CAPÍTULO IV - Desarrollo

Para llevar a cabo el desarrollo del presente Trabajo Especial de Grado se partió de un chásis de cuadricóptero Draganflyer V, con su estructura de tubos, cabina central, motores, transmisión y propelas. Se diseñaron circuitos para el control y alimentación de los motores del cuadricóptero, y para servir de interfaz entre la unidad de control Arduino Nano y los sensores y módulos de comunicación utilizados, los cuales son descritos en los apartados I y II.

Se desarrollaron algoritmos para la estimación de posición angular, velocidad angular, y posición y velocidad lineal en el eje z del cuadricóptero (altura y velocidad de elevación). Para más información sobre los algoritmos implementados pueden consultarse los apartados III y IV del presente trabajo.

Se desarrolló una interfaz de comunicación inalámbrica para llevar a cabo comando remoto del cuadricóptero y telemetría, la cual es descrita con detalle en el apartado V.

Se adaptó y linealizó el modelo dinámico del cuadricóptero Draganflyer V desarrollado en [Kivrak 2006], se analizó la estabilidad del sistema lineal obtenido en condiciones de vuelo en posición horizontal, y se simularon exitosamente dos arquitecturas de controladores PID para estabilizar al cuadricóptero en vuelo horizontal y asegurar el seguimiento de consignas de movimiento simple por parte del usuario. Se implementaron las arquitecturas de control desarrolladas, y se realizaron pruebas. El proceso citado anteriormente se expone con más detalle en el apartado VI.

Se desarrolló una plataforma de pruebas, que consta de un software de control y telemetría del cuadricóptero implementado sobre la plataforma Robot Operating System, una serie de scripts para el análisis de datos de los sensores del cuadricóptero en el dominio de la frecuencia, y estrategias para realizar pruebas de vuelo bajo condiciones controladas. La plataforma en cuestión se expone en el apartado VII.

Finalmente, en el apartado VIII se describe una perspectiva general del sistema implementado.

1. ***Circuito de alimentación y control de motores***

Se desarrolló un circuito para llevar a cabo la alimentación y regulación de velocidad de los motores de corriente continua del chasis de cuadricóptero Draganflyer V utilizado en el presente trabajo. En primer lugar, se decidió utilizar una batería de Polímero de Litio (Li-Po, del inglés Lithium Polymer) de 3 celdas, ya que es la configuración de alimentación de los motores de corriente continua recomendada por el fabricante en [Draganfly Innovations 2006].Por otro lado, en [Nadales 2009], se describen las características de funcionamiento de los motores en cuestión, y la relación entre el voltaje de alimentación y la corriente consumida por los mismos al hacer girar las hélices del cuadricóptero. A continuación se detallan las características de los motores citadas previamente:

* Modelo de motores DC: Mabuchi RC 280SA.
* Voltaje de alimentación recomendado: 4,5 a 12 Voltios.
* Consumo sin carga: 0,14 Amperios.
* Consumo medio: 4 Amperios.
* Peso individual: 47 Gramos.



Ilustracion 4. Relación voltaje-corriente de los motores DC del chasis Draganflyer V.

Fuente: [Nadales 2009]

Tomando en cuenta las características de los motores anteriormente expuestas, se puede estimar un consumo promedio de 16 Amperios al alimentar los cuatro motores de forma continua, y de hasta 24 amperios considerando un pico de consumo de hasta 2 amperios por motor al realizar el encendido de los mismos, los cuales deben ser provistos por la batería de polímero de litio de forma instantánea. Para una batería de polímero de litio cualquiera, se tiene que:

Para satisfacer la necesidad de consumo de los motores del cuadricóptero se utilizó una batería Li-Po Yuntong de 3 celdas, con una capacidad de 1250 miliAmperios por Hora (mAH), y una tasa de descarga de 25C. Para la batería seleccionada se estimaron los siguientes valores de consumo y duración, asumiendo un ciclo de trabajo máximo por parte de los motores:

Para regular la velocidad de los motores de corriente continua del cuadricóptero, se utilizó Modulación por Ancho de Pulso (PWM, del inglés Pulse Width Modulation), enviada desde los puertos de la tarjeta Arduino Nano 3.0, para realizar una conmutación rápida entre los estados de encendido y apagado de los motores, pudiendo mantener un control de lazo abierto sobre su velocidad a partir del ciclo de trabajo del PWM emitido.

