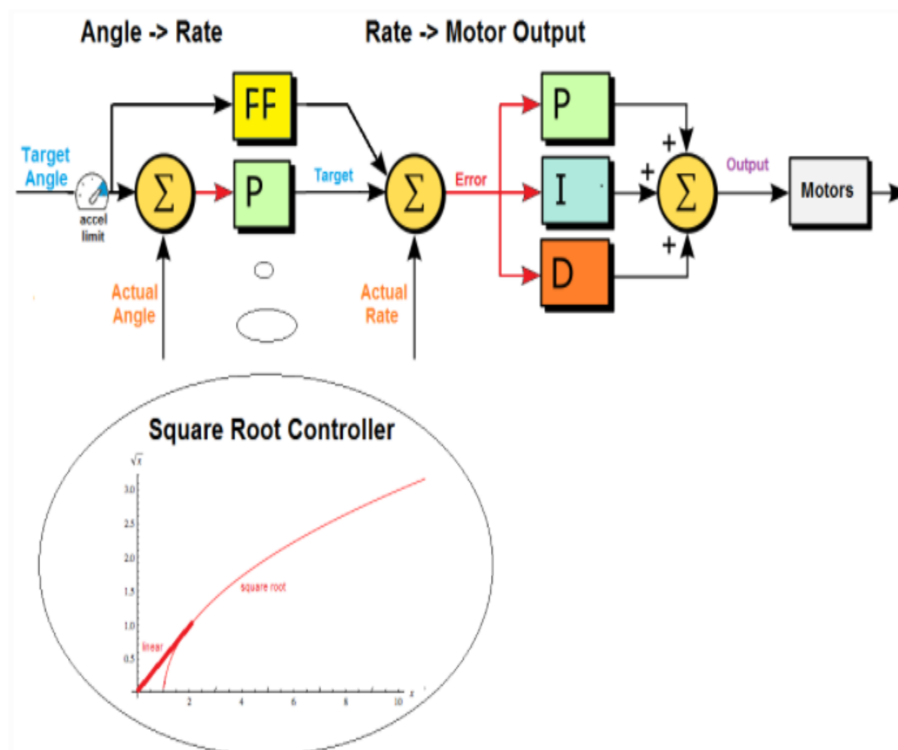

Figura 2-9: Diagrama del control del *attitude* para cada eje

mc_position_control
Utiliza los valores de posición local actual y las velocidades locales obtenidas del estimador de posición local y de la posición deseada para calcular alabeo, cabeceo, guiñada y empuje. Esta aplicación utiliza un control PID similar al de orientación.

systemlib
Esta librería se encarga de llamar a diferentes funciones como la de control de los mezcladores que su función principal consiste en llamar a las señales PWM dependiendo de la plataforma de vuelo que se esté utilizando.

Firmware PX4io
Como se muestra en la (figura **2-7**), los bloques de color naranja son los que maneja esta aplicación, cada uno tiene una determinada función que se encarga de ejecutar:

- **PX4io Controls:** Recibe los datos de la entradas de controles RC.

- **PX4io Safety:** Su función principal consiste en servir como un botón de seguridad que permite el arme y el desarme del sistema.

- **PX4io ADC:** Este bloque se encarga de habilitar las entradas ADC de 3.3v.

- **PX4io Mixer:** Su función consiste en controlar las entradas y salida del mezclador (controladores de los actuadores).

Los módulos son utilizados y configurados para un tipo de airframe (vehículo aéreo) determinado, como puede ser aviones de ala fija, cuadrirotores y VTOL como se muestra en la Tabla **2-2**. Para el 3DR Iris se utiliza el airframe *Multi copter.*

Tabla 2-2: Diferentes Módulos y tipos de airframes

	Fixed Wing	**Multi Copter**	**VTOL**
Navegador	navigator	navigator	navigator
Estimador	ekf_att_pos_estimator	attitude_estimator_q local_position_estimator	attitude_estimator_q local_position_estimator
Controlador	fw_att_control fw_pow_control_l1	mc_att_control mc_pos_control	vtol_att_control mc_att_control mc_pos_control fw_att_control fw_pow_control_l1

Como se mencionó anteriormente, la comunicación entre módulos se basa en la implementación de la estructura de datos uORB, donde el módulo puede publicar mensajes a otro módulo y al mismo tiempo subscribirse para recibir mensajes, por ejemplo, para el módulo atttude_estimator_q los mensajes enviados y recibidos son los siguientes:

- Mensajes Publicados
  - vehicle_attitude
  - control_state
- Mensajes suscripción
  - sensor_combined
  - vision_position_estimate
  - att_pos_mocap
  - airspeed
  - parameter_update
  - vehicle_global_position

2.2. ROS

El *Sistema Operativo Robótico* (ROS) [9] es un middleware robótico, es decir, un software que ayuda o asiste a una aplicación para comunicarse con otras aplicaciones, también es un

framework para el desarrollo de sofware para robots. ROS es un software de código abierto y se desarrolló en el 2007 en la universidad de Stanford y posteriormente el proyecto pasó al instituto de investigación Willow Garage. Al ser un framework aporta una gran variedad de librerías y herramientas para compilar y ejecutar código a través de múltiples ordenadores con el objetivo de simplificar la creación y comportamiento de robots robustos.
La arquitectura y las librerías de ROS están orientados para sistemas operativos del tipo UNIX (Linux, Ubuntu), también ROS está adaptado para sistemas operativos como Mac OSx, Windows, Arch, etc. pero de manera experimental, de igual manera ROS trabaja con diversas integraciones o librerías externas para desarrollar el entorno de trabajo. Las principales librerías externas que utiliza ROS son:

- **GAZEBO:** Es un simulador 3D de robots en interiores y exteriores. La integración entre *ROS y GAZEBO* se debe a que éste último tiene soporte a una gran variedad de robots y sensores que utilizan ROS, por lo tanto, el usuario puede simular trayectorias y comportamientos de un robot utilizando este integrador.

- **OpenCV:** Es una librería de algoritmos de visión por computadora, por lo que, al integrarse con ROS proporciona al usuario una mayor facilidad de publicar los datos de las cámaras a cualquier robot.

- **pointcloudlibrary:** Es una librería de percepción para la manipulación y procesamiento de datos e imágenes tridimensionales, tiene librerías de diversos sensores como el Microsoft Kinect.

- **ROS Industrial:** es un proyecto de código abierto que se extiende al área industrial como en la automatización de manufactura y róbótica.

ROS está liberada bajo los términos de la licencia BSD (Berkeley Software Distribution) y un software open source, gratuito para el uso comercial y de investigación. Promueve la reutilización de código, así los desarrolladores y científicos pueden usar el código de los repositorios, mejorarlo y compartirlo de nuevo.
Las áreas que incluye ROS actualmente se muestran a continuación::

- Un nodo principal de coordinación.

- Publicación o subscripción de flujos de datos.

- Multiplexación de la información.

- Creación y destrucción de nodos.
- Login.
- Parámetros de servidor.
- Pruebas de sistemas.

2.2.1. Espacio de trabajos y paquetes ROS

catkin
El compilador oficial del sistema ROS es *catkin*, el cual combina los macros de Cmake y los scripts de phython haciendo que el flujo de trabajo en cmake sea más eficiente, mejor distribución de paquetes, una mejor compilación y mejor portabilidad. El compilador *catkin* reemplaza al antiguo compilador *rosbuild*.

Espacio de trabajo ROS
Para escribir el código ROS es necesario hacer y configurar un espacio de trabajo para que el código ROS se ejecute dentro de ese espacio. El es espacio de trabajo son directorios que están relacionados con ROS. En general se pueden tener varios espacios de trabajo pero sólo se puede usar y ejecutar uno a la vez. Para hacer un espacio de trabajo catkin se ejecuta el siguiente código en una nueva terminal linux:

```
usuario@mario1577$ mkdir -p ~/catkin_ws/src
usuario@mario1577$ cd ~/catkin_ws/src
usuario@mario1577$ catkin_init_workspace
```

El espacio de trabajo es el directrio llamado *catkin_ws*, el comando *catkin_init_workspace* crea el archivo CMakeLists.txt dentro de la carpeta de trabajo *src* del directorio catkin_ws.

```
usuario@mario1577$ cd ~/catkin_ws
usuario@mario1577$ catkin_make
```

Para compilar en el directorio del espacio de trabajo se utiliza el comando *catkin_make*. Este comando da como resultado dos nuevos directorios: *build* y *devel*. El directorio *build* es el directorio donde se guardan los resultados como librerías y programas ejecutables, mientras que el directorio *devel* contiene los archivos de configuración del espacio de trabajo. Para convertir un espacio de trabajo en un directorio de alto nivel se utiliza el comando:

```
usuario@mario1577$ source devel/setup.bash
```

Paquetes ROS
ROS está organizado por paquetes, cada uno contiene código ejecutable, datos y documentación. Los paquetes se descargan y se instalan en el espacio de trabajo dentro del directorio *src*. Cada paquete debe estar incluido en el archivo *CMakeLists.txt* y en el archivo *package.xml*. Para crear un nuevo paquete se utiliza el paquete:

```
usuario@mario1577$ cd ~/catkin_ws/src
usuario@mario1577$ catkin_create_pkg paquete [dependencia 1.....n]
```

El comando *catkin_create_pkg* se encarga de crear el nuevo paquete junto con las dependencias en el directorio *src*.

Sistema de dependencias
Las dependencias son librerias o herramientas externas requeridas por determinados paquetes, este sistema de dependencias no se instalan automaticamente en ROS. Para instalar y descargar dependencias, ROS proporciona una herramienta llamada *rosdep*.
Las dependencias basicas que utiliza ROS son las siguientes:

- **rospy:** Es un tipo de libreria de cliente pynthon que permite interactuar con los tópicos, servicios y párametros de ROS.
- **roscpp:** Es una librería de C++ implementada por ROS que permite interactuar con los tópicos, servicios y párametros de ROS.
- **std_msgs:** Son los tipos de datos que se emplean en ROS, como tambien se define la estructura basica o multidimensional del mensaje.

2.2.2. Nodo maestro y nodos ROS

La comunicación de ROS se basa en unos módulos llamados nodos, donde un nodo es un código ejecutable en una computadora o en varias computadoras o robots. La principal ventaja de utilizar ROS consiste en que cada nodo es una parte de un sistema.
Nodos: Los nodos son procesos independientes que realizan una determinada acción o tarea con la capacidad de comunicarse con otros nodos, es decir, con otras partes del sistema. Los nodos son creados por programas utilizando el lenguaje C++ o phynton. El principal

propósito de ROS radica en que cada nodo se comunica uno con otro enviando y recibiendo mensajes.
Nodo Maestro: Es el nodo principal de todo el sistema ROS, éste se encarga de encontrar otros nodos y de establecer la comunicación entre ellos, el nodo maestro proporciona los servicios de registro. La localización de nodos y la comunicación entre ellos se logra mediante la utillización del protocolo de comunicación TCPROS/IP (Transmission Control Protocol ROS/Internet Protocol).
Si el nodo maestro no es ejecutado, no notificara a los otros nodos ROS de su existestencia y por lo tanto los nodos ROS no se ejecutan y no establecen comunicación entre ellos.

Par invocar el nodo maestro es necesario ejecutar el comando *roscore*, éste comando es un servicio que proporciona la información de conexión a otros nodos. Todos los nodos nuevos se conectan al comando *roscore*, el cual les proporciona la información necesaria para que éstos puedan publicar y subscribirse a otros nodos. Para iniciar el sistema ROS siempre es necesario ejecutar el comando *roscore*. Al ejecutar este comando empieza lo siguiente:

- **Ros Master**
- **Ros parameter Server:** Se utliza para configurar cualquier nodo ROS, esto permite almacear estructura de datos con el fin de cambiar parámetros de algoritmos robóticos. Los los parámetros asociados con el Maestro son los siguientes: */rosdistro: kinetic, /rosversion:1.11.13*
- **rosout:** nodo para subscribirse, iniciar sesión y publicar los mensajes

El nodo maestro en una terminal linux se ejecuta con el comando:

```
$ roscore
```

Obteniendo en la salida de la terminal linux lo siguiente:

```
... logging to /home/harman/.ros/log/94248b4a-3f05-11e5-b5ce-
00197d37ddd2/roslaunch-Laptop-M1210-2322.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.
started roslaunch server http://Laptop-M1210:46614/
```

```
ros_comm version 1.11.13
SUMMARY
========
PARAMETERS
* /rosdistro: kinetic
* /rosversion: 1.11.13
NODES
auto-starting new master
process[master]: started with pid [2334]
ROS_MASTER_URI=http://Laptop-M1210:11311/
setting /run_id to 94248b4a-3f05-11e5-b5ce-00197d37ddd2
process[rosout-1]: started with pid [2347]
started core service [/rosout]
```

La variable *ROS_MASTER_URI* contiene un string en forma *http://hostname:11311/*, este indica que el comando *roscore* es accesible en el puerto 11311.
El *parámetro ROS de servicio* se utiliza para configurar cualquier nodo ROS, esto permite almacenar estructura de datos con el fin de cambiar parámetros de algoritmos robóticos.
Algunos comandos básicos para determinar los nodos son los siguientes:

- ***roscore:*** Invoca el nodo maestro permitiendo la comunicación entre nodos.
- ***rosnode list:*** Muestra la lista de los nodos activos.
- ***rosnode info:*** Muestra información acerca del nodo.
- ***rosnode kill:*** Cancela la ejecución de cualquier nodo
- ***rosnode ping:*** Prueba de conectividad para los nodos.
- ***rosnode cleanup:*** Limpia información de los nodos.

Comando rosrun
Este comando ejecuta un archivo o nodo ROS ejecutable de cualquier paquete. Se tiene la opción de modificar los parámetros del nodo o archivo ejecutable.
Sintaxis del comando rosrun:

```
usuario$ rosrun paquete nodo_ejecutable nombre_parametro:=valor_deseado
```

Donde *paquete* se refiere al nombre del paquete ubicado en el espacio de trabajo, *nodo_ejecutable* se refiere al nombre del archivo .cpp o .py que se utilizó para crear el nodo y *nombre_parametro:=valor_deseado* se refiere al parámetro que se desea agregar o modificar del archivo ejecutable.

Comando roslaunch

Es una herramienta de ROS para lanzar varios nodos en la ventana ssh o terminal linux. Los archivos *.launch* son archivos XML, así como para establecer parámetros en el servidor de parámetros ROS. *roslaunch* toma uno o más archivos de configuración *.launch* que especifican los parámetros a configurar y los nodos para lanzar, así como en las máquinas en las que se deben ejecutar.
Sintaxis del comando de ejecución *roslaunch*:

```
usuario@mario1577$ roslaunch paquete_ros archivo_launch
```

Donde paquete_ros se refiere al nombre del paquete ubicado en el espacio de trabajo y archivo_launch es el nombre del archivo .launch que se desea ejecutar.
El uso de archivos .launch te sirve para cambiar cualquier páarametro de configuración de varios nodos. También este tipo de archivos es usado para remapear los mensajes de un nodo a otro nodo ROS.
Comandos utilizados en el archivo *.launch*:

```
<launch>
<node name="" pkg=""
type="" output="screen" />
<node name="" pkg=""
type="" output="" />
</launch>
```

la instrucción <launch> indica el comienzo y la finalización del archivo *.launch* en la primera línea de instrucciones *node name=* se refiere al nombre que recibirá el nodo del archivo *.cpp* y *pkg=* al paquete donde se encuentra ubicado el ejecutable. En la segunda línea, la instrucción *type=* se refiere al nombre y tipo de archivo (.cpp, .py) ejecutable, por último, la intrucción *output* se refiere al lugar donde se despliegan los mensajes, generalmente se usa el parámetro *“screen”* que consiste en un despliegue de datos en ventanas ssh o terminales linux.

2.2.3. Tópicos y mensajes

Como se menciono anteriormente, los *nodos* se comunican con otro debido a que hay un intercambio de información y datos entre ellos,. La manera de hacer este intercambio es por medio de *tópicos*. Un *tópico* es un bus de datos sobre los cuales los nodos intercambian mensajes.
Los *tópicos* implementan una metodología de comunicación de *publicar/suscribir*. Para que un nodo empiece a publicar mensajes a un tópico es necesario:

- Declarar que el nodo *publicará* mensajes.
- Definir el tipo del mensaje.

Los nodos que desean recibir mensajes de un tópico de otro nodo necesitan:

- Suscribirse a ese tópico.
- Hacer un solicitud a *roscore*

Publicación de mensajes a tópicos con C++
Configuración de un tópico publicador:

```
#include <ros/ros.h>
#include <std_msgs/Int32.h>
int main(int argc, char **argv) {
  ros::init(argc, argv, "count_publisher");
  ros::NodeHandle node;
  ros::Publisher pub = node.advertise<std_msgs::Int32>("counter", 10);
  ros::Rate rate(1);
  int count = 0;
  while (ros::ok()) {
    std_msgs::Int32 msg;
    msg.data = count;
    pub.publish(msg);
    ++count;
    rate.sleep();
  }
  return 0;
}
```

donde la intrucción:

```
ros::Publisher pub = node.advertise<std_msgs::Int32>("counter", 10);
```

se encarga de crear el publicador y de definir el tipo de mensaje que se enviará, mientras que la intrucción:

```
std_msgs::Int32 msg;
```

se dedica a crear los *mensajes* que se enviarán al tópico deseado.

Suscripción a tópicos con C++
Configuración de un tópico subscritor:

```
#include <ros/ros.h>
#include <std_msgs/Int32.h>
#include <iostream>
void callback(const std_msgs::Int32::ConstPtr &msg) {
std::cout << msg->data << std::endl;
}
int main(int argc, char **argv) {
  ros::init(argc, argv, "count_subscriber");
  ros::NodeHandle node;
  ros::Subscriber sub = node.subscribe("counter", 10, callback);
  ros::spin();
}
```

Donde la instrucción:

```
void callback(const std_msgs::Int32::ConstPtr &msg) {
```

define la función *callback* y el tipo de mensaje que se estará utilizando, mientras que la instrucción:

```
ros::Subscriber sub = node.subscribe("counter", 10, callback);
```

crea el suscriptor donde se define el nombre del tópico, el segundo argumento consiste en el tamaño de mensajes recibidos, es decir, cuando se reciban más de 10 se empezarán a descartar los mensajes antiguos.
Comandos para tópicos
Estos comandos son usados para registrar y depurar los tópicos que se estén usando:

- **rostopic bw:** Despliega el ancho de banda usado por el tópico.
- **rostopic echo:** Imprime los mensajes a la pantalla.
- **rostopic find:** Encuentra los tópicos por tipo.
- **rostopic hz:** Despliega la velocidad de pulicación del tópico.
- **rostopic info:** Muesta la información acerca del tópico activo.
- **rostopic list:** Lista activa de tópicos.
- **rostopic pub:** Publica datos a un tópico.
- **rostopic type:** Muestra el tipo de tema.

Tipos de Mesajes
El paquete *std_msgs* define los tipos de datos primarios que son usados para construir todos los mensajes utilizados en ROS. Los mensajes pueden encontrarse en los paquetes *std_msgs* y *commmon_msgs* (Tabla **2-3**).

Tabla 2-3: Tipos de datos para Variables de mensajes ROS

ROS	Serialización	C++
bool	entero de 8-bit	uint8_t
int8	entero de 8-bit	int8_t
int16	entero de 16bit	int16_t
int32	entero de 32-bit	int32_t
int64	entero de 64-bit	int64_t
float32	32-bit IEEE float	float
float64	64-bit IEEE float	double
string	ASCII string	std::string
time	secs/nsecs 32-bit ints	ros::Time

Los principales comandos utilizados en ROS se muestran en la Tabla **2-4**.

Tabla 2-4: Comandos de uso general en ROS

Comandos ROS	Descripción
roscore	Ejecuta el nodo maestro.
rosrun	Ejecuta cualquier nodo del espacio de trabajo.
rosnode	Despliega la lista e información de los nodos que se están ejecutando.
rostopic	Despliega la lista, información y banda ancha del tópico.
rosservice	Conecta a un Servidor/Cliente de ROS.
rosparam	Almacena y manpula los parametros de diferentes nodos o tópicos.
roslaunch	Conecta múltiples nodos con un archivo de tipo .XML
rqt_graph	Proporciona un complemento de tipo GUI para visualizar los nodos y tópicos.
rqt_plot	Proporciona un complemento de tipo GUI para visualizar valores numéricos en una gráfica 2D.