****

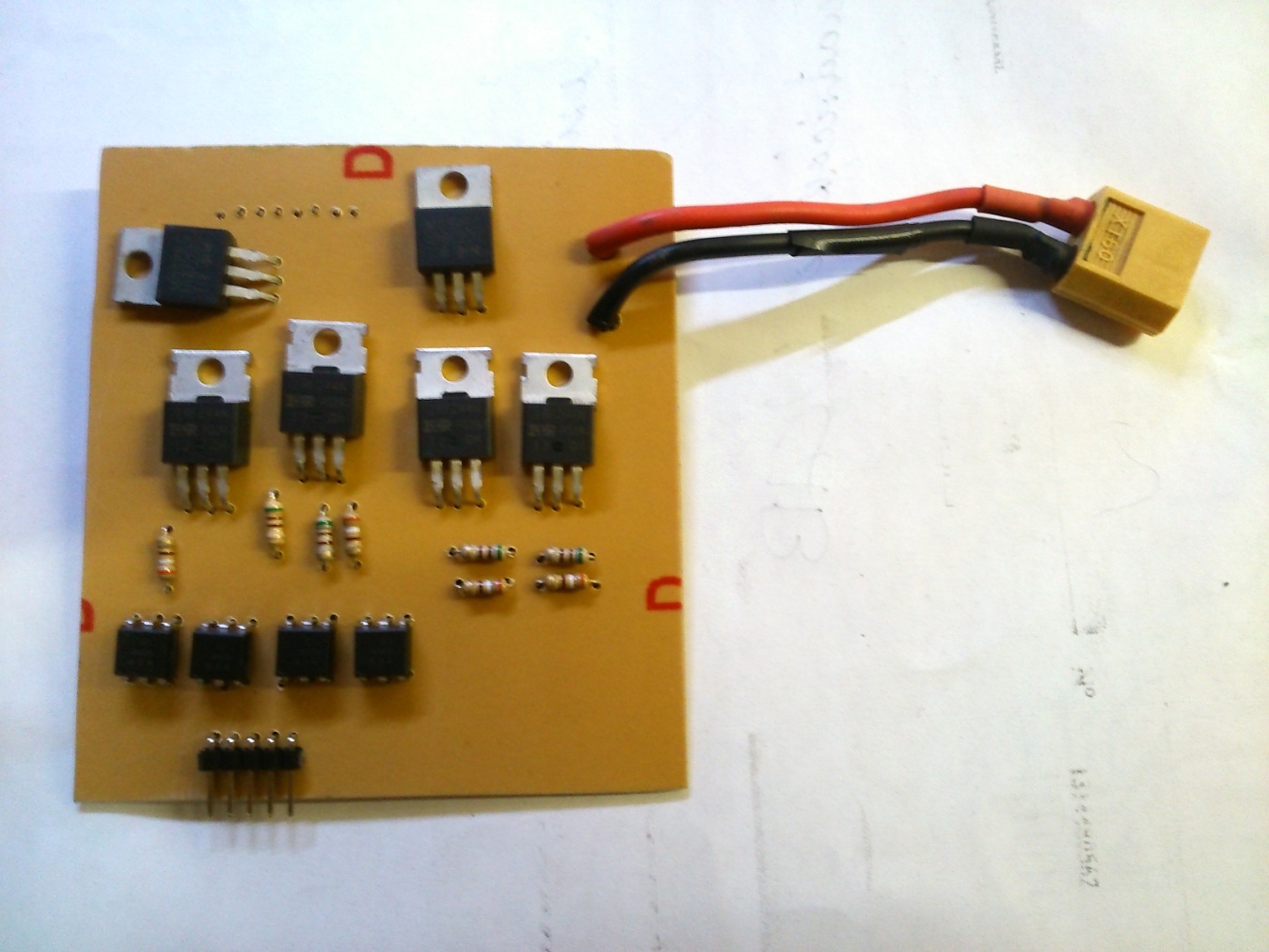
Ilustracion 4. Diagrama de bloques del sistema de control de velocidad de los motores DC del cuadricóptero.

Fuente: elaboración propia.

El circuito de control de motores se desarrolló en dos fases:

1. **Primera fase:** se elaboró un prototipo simple con componentes existentes en el mercado venezolano, asumiendo un comportamiento ideal de los mismos en conjunto. Se detectó una alta tasa de disipación, y con ello, una reducción en la velocidad de los motores del cuadricóptero en funcionamiento.
2. **Segunda fase (final):** para disminuir la disipación presentada en el primer prototipo, se caracterizó la respuesta de los mismos ante las cargas de los motores del cuadricóptero, y se modeló el proceso de descarga de la batería para asegurar disipación mínima en los componentes del circuito, y un máximo consumo de corriente por parte de los motores del cuadricóptero. El circuito desarrollado tuvo un rendimiento satisfactorio en pruebas, y fue utilizado para llevar a cabo las pruebas de vuelo del cuadricóptero en el presente trabajo.