3 Desarrollo Experimental

Las APIs (Interfaz de Programación de Aplicaciones) dedicadas para la robótica permiten controlar el pixhawk u otras unidades de control de vuelo (FCU) desde afuera sin la necesitad de utilizar su entorno computacional del FCU. Las APIs se comunican con el PX4 utilizando MAVLink. El Pixhawk utiliza unas APIs que incluyen *ROS* o *Dronecode*.

3.1. Modo no abordo y configuración del Pixhawk

3.1.1. Configuración del hardware de Pixhawk

Las recientes versiones del firmware PX4 no soportan la recepción de datos tipos MOCAP, la versión más actualizada tiene agregado el estimador *EKF* (Filtro de Kalman extendido) y éste no es compatible con los datos tipo MOCAP.

El estimador *LPE* (Estimador de Posición Local) es el que soporta la entrada de datos del tipo visión o MOCAP. En la unidad Pixhawk viene por defecto el estimador *EKF* (figura **3-1**), la opción de cambiar al estimador LPE esta inhabilitada debido a que la tarjeta pixhawk no tiene suficiente memoria para tener disponible los dos estimadores por lo que es necesario cambiar el módulo *EKF* por el *LPE* desde el código fuente.

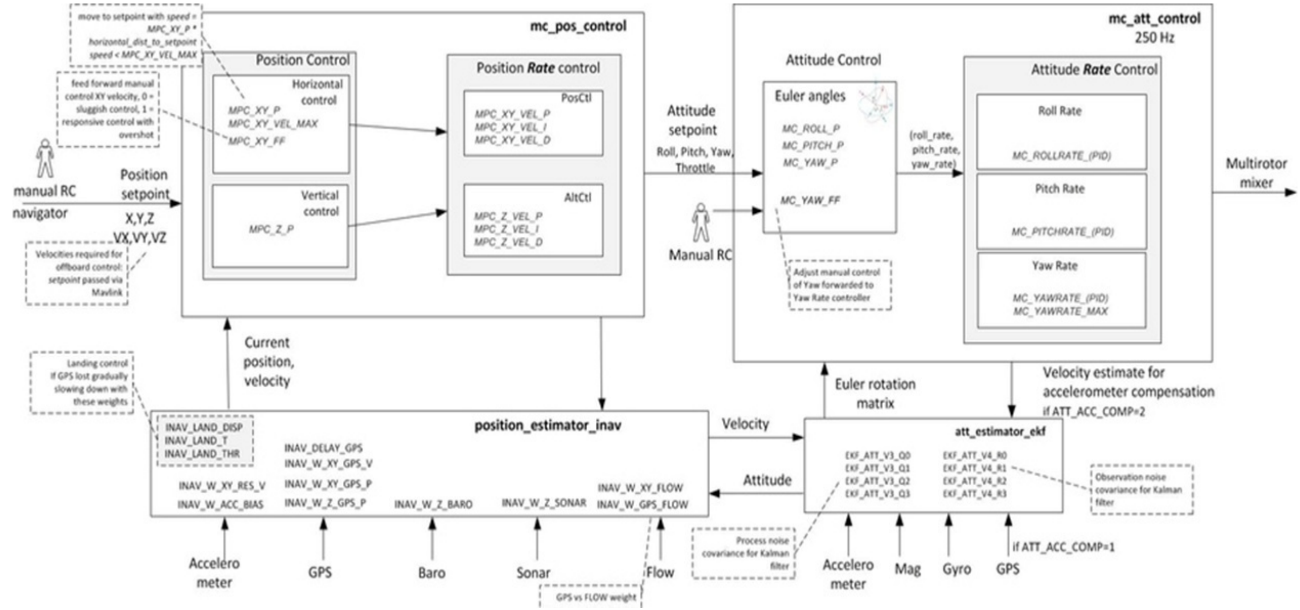

Figura 3-1: Diagrama general de los módulos del PX4 con el estimador EKF

Las maneras de cambiar el módulo EKF por el LPE es por la estación de tierra Qground-Control o construyendo y actualizando por medio de Nuttx desde linux.

Método estación de tierra: Utilizando QgroundControl es necesario primero actualizar la versión del firmare a la última versión, ya actualizada se procede a cambiar los módulos por medio de un archivo tipo *px4*. Los pasos a ejecutar son los siguientes:

1. Ejecutar QgroundControl y abrir la pestaña *firmware* (figura **3-2**).

2. Seleccionar PX4 Autopilot y configuración avanzada.

3. Seleccionar el archivo **px4fmu-v2_default.px4** que contenga la última versión del firmware.

4. Seleccionar el archivo **px4fmu-v2_lpe.px4** que es el que contiene el módulo *LPE*.

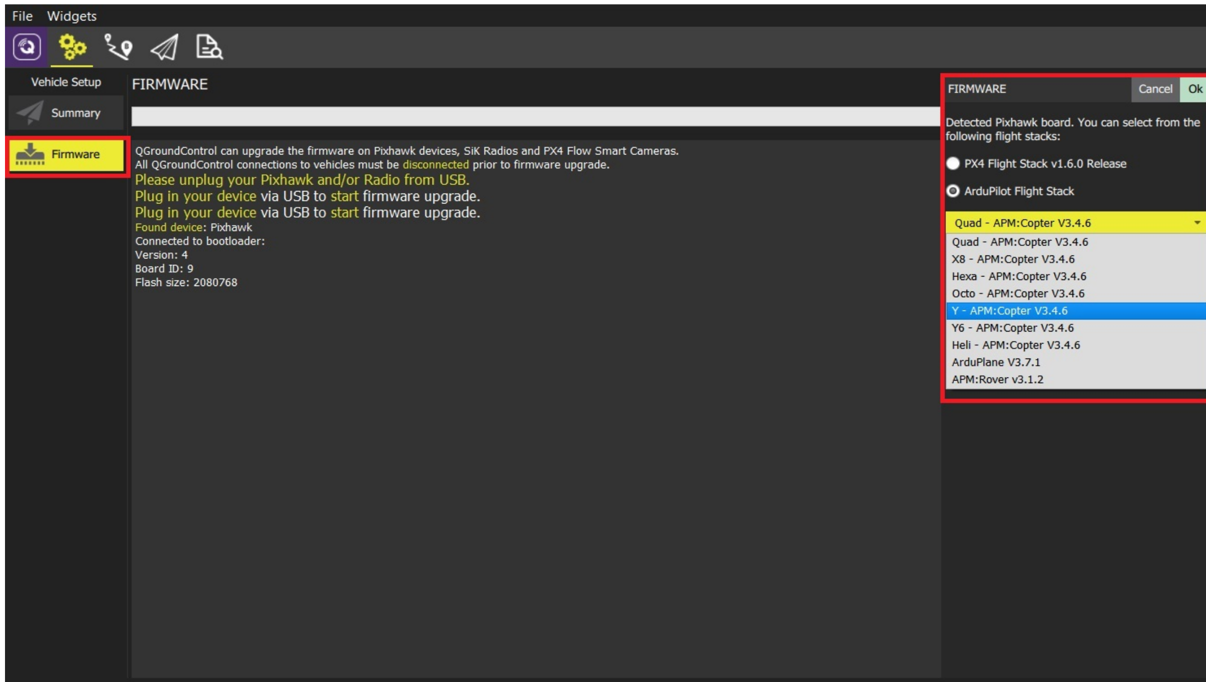

Figura 3-2: Pestaña firmware del QgrondControl

Este método no es muy recomendado debido a que en muchas ocasiones el módulo del firmware no se construye y se compila bien. Por lo tanto, se presentan múltiples problemas en el cuadrirotor a la hora de recibir los datos MOCAP, además de que la mayoría de las veces no se sube el archivo *.px4* que contiene el módulo LPE con su respectiva versión del firmware, por lo que la mejor manera de cambiar de módulo es usando las herramientas del entorno de desarrollo utilizando *nuttx* desde linux.

Método Nuttx: Para este método es necesario descargar el entorno y las herramientas de desarrollo del PX4 Autopilot en una máquina linux. Estas herramientas permiten construir y compilar en cualquier plataforma PX4 (Pixhawk, Raspberry PI, Crazyfile, Qualcomm Snapdragon Flight hardware, etc.) Para intalar este entorno es necesario seguir el procedimiento mencionado en la página de desarrolladores del PX4.

1. Instalar Herramientas y entorno de desarrolo PX4 (Ver [3]).

2. Descargar el firmware PX4.

3. Abrir una terminal linux y construir el firmware PX4.

4. En la misma terminal cargar el firmware al Pixhawk.

5. Cargar y subir el módulo *LPE* al Pixhawk

El código para construir y cargar el firmware y el módulo *LPE* al Pixhawk en una terminal linux es el siguiente:

```
$cd Firmware

$make px4fmu-v2_default

$make px4fmu-v2_default upload
 Erase   : [====================] 100.0%
 Program: [====================] 100.0%
 Verify  : [====================] 100.0%
 Rebooting.
 [100%] Built target upload

$make px4fmu-v2_lpe
```

3.1.2. Sintonización del estimador LPE para MOCAP o visión

Los datos MOCAP se utilizan para la actualización de la posición local en relación con el origen, así mismo, los sistemas MOCAP como Optitrack sirven para obtener los datos de la orientación, teniendo la opción de integrar estos datos al módulo del estimador de ángulos del Pixhawk.
Para sistemas de visión el mensaje *MAVLINK* usado para enviar los datos de posición es *VISION_POSITION_ESTIMATE* y el mensaje para sistemas de captura de movimiento como Optitrack es *ATT_POS_MOCAP*.
Para habilitar o deshabiltar la entrada de datos de tipo visión o MOCAP, es necesario configurar algunos parámetros desde QGroundControl o la ventana de comandos NSH shell, dichos parámetros son:

1. El primer valor del parámetro a configurar es: ***SYS_MC_EST_GROUP***, este parámetro define los estimadores que utilizan los cuadrirrotores y VTOL. Los valores usados por este parámetros son:
   - **1**: local_position_estimator, attitude_estimator_q
   - **2**: ekf2

Por lo que se cambia el valor de **2** → **1** para utilizar el estimador *LPE*.

2. Para habilitar la entrada de datos de posicionamiento externo el valor parámetro a configurar es: ***ATT_EXT_HDG_M***. Este parámetro habilita y deshabilita la entrada de datos de tipo vision o MOCAP, los valores son:
   - **0:** Deshabilita la entrada de datos de posicionamiento externo.
   - **1:** Habilita la entrada de datos de posicionamiento de tipo visión
   - **2:** Habilita la entrada de datos de posicionamiento de tipo MOCAP

   El sistema de cámaras Optitrack es un sistema MOCAP, se configura el valor de **0** → **2** para datos *MOCAP* (ver figura **3-3**).

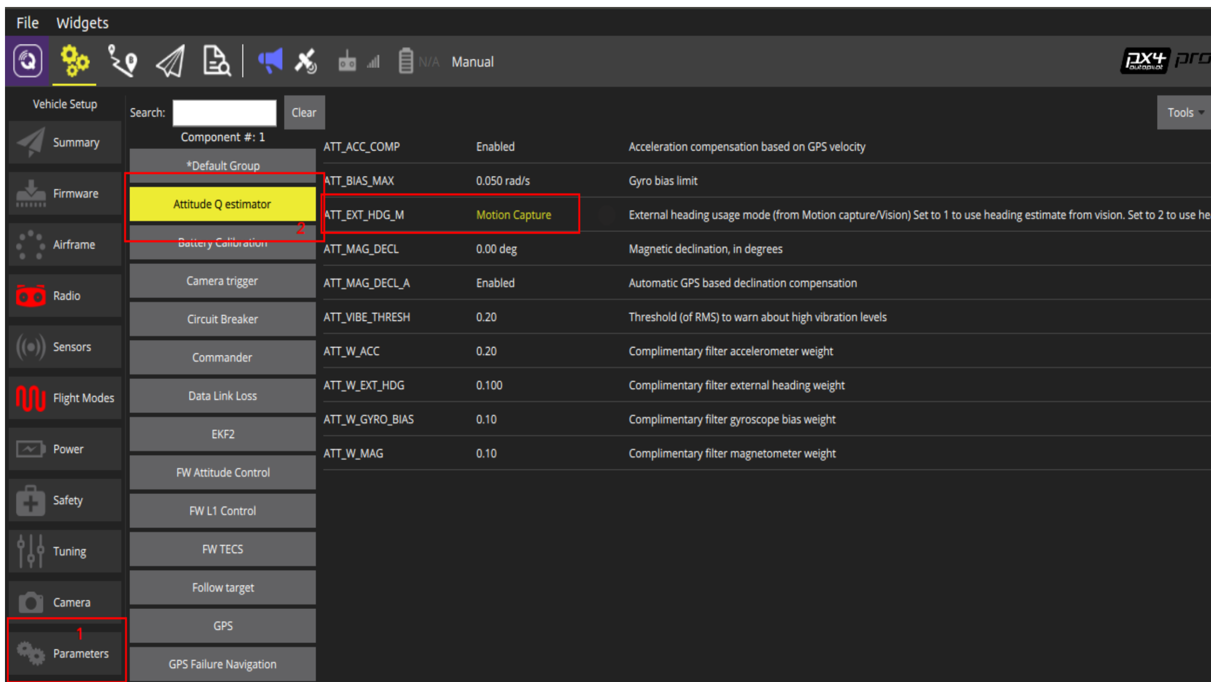

Figura 3-3: Habilitando la entrada de datos de posición tipo MOCAP.

Parámetro LPE_FUSION

Los *datos de fusión* es el proceso de integrar múltiples fuentes de datos obtenidos por sensores para producir una fuente de datos más precisa, este parámetro modifica la máscara de bits que controla la fusión de datos, 0 → 255.

Máscara de bits LPE_FUSION:

- **0:** fuse GPS, habilita el uso del sensor GPS.
- **1:** fuse optical flow, habilita el uso del sensor Optical flow.
- **2:** fuse vision position, habilita el envío de datos por visión.
- **3:** fuse vision yaw, habilita el cálculo de la guiñada por visión.
- **4:** fuse land detector, habilita el detector de tierra.
- **5:** flow gyro compensation, habilita la compensación del giroscopio utilizando el sensor flow.
- **6:** fuse baro, habilita el uso del barómetro.

Para el uso del sistema de cámaras Optitrack en el Pixhawk se deben desactivar los parámetros relacionados con el sensor optical flow enviando su respectivo número a MAVLink. Es importante desactivar el parámetro *fuse baro* para obtener una altitud más exacta y reducir el drift en el eje z.

3.1.3. Control no abordo

El control no abordo es una modalidad para controlar el software autopilot utilizando un sofware que se está controlando fuera del Pixhawk. Esto se realiza a través del protocolo MAVLink, donde los principales mensajes MAVLink que se utilizan son:

- **SET_POSITION_TARGET_LOCAL_NED:** Establece la posición deseada del vehículo en un marco de coordenadas NED utilizando un controlador externo para controlar el vehículo.
- **SET_ATTITUDE_TARGET:** Establece la altitud deseada del vehículo utilizando un controlador externo para controlar el vehículo.

Para activar el control no abordo es necesario configurar dos parámetros del firmware:

1. **Activación del control no abordo por RC y MAVLink:** Para asignar un canal o interruptor para la activación del modo en el RC, es necesario modificar el parámetro *RC_MAP_OFFB_SW*, a este parámetro se le asignan valores de **0** a **18** donde cada

uno representa un canal o interruptor. La mayoría de los *FCU* al activar el *modo no abordo* desde un interruptor, se recibe un mensaje de "*Estado denegado*"debido a que este modo es peligroso por lo que la mejor manera de acceder a este modo es enviando un mensaje MAVLink.

La manera más fácil de enviar el mensaje MAVLink es utilizando *MAVROS*, al hacer un nodo o un archivo .cpp, se debe de escribir el siguiente código para acceder al *modo de control no abordo*:

```
mavros_msgs::SetMode offb_set_mode;
offb_set_mode.request.custom_mode = "OFFBOARD";
```

NOTA: Para evitar que el modo no abordo sea rechazado los datos enviados deben enviarse dentro de los 0.5s de haber activado el modo.

2. **Habilitar interfaz de computadora compañera:** Ir al parámetro SYS_COMPANION en la estación QGC o NSH shell y seleccionar uno de los valores. Este parámetro activará una transmisión MAVLink en el puerto Telem2 con flujos de datos específicos con la velocidad de baudios adecuado.

   Valores:

   - **0:** Desactivado computadora de compañía.
   - **10:** FrSky Telemetria.
   - **20:** Tarjeta Crazyflie.
   - **57600:** Conexión de computadora de compañía (57600 baud, 8N1).
   - **157600:** OSD (57600 baud, 8N1).
   - **921600:** Conexión de computadora de compañía (921600 baud, 8N1)

Nota: Se requiere reinicio del FCU.
Los valores **57600** y **921600** son los que se deben activar para habilitar el uso de computadoras de compañía.

3.1.4. Configuración del hardware del control no abordo

Hay tres maneras de establecer la comunicación no abordo:

1. **Radios seriales:** Esta comunicación hace uso de las diferentes tecnologías inalámbricas que usan el protocolo serial para la transmisión y recepción de datos, en el uso de cuadrirotores los radios seriales mas comunes son: Lairdtech RM024, xbee, 3DR telemetrias etc. En la figura **3-4** se muestra un diagrama de esta configuración.

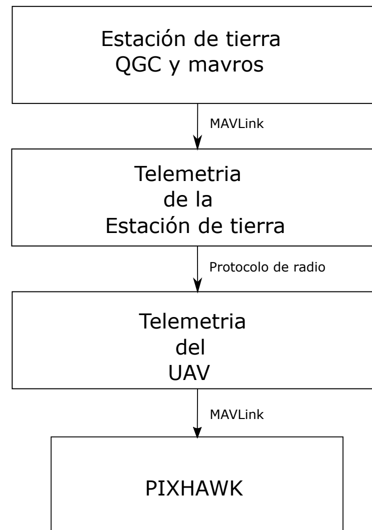

Figura 3-4: Esquema de configuración de radios seriales

Generalmente la estación de tierra es una computadora con la estación QGC o una computadora con linux donde se tiene instalado MAVROS, en esa computadora tiene conectado un telemetría que se encarga de enviar los datos a otro telemetría montado en el puerto TELEM2 del Pixhawk o cualquier otro FCU.

2. **Computadora a bordo:** Es una pequeña computadora montada en el cuadrirrotor, conectada al FCU a través de un adaptador UART a un adaptador USB. En esta configuración (ver figura **3-5**) generalmente la computadora montada es configurada como computadora de compañía en el FCU como el Pixhawk, en esa computadora está instalado QGC y sirve como estación de tierra para enviar comandos MAVLink por medio de un adaptador serial a un puerto del FCU.

   Ejemplos de computadoras de compañía:

   - Odroid C1, Odroid XU4
   - Raspberry Pi

- Intel NUC
- Nvidia jetson

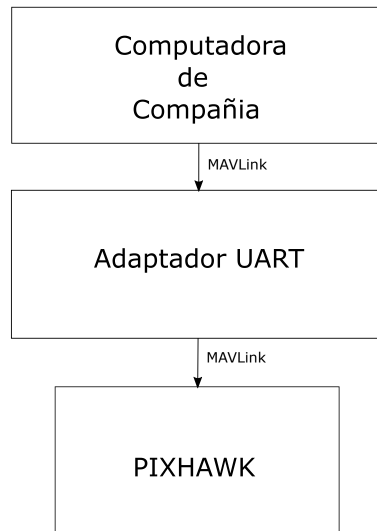

Figura 3-5: Esquema de configuración de computadora montada

3. **Computadora montada con conexión WiFi a ROS:** Es una pequeña computadora montada en el cuadrirrotor, configurada como computadora de compañía, la diferencia con la anterior configuración consiste en que la computadora de compañía trae integrado un módulo WiFi y sirve de enlace con la estación de tierra donde ésta controla remotamente a la computadora de compañía ejecutando ROS y MAVROS. Las computadoras de compañía nombradas anteriormente también sirven para esta configuración (ver figura **3-6**).