A continuación se expone el circuito impreso construido durante la segunda fase de desarrollo:



Ilustracion 4. Placa del circuito de alimentación y control de motores.

Fuente: elaboración propia.

. Para más detalles acerca del proceso de desarrollo del prototipo final del circuito de control de motores, consúltese el Apéndice A del presente trabajo.

1. ***Circuito de lógica, sensores y comunicación***

Se diseñó una placa de circuito que consta de una serie de puertos pin header para situar la tarjeta Arduino y la unidad de medición inercial, además de conectar el sensor de ultrasonido, las salidas de PWM y el módulo de XBEE mediante cables “jumper”. La misma sirve de interfaz de comunicación entre la tarjeta Arduino, y los demás módulos del cuadricóptero. Además, permite la alimentación de todo el circuito haciendo uso de la etapa de regulación de voltaje embebida en la tarjeta Arduino, la cual puede ser alimentada con voltajes de entre 6 y 20 voltios. A continuación se detalla el consumo promedio de cada componente del circuito, y el consumo total del mismo:

Tabla 4. Desglose del consumo promedio por componente, y acumulado, del circuito de lógica, sensores y comunicación.

|  |  |
| --- | --- |
| **Componente** | **Consumo promedio** |
| Arduino Nano | 50 mA |
| Pololu minIMU-9 V2 | 10mA |
| XBee Series 1mW Wire Antenna – Series 1 | 50mA |
| Parallax Ping | 30mA |
| 4 Optocouplers 4N26 – Módulo de control de motores | 240 mA |
| **Total** | **380mA** |

Fuente: elaboración propia.

Para alimentación del módulo de lógica, sensores y comunicación durante la realización de pruebas de vuelo se utilizó una batería de polímero de litio e-Flite de 300mAH, con un coeficiente de descarga de 3C. A continuación se presenta el cálculo de la máxima capacidad de descarga, y duración de la carga de la misma al alimentar el circuito desarrollado:

En la ilustración que sigue se presenta una fotografía del circuito desarrollado:



Ilustracion 4. Placa del circuito de lógica, sensores y comunicación.

Fuente: elaboración propia.

1. ***Estimación de orientación del cuadricóptero***
   1. **Convenciones respecto a los ángulos**

Para simplificar la implementación de los sistemas de control del cuadricóptero, se decidió representar la posición y velocidad angular del mismo en cada eje mediante ángulos de Euler, bajo la siguiente convención:

* Ángulo de Yaw: ángulo de giro respecto al eje Z del acelerómetro y giroscopio.
* Ángulo de Pitch: ángulo de giro respecto al eje Y del acelerómetro y giroscopio.
* Ángulo de Roll: ángulo de giro respecto al eje X del acelerómetro y giroscopio.
* Velocidad de Yaw: velocidad de giro respecto al eje Z del giroscopio.
* Velocidad de Pitch: velocidad de giro respecto al eje Y del giroscopio.
* Velocidad de Roll: velocidad de giro respecto al eje X del giroscopio.
  1. **Descripción de la Unidad de Medición Inercial (IMU)**

Se utilizó una tarjeta Pololu MinIMU-9 v2, la cual está constituida porun giroscopio de tres (3) ejes L3GD20, y por un acelerómetro de tres (3) ejes y un compás de tres (3) ejes LSM303DLHC. La misma provee de una interfaz que permite acceder a las mediciones, por eje, de cada uno de sus sensores, e incluye un regulador de voltaje y un convertidor de nivel de voltaje que permite su operación con una entrada de voltaje de 2,5V a 5,5V.



Ilustracion 4. Pololu minIMU-9 V2.