3.2. MAVROS

MAVROS es un paquete de ROS que permite a ROS interactuar con el software PX4 Autopilot utilizando el protocolo MAVLink, también permite interactuar con la estación QGC debido a que la estación utiliza el mismo protocolo.
Generalmente MAVROS está habilitado para comunicarse con cualquier FCU que utilice PX4 pero también se puede establecer comunicación entre el FCU y una computadora de compañía.
Las características de MAVROS son:

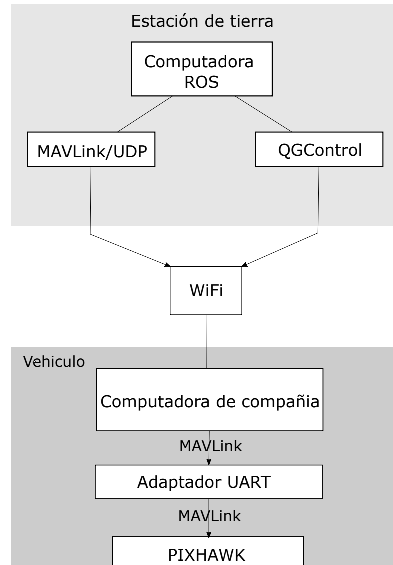

Figura 3-6: Esquema de configuración de computadora montada con enlace WiFi a ROS

- Comunicación con el software autopilot.
- Proxy para la estación QGC (Serial, UDP, TCP)
- Un sistema para la translación de código ROS a MAVLink
- Soporte para el sensor PX4FLOW
- Soporte para el modo no abordo.

3.2.1. MAVLink

MAVLink es un protocolo de mensajes [8], diseñado para intercambiar información entre un UAV y una estación GCS o un subsistema, usando un canal de comunicación serial. Se puede usar para transmitir la orientación del vehículo, su ubicación GPS y velocidad.

La estructura de un paquete MAVLink se divide en en 7 grupos donde la longitud mínima del paquete sin carga es de 8 bytes mientras que la longitud máxima es de 263 bytes (ver figura **3-7**)
En la Tabla **3-1** se muestra una definición de cada componente de la estructura de un paquete de mensajes de MAVLink.

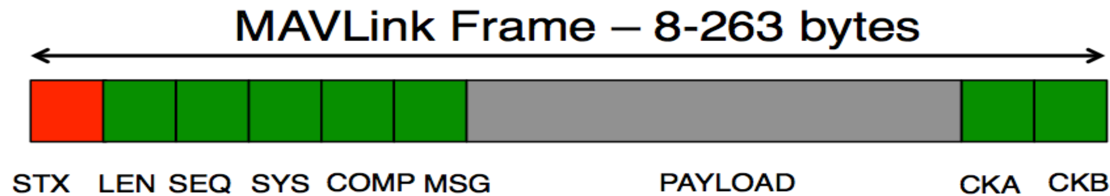

Figura 3-7: Estructura general de un aquete MAVLink

Tabla 3-1: Estructura detallada de un paquete de mensajes MAVLink

Nombre	Índice (Bytes)	Valor	Explicación
Inicio	0	v1.0: 0xFE (v0.9: 0x55)	Indica el inicio de un nuevo paquete.
Longitud de carga	1	0-255	Indica la longitud de la siguiente carga (n).
Secuencia de paquete	2	0-255	Permite detectar los paquetes perdidos. Cada componente cuenta con una secuencia.
Sistema ID	3	1-255	Permite diferenciar los diferentes MAVs en la misma red, es el ID del sistema de envío.
Componente ID	4	0-255	Permite diferenciar los diferentes componentes del mismo sistema,
Mensaje ID	5	0-255	Identificador del mensaje, indica cómo debe decodificarse correctamente.
Datos	6 a (n+6)	0-255	Los datos dentro del mensaje, depende del Mensaje ID.

3.2.2. nodo y plug-ins MAVROS

El *nodo MAVROS* proporciona un driver que comunicación para varios tipos de fcu con el software Autopilot además de que proporciona un puente MAVLink de tipo UDP para las estaciones de tierra como QGC, MAVROS proporciona un nodo principal y varios plugins que incluyen diversos tópicos para la obtención y el envío de datos. Un ejemplo de nodos de MAVROS se muestra en la figura **3-8**.
Nodo MAVROS: Es el nodo de comunicación principal, la función de este nodo es la de enviar mensajes MAVLINK al Autopilot y de recibir los mensajes del Autopilot. Al ejecutar este nodo también se puede recibir información de tipo diagnóstico por parte del software Autopilot. MAVROS se puede conectar al fcu de forma serial, UDP y TCP.

Conexiónes URL para el nodo MAVROS:

- **Serial:** /path/to/serial/device[:baudrate]
- **UDP:** udp://[bind_host][:port]@[remote_host][:port][/?ids=sysid,compid]
- **TCP:** tcp://[server_host][:port][/?ids=sysid,compid]

Plugin global_position: Publica información de la posición global por el FCU y los datos primarios del GPS. Los tópicos para suscribirse son:

- *global_position/global*
- *global_position/local*

Plugin imu_pub: Su función es la de publicar el estado del sensor IMU. Tópicos para suscribirse:

- *imu/data:* Muestra los datos de la IMU y su orientación calculada por la FCU.
- *imu/mag:* Muestra los datos de la brujula integrada al FCU.
- *imu/atm_pressure:* Muestra el dato de la presión del aire.

Plugin local_position: Su función es la de publicar los datos de posición local (X, Y, Z) utilizando un marco de coordenadas del tipo NED. Tópicos para suscribirse:

- *local_position/pose:* Obtiene la posición local (X, Y, Z) del FCU.

- *local_position/velocity:* Obtiene los datos de velocidad del FCU.

Plugin setpoint_accel: Su función es la enviar valores de aceleración. Tópico para publicar:

- *setpoint_accel/accel:* Envía un vector de fuerzas o aceleraciones.

Plugin setpoint_attitude: Su función es la de enviar un cuaternión de la orientación deseado utilizando el mensaje MAVLink: *SET_ATTITUDE_TARGET*.
Tópicos para publicar:

- *setpoint_attitude/cmd_vel:* Enviar valores de velocidad angular
- *setpoint_attitude/attitude:* Enviar cuaternión de la orientación (X, Y, Z, w).
- *setpoint_attitude/att_throttle:* Se configura el valor entre 0 y 1.

Plugin setpoint_position: Su función es la de enviar los valores de posición local deseados en un marco de coordenadas en metros, utilizando el mensaje MAVLink:
SET_POSITION_TARGET_LOCAL_NED. Tópico para publicar:

- *setpoint_position/local:* Enviar la posición local deseada X, Y, Z en NED.

Plugin setpoint_velocity: Su función es la de enviar el valor de la velocidad deseada. Tópico para publicar:

- *setpoint_velocity/cmd_vel:* Enviar velocidad deseada.

Plugin sys_status: Su función es la de informar el estado del sistema. Tópicos para suscribirse:

- */state:* Informar el estado del fcu.
- */battery:* Reporta el estado de la batería.
- */extended_state:* Detector de aterrizaje y estado del VTOL.
- */actuator_control:* Envía comandos a los actuadores.

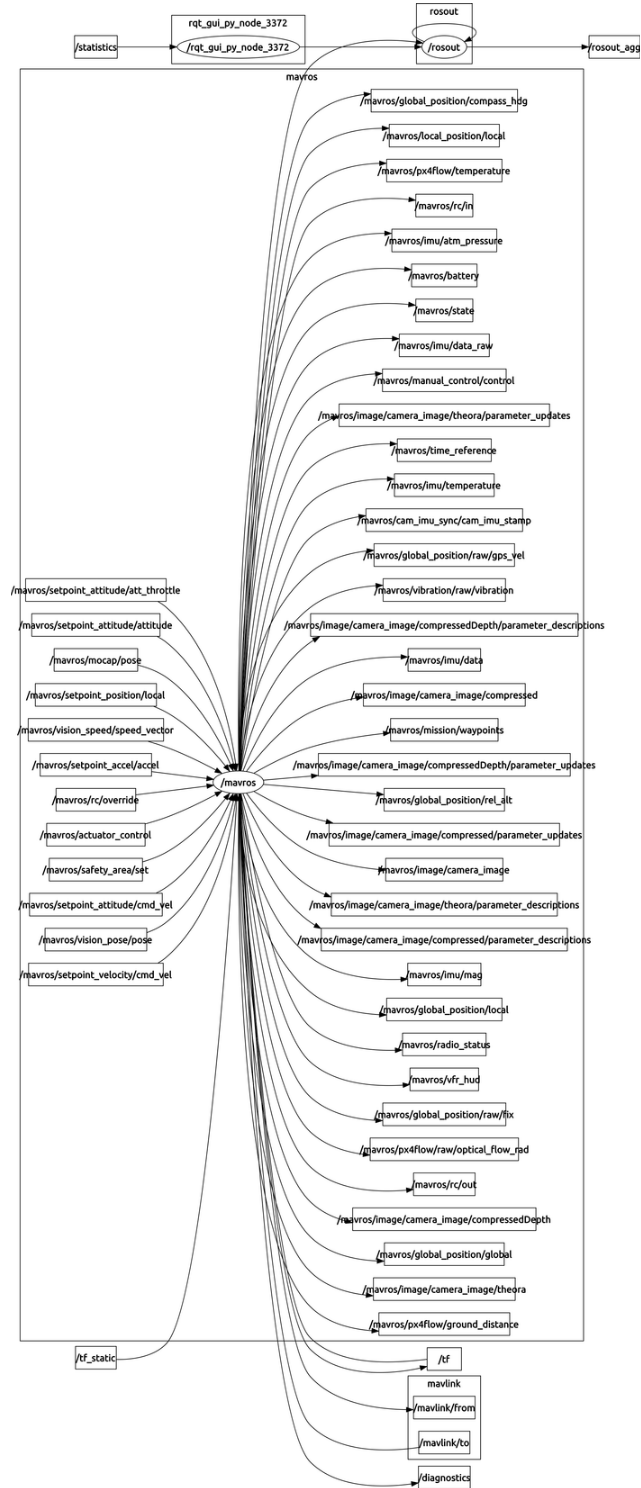

Figura 3-8: Nodo y tópicos MAVROS

3.3. Transmisión de datos de sistema Optitrack a Pixhawk

3.3.1. Esquema de transmisión de datos Optitrack a Pixhawk

Las configuraciones más aptas para la transmisión de datos tipo MOCAP, son las que se muestran en las figuras **3-4** y **3-6**. En la configuración de **radio serial** su ventaja principal radica en que no se necesita una computadora compañera montada en el cuadrirrotor debido a que una PC la sustituye.

Para transmitir los datos MOCAP es necesario configurar el puerto **TELEM2** como entrada de datos externos, ese puerto se configura a una velocidad de transmisión de 57600 o 921600 baudios. En el Pixhawk dentro del módulo LPE, el programa *mocap.cpp* [4] establece que para haya un correcto funcionamiento del cuadrirrotor utilizando datos MOCAP el tiempo máximo de espera de cada dato MOCAP es de **0.2s**, por lo que si no se cumple ese requisito el Pixhawk mandará a la estación de tierra un mensaje de *TIME OUT* y como consecuencia el Pixhawk se reiniciará constantemente.

La desventaja de la configuración de radio serial radica en que los radios Telemetria 3DR/SiK sólo alcanzá una velocidad de transmisión de 57600 baudios y eso tiene como consecuencia la aparición del mensaje TIMEOUT en la estación de tierra, por lo que la mejor configuración es la de usar una computadora montada con conexión WiFi a ROS (figura **3-6**).

La computadora de compañía que se selecciono fue una **Raspberry Pi3**, esta computadora esta montada arriba del 3DR Iris conectada de manera serial al puerto TELEM2 a una velocidad de **921600**, el esquema de conexión se muestra en la figura **3-9**.

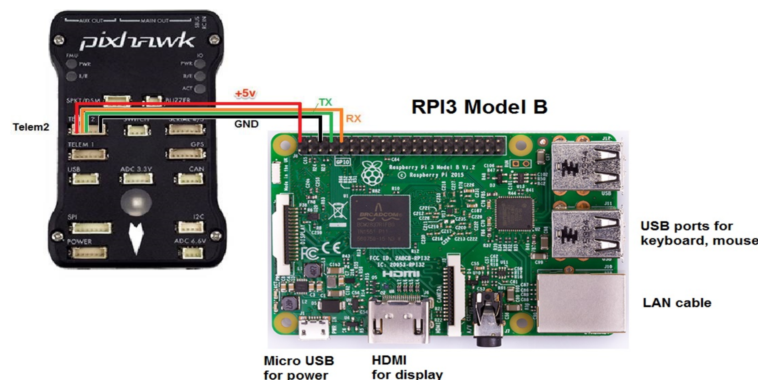

Figura 3-9: Conexión Raspberry Pi3-Pixhawk

Se seleccionó la computadora Raspberry Pi 3 debido al buen rendimiento del procesador y al buen comportamiento con la temperatura además de que esta placa trae integrado un módulo WiFi, por lo que la transmisión de los datos Optitrack se realiza por medio de este módulo.
El esquema final con los componentes que se utilizan para la transmisión de datos Optitrack se muestra en la figura **3-10** y se describen a continuación::

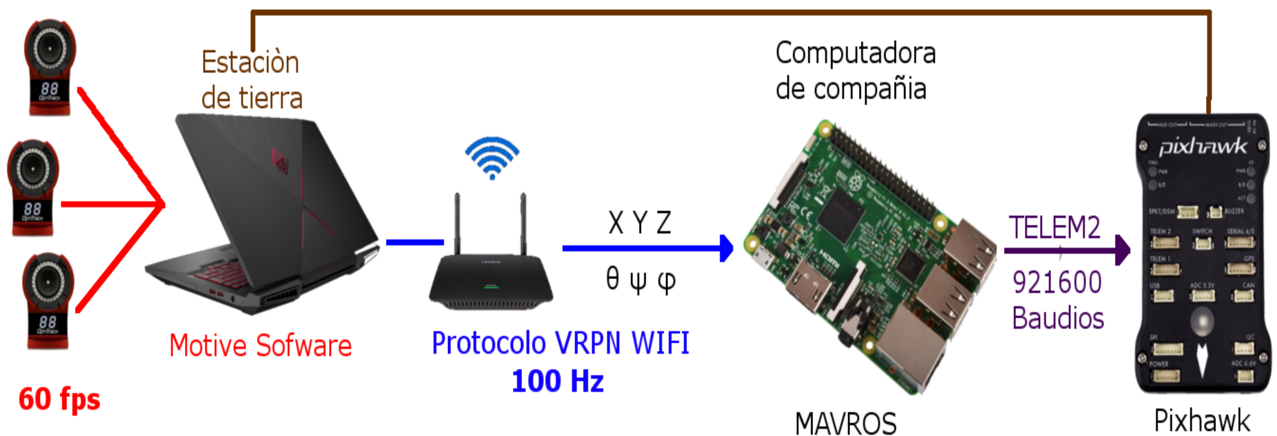

Figura 3-10: Conexión Raspberry Pi3-Pixhawk

Donde:

- **Estación de tierra:** Computadora con Windows con el Software Motive de Optitrack enviando los datos de posición, controla remotamente a la computadora de compañía por medio de WiFi utilizando ROS y QGC.
- **Computadora de compañía:** Computadora Raspberry Pi 3, Utiliza ROS y MAVROS para recibir los datos de posición y los envía a Pixhawk.
- **Pixhawk**: Recibe los datos de posición de manera serial de la computadora de compañía, envía información sobre estado del sistema a la estación de tierra.

3.3.2. Configuración Motive Software y nodo vrpn_client_ros

Motive es el software de captura de movimiento de Optitrack, sólo disponible para Windows. Al tener dos computadoras con diferentes sistema operativo, se tiene que utilizar un protocolo de comunicación para transmitir los datos del programa Motive Software a ROS. Hay dos

tipos de protocolo que el software Motive utiliza, el primero es la transmisión de datos multicast y el segundo es por VRPN (Red Periférica Virtual).
ROS Kinetic posee dos paquetes que utilizan los protocolos mencionados, un paquete llamado *mocap_optitrack* y el otro *vrpn_client_ros*. Como se observa en la figura **3-10**, en el protocolo de comunicación VRPN, las cámaras están configuradas a 60 fps. Al utilizar el paquete vrpn_client_ros, en el archivo configuración se especifica una frecuencia de actualización de 100 Hz y una IP de una red de tipo LAN.
El código para ejecutar:

```
$ . ~/catkin_ws/devel/setup.bash
$roslaunch vrpn_client_ros sample.launch server:="192.168.0.168"
```

3.3.3. Nodo MAVROS y transformación ENU→NED

Los datos de posición y orientación provenientes del nodo VRPN se publicaron en el tópico ***/mavros/mocap/pose*** del nodo MAVROS. Los datos de las cámaras Optitrack vienen en coordenadas ENU y el Pixhawk emplea coordenadas NED, por lo que este tópico es el encargado de convertir las coordenadas de ENU→NED. Para publicar los datos del nodo VRPN al nodo MAVROS, primero se inicia MAVROS.
El código para ejecutar:

```
$source /opt/ros/kinetic/setup.bash
$rosrun mavros mavros_node fcu url:=/dev/ttyAMA0:921600
```

ROS usa el marco de coordenadas ENU, mientras que el firmware PX usa el marco de coordenadas NED (ver figura **3-11**), por lo que es necesario aplicar una transformación de coordenadas.

Figura 3-11: Coordenadas ENU y NED

Por lo tanto, en el software Motive se tiene que definir la coordenada ENU ver figura (**3-12**), el Pixhawk tiene el marco NED como en la figura **3-13**:

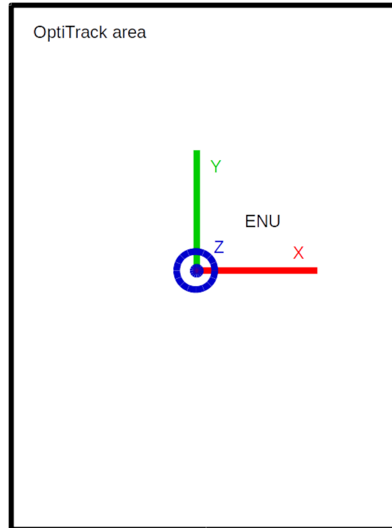

Figura 3-12: Área de Optitrack en coordenadas ENU.

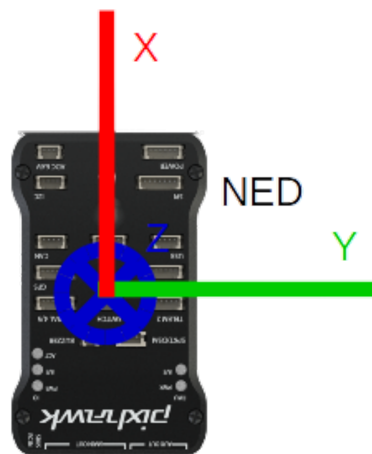

Figura 3-13: Área de coordenadas del Pixhawk en NED.

Para realizar una transformación es necesario la correcta inicialización del cuerpo rígido dentro de Optitrack. Dado que un cuerpo rígido está compuesto por múltiples marcadores, el software no puede conocer la orientación absoluta del Pixhawk con respecto al marco

Optitrack. Cuando se crea un cuerpo rígido, se define su orientación actual como una rotación de unidad $(1, 0, 0, 0)$ cuaternión, por lo que depende de cómo se inicie el Pixhawk dentro del área de Optitrack.
Así al usar MAVROS se debe de alinear el eje x del Pixhawk con el eje x de Optitrack (figura **3-14**).

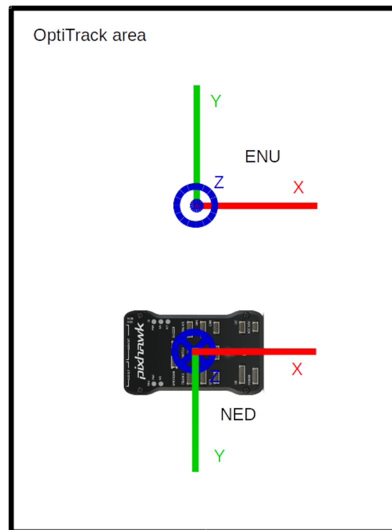

Figura 3-14: Área de coordenadas del Pixhawk en NED.

La conversión de coordenadas es una rotación de 180 grados alrededor del eje x para alinear ambos marcos y esto es lo que sucede dentro de la función MAVROS *transform_frame_enu_ned*. Al realizar esta transformación, los datos de posición del sistema Optitrack son transmitidos por el nodo *vrpn_client_ros* publicando directamente a MAVROS, utilizando el tópico *mocap_pose_estimate*.

Por último, para hacer la conexión entre los dos nodos se utilizó una herramienta de ROS Kinetic llamada topic tools relay, que básicamente consiste en un nodo que sirve como enlace a dos nodos que intercambian publicaciones de datos. Código para iniciar topic tools relay:

```
$source /opt/ros/kinetic/setup.bash
$. ~/catkin_ws/devel/setup.bash
$rosrun topic_tools relay /vrpn_client_node/Quadcopter/pose mavros/mocap/pose
```

3.4. Nodo de generación de trayectorias

Es un nodo donde sus funciones consisten en entrar al modo no abordo, armar, desarmar los motores cada cierto tiempo y de enviar puntos de trayectorias. Las principales funciones de MAVROS que se involucran son:

Modo no abordo:

```
mavros_msgs::SetMode offb_set_mode;
```

Armado de motores:

```
mavros_msgs::CommandBool arm_cmd;
```

Envío de puntos de trayectorias:

```
ros::Publisher local_pos_pub = nh.advertise<geometry_msgs::PoseStamped

("mavros/setpoint_position/local", 10);
```

Se utilizó el simulador Gazebo probar si los puntos de trayectoria programados son correctos. Gazebo [5] es un poderoso entorno de simulación 3D que es particularmente adecuado para probar la evasión de objetos y la visión por computadora. También se puede usar para simulación de vehículos múltiples y se usa comúnmente con ROS, una colección de herramientas para automatizar el control del vehículo.
Vehículos compatibles: Quad (Iris y Solo), Hex (Typhoon H480), Quad delta genérico VTOL, Tailsitter, Ala fija, Rover etc.
Para utilizar Gazebo es necesario tener instalado el código fuente y el firmware PX4. Al tener los dos requisitos en una terminal Linux se ejecuta gazebo dentro de la carpeta del firmware nombrando el cuadrirrotor que se desea simular.
La figura **3-15** muestra una ventana de simulación de vuelo del cuadrirrotor 3DR Iris+ en Gazebo. Las instrucciones para realizar la simulación y ejecutar un nodo generador de trayectorias son las siguientes:

```
$: cd src/firmware
$firmware: make posix gazebo_iris
$rosrun espacio_trabajoROS nombre_nodo.cpp
```

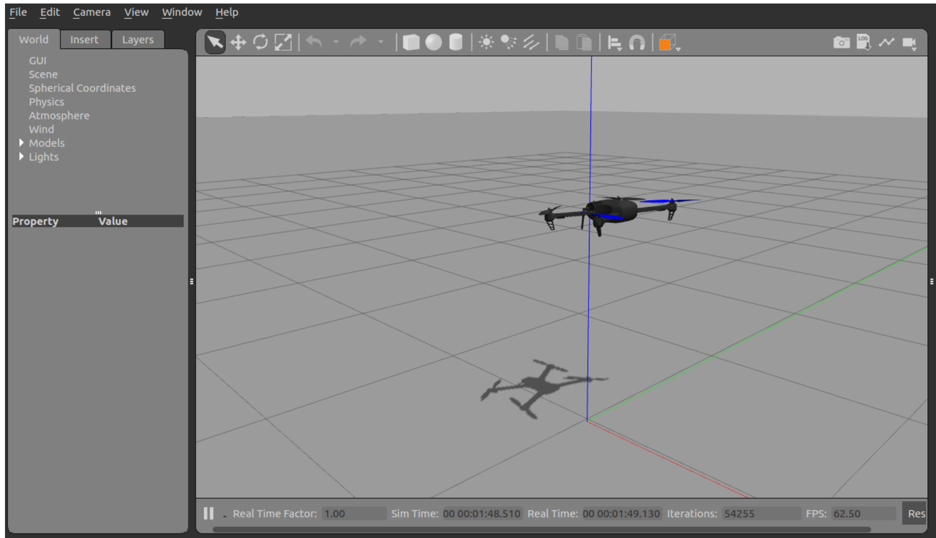

Figura 3-15: Simulación de trayectorias de un 3DR Iris+ en Gazebo.

El diagrama final de nodos para transmitir los datos de posición del sistema Optitrack se muestra en la figura **3-16**.

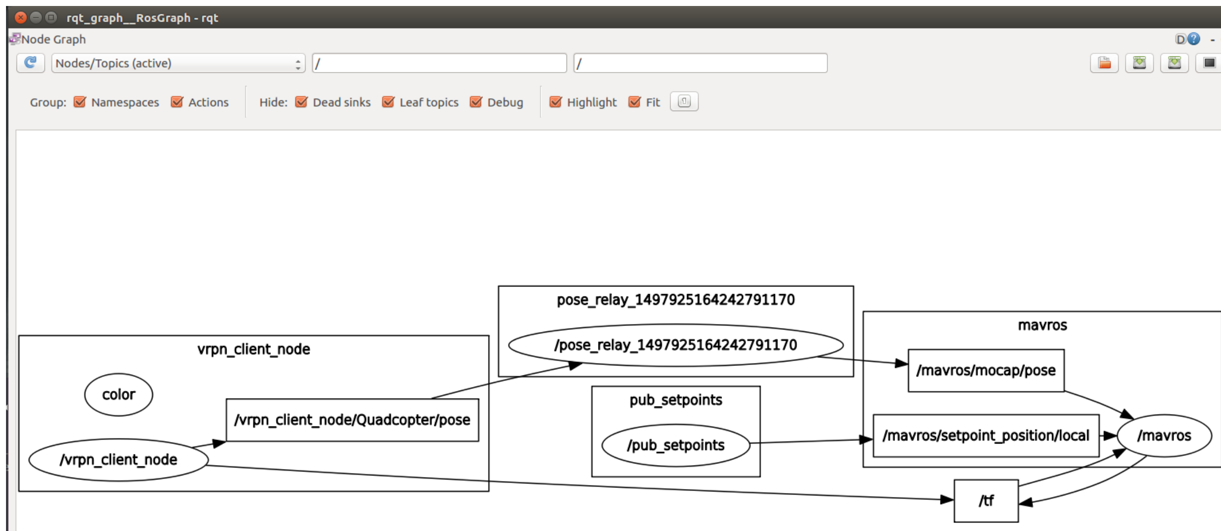

Figura 3-16: Nodos utilizados para transmitir los datos de posición Optitrack.

Las instrucciones de ejecución de nodos para la transmisión de datos Optitrack al Pixhawk:

```
$roslaunch vrpn_client_ros sample.launch server:="192.168.0.168"
$rosrun mavros mavros_node fcu url:=/dev/ttyAMA0:921600
$rosrun topic_tools relay /vrpn_client_node/Quadcopter/pose mavros/mocap/pose
$rosrun quadrirotor offboard_example
```

4 Resultados experimentales

4.1. Resultados Experimentales

4.1.1. Prueba de altitud

En esta primera prueba se mandaron puntos de trayectoria a los ejes x, y y z. El objetivo de esta primera prueba consiste que el cuadrirrotor se mantenga en una altitud de 1m, mientras que la posición x,y se mantenga en 0. En las figuras **4-1** y **4-2** se puede ver la posición mantenida del 3DR Iris+ en 2D. La figura **4-1** muestra una gráfica x contra y, mientras que el la gráfica **4-2** se muestra la altitud desarrollada en el tiempo.

```
msg.pose.position.x = 0.0;
msg.pose.position.y = 0.0;
msg.pose.position.z = 0.5;
msg.pose.orientation.x = 0;
msg.pose.orientation.y = 0;
msg.pose.orientation.z = 0;
msg.pose.orientation.w = 1;
```

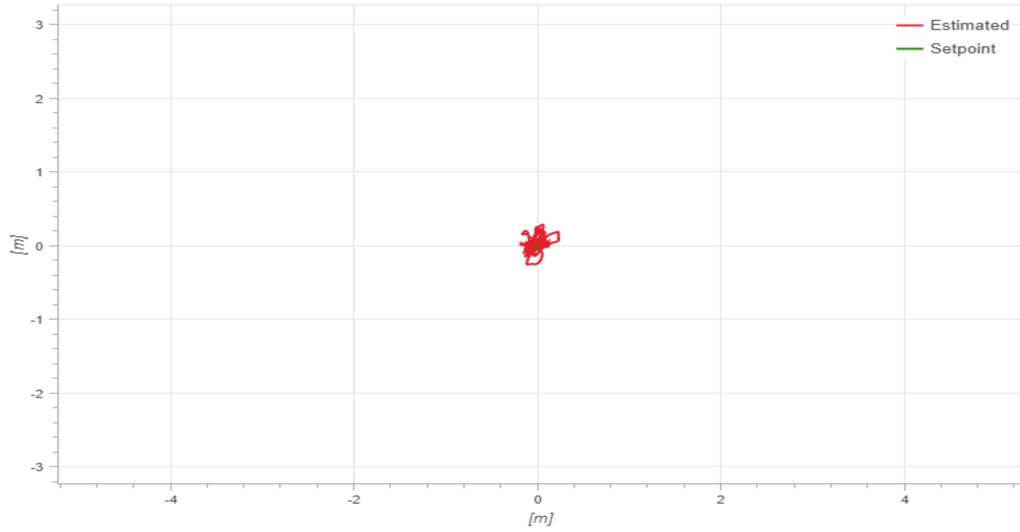

Figura 4-1: Gráfica 2D de la prueba de altitud.

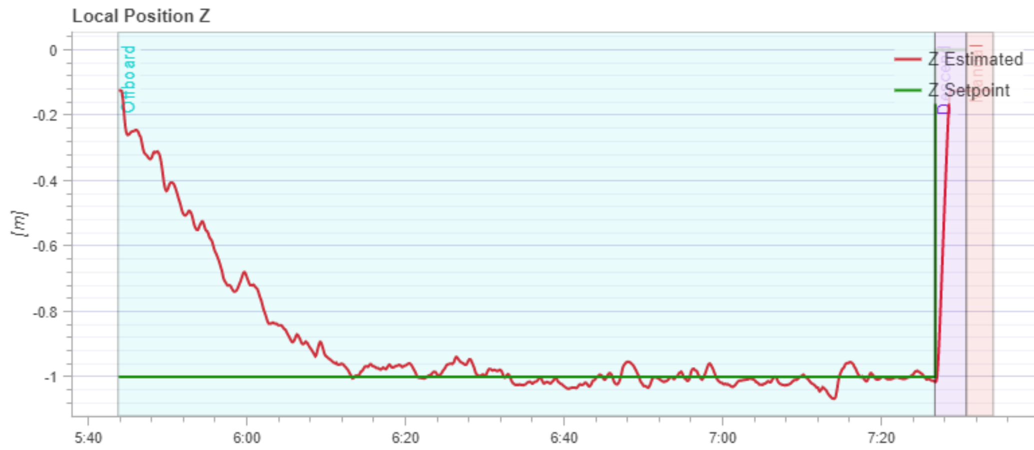

Figura 4-2: Gráfica de posición local del eje z.

Observando las figuras **4-1** y **4-2** podemos apreciar que el cuadrirrotor si se mantiene en un punto estable con un poco margen de error, en la altitud el cuadrirrotor se mantiene en un margen de entre 0.9m y 1.05 m.

4.1.2. Prueba de trayectoria circular

En la segunda prueba que se reporta (figura **4-3**) se mandaron puntos para generar una trayectoria circular con radio es de $r = 1$m y a una altitud de $z = 1m$, Las coordenadas x, y para generar la trayectoria deseada están dadas por las ecuaciones (4-1) y (4-2). La trayectoria recorrida por el dron se muestra en la figura **4-3**. En la figura **4-4** se muestra la gráfica de la altitud en la que se mantuvo el 3DR Iris+.

$$x = r * \sin(\theta) \tag{4-1}$$

$$y = r * \cos(\theta) \tag{4-2}$$

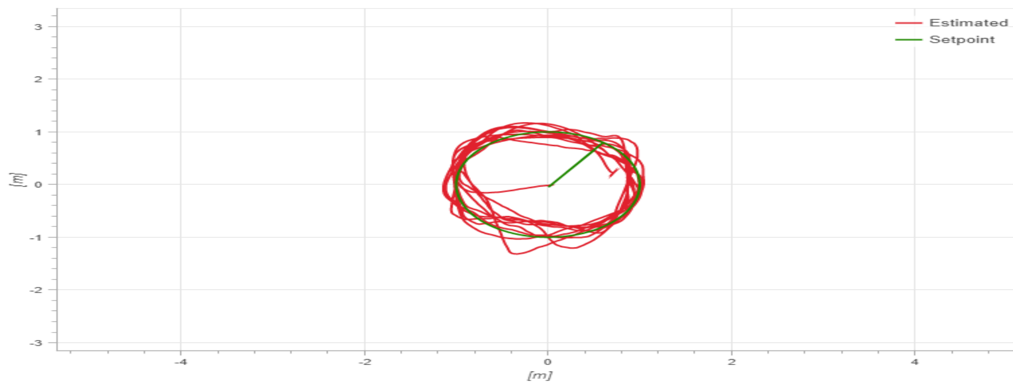

Figura 4-3: Gráfica 2D de la trayectoria circular del 3DR Iris+.

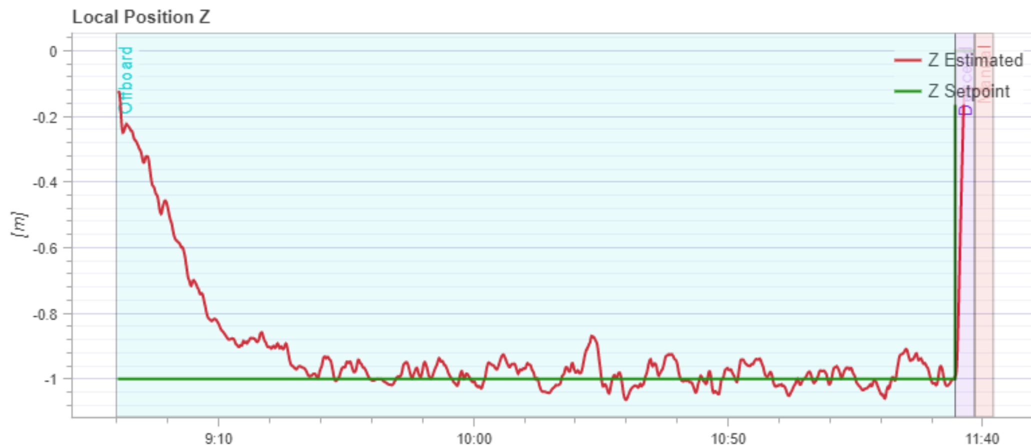

Figura 4-4: Gráfica de posición local del eje z.

4.1.3. Prueba de trayectoria en forma de 8

En la última prueba que se reporta se mandaron puntos para generar una trayectoria en forma de 8. Se estableció un constante de tangente vertical $a = 1$ y a una altitud de $z = 1$m. La trayectoria deseada se genera mediante las ecuaciones (4-3) y (4-4). En la figura **4-5** se muestra la trayectoria recorrida por el vehículo. En la figura **4-6** se muestra la gráfica de la altitud en la que se mantuvo el 3DR Iris+.

$$x = a * \sin(\theta) \tag{4-3}$$

$$y = a \sin(\theta) * \cos(\theta) \tag{4-4}$$

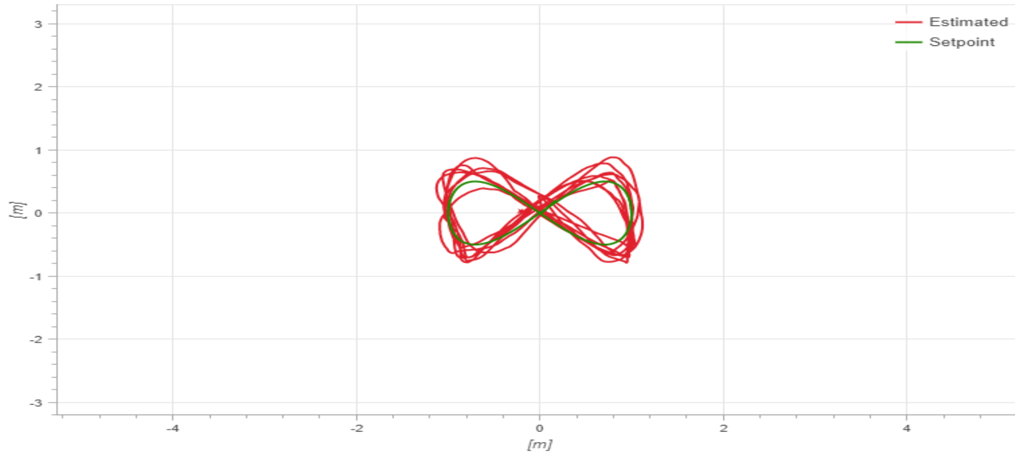

Figura 4-5: Gráfica 2D de la trayectoria en forma de 8 del 3DR Iris+.

Figura 4-6: Gráfica de posición local del eje z.

5 Conclusión

En esta tesis, se demostró la implementación de vuelo autónomo cuadrirrotor 3DR Iris+, utilizando la unidad de vuelo Pixhawk, y obteniendo los datos de posición mediante un sistema de captura de movimiento Optitrack. Se logró utilizar y modificar los módulos, librerías de una plataforma PX4 Autopilot utilizando lenguaje de programación de alto nivel.

Se aplicó el framework ROS aprovechando las ventajas que éste representa en la programación de robots, La principal función de ROS en la transmisión de datos MOCAP consistió en la creación de un nodo de comunicación VRPN para transmitir los datos de posición del software Motive Optitrack a la computadora Linux aplicando las funciones y tópicos.

Para la transmisión de datos del nodo VRPN al Pixhawk se utilizó MAVROS, un paquete de extensión MAVLink que comunica las computadoras ROS con los FCU que tienen el software PX4 Autopilot. Utilizando MAVROS se transmitieron los datos de posición al tópico MOCAP del Pixhawk.

Se implementó una computadora externa (Raspberry PI3) montada en el 3DR Iris+. En la RPi3 se instaló el software ROS, y se logró que esta computadora de manera remota controle externamente el Pixhawk, esta computadora manda datos de posición deseada para que el cuadrirrotor realice trayectorias. La función principal del RPi3 es la de recibir los datos de posición MOCAP de la estación de tierra.

La utilización de ROS y MAVROS simplificó la programación debido a que éstos facilitan el envío y recepción de datos a comparación de MAVLink, debido a la gran variedad de tópicos

creados para cada sensor que compone el sofware Autopilot.

El esquema de control planteado en este trabajo de tesis resultó ser exitoso para implementar trayectorias de vuelo en interiores, en ambientes controlados con captura de movimiento mediante visión computacional. Se logró realizar las trayectorias propuestas, presentando errores menores a 12 cms. Debido a falta de módulos Telemetria a mayor velocidad de comunicación, la opción para que Pixhawk tome los datos medidos por el sistema Optitrack, es mediante una computadora compañera, en este caso RPi3. Se puede lograr mejor desempeño de vuelo si se reduce el tamaño de la computadora compañera o colocar as cámaras a una mayor altura para realizar vuelos a una mayor altitud.

5.1. Trabajo a futuro

Con el estudio de Pixhawk, MAVink y MAVROS se puede modificar cualquier librería y módulos, por lo que para proyectos a futuro se sugiere:

- Realizar diferentes tipos de leyes de control para el control de ángulos o de altitud.
- Utilizando ROS y MAVROS utilizar las cámaras para realizar algoritmos de evasión de obstáculos.
- Implementar nuevos sensores para el Pixhawk como los RTK u otras IMU.
- Utilizando ROS implementar técnicas de mapeo como los algoritmos SLAM.

Bibliografía

[1] Ardupilot: *Copter Attitude Control*. http://ardupilot.org/dev/docs/apmcopter-programming-attitude-control-2.html. – Accessed: 2017-10-11

[2] Autopilot, PX4: *About Pixhawk*. https://pixhawk.org/. – Accessed: 2017-09-27

[3] Autopilot, PX4: *Development Environment on Ubuntu LTS / Debian Linux*. https://dev.px4.io/en/setup/dev_env_linux_ubuntu.html. – Accessed: 2017-09-27

[4] Autopilot, PX4: *PX4 Firmware*. https://github.com/PX4/Firmware/blob/master/src/modules/local_position_estimator/sensors/mocap.cpp. – Accessed: 2017-09-27

[5] development, Gazebo: *Gazebo*. https://dev.px4.io/en/simulation/gazebo.htmlp. – Accessed: 2017-09-27

[6] Lorenz Meier, Marc P.: PX4: A Node-Based Multithreaded Open Source Robotics Framework for Deeply Embedded Platforms. En: *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, p. 6235–6240

[7] NaturalPoint, Inc. DBA O.: *OptiTrack Documentation*. http://v110.wiki.optitrack.com/index.php?title=OptiTrack_Documentation_Wiki. – Accessed: 2017-09-27

[8] QGroundControl, Organización: *MAVLinl*. http://qgroundcontrol.org/mavlink/start. – Accessed: 2017-09-27

[9] ROS. *Ros About*. http://www.ros.org/about-ros/

[10] Team, ArduPilot D.: *Working with the ArduPilot Project Code*. http://ardupilot.org/dev/docs/where-to-get-the-code.html/. – Accessed: 2017-09-27