Fuente: <http://www.pololu.com/product/1268>

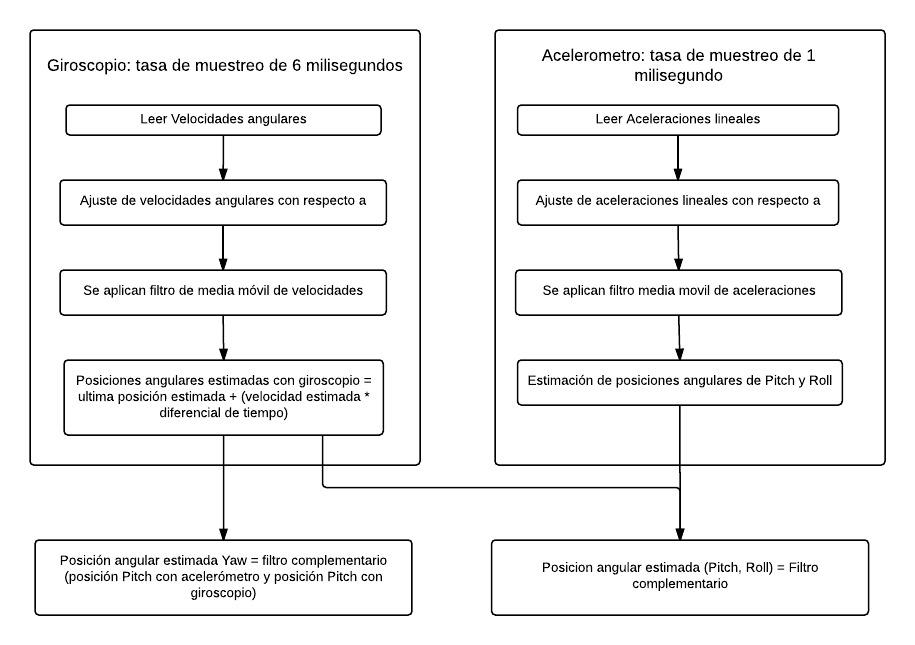
Para la obtención de datos de la tarjeta Pololu MinIMU-9 v2 se utilizaron las librerías diseñadas por el fabricante para su manejo desde tarjetas Arduino.

La estimación de estimación angular del cuadricóptero fue realizada siguiendo el procedimiento descrito en el *Apéndice B*.

* 1. **Algoritmo de estimación de posición angular y velocidad angular del cuadricóptero**

Para estimar la posición angular y velocidad angular del cuadricóptero se desarrolló un modelo matemático del problema, en base a las características de los sensores acelerómetro y giroscopio. El modelo en cuestión se encuentra en el *Apéndice B* del presente trabajo.

A continuación se describe el algoritmo de estimación de posición angular y velocidad angular desarrollado:

****

Ilustracion 4. 16: funcionamiento del algoritmo de estimación de posición y velocidad angular

Fuente: <http://www.parallax.com/product/28015>

1. **Estimación de posición y velocidad lineal en el eje Z del cuadricóptero**
   1. **Descripción del sensor de altura utilizado**

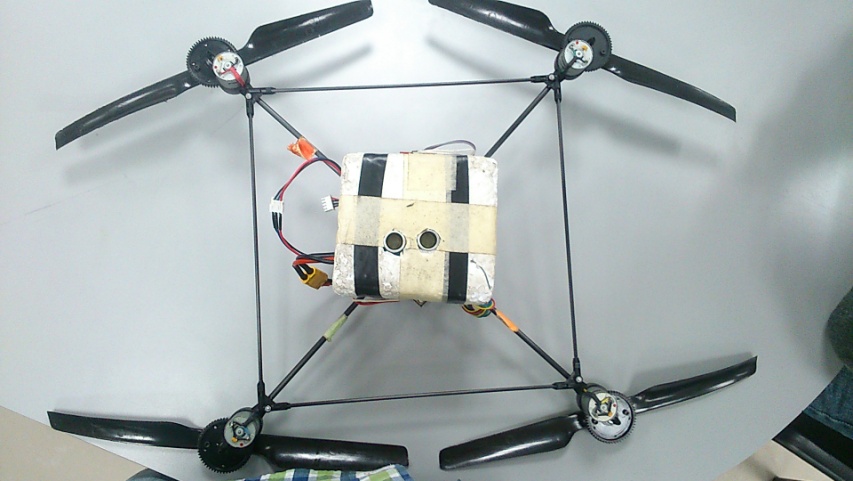
Para la estimación de altura del cuadricóptero se utilizó un sensor Parallax Ping, el cual opera con niveles lógicos TTL, provee un rango de alcance de entre 2cm y 3m, y un peso de 9 gramos. Adicionalmente, el sensor porta un LED en su cara frontal, que indica si el proceso de estimación de altura se está realizando de forma efectiva.



Ilustracion 4. Sensor ultrasónico Parallax Ping.

Fuente: <http://www.parallax.com/product/28015>

El sensor ultrasónico de distancia se ubicó en la parte inferior del cuadricóptero, debajo de las baterías, y dentro de una base de anime esculpida para amortiguar el aterrizaje del cuadricóptero y sostener al sensor ultrasónico en una posición fija.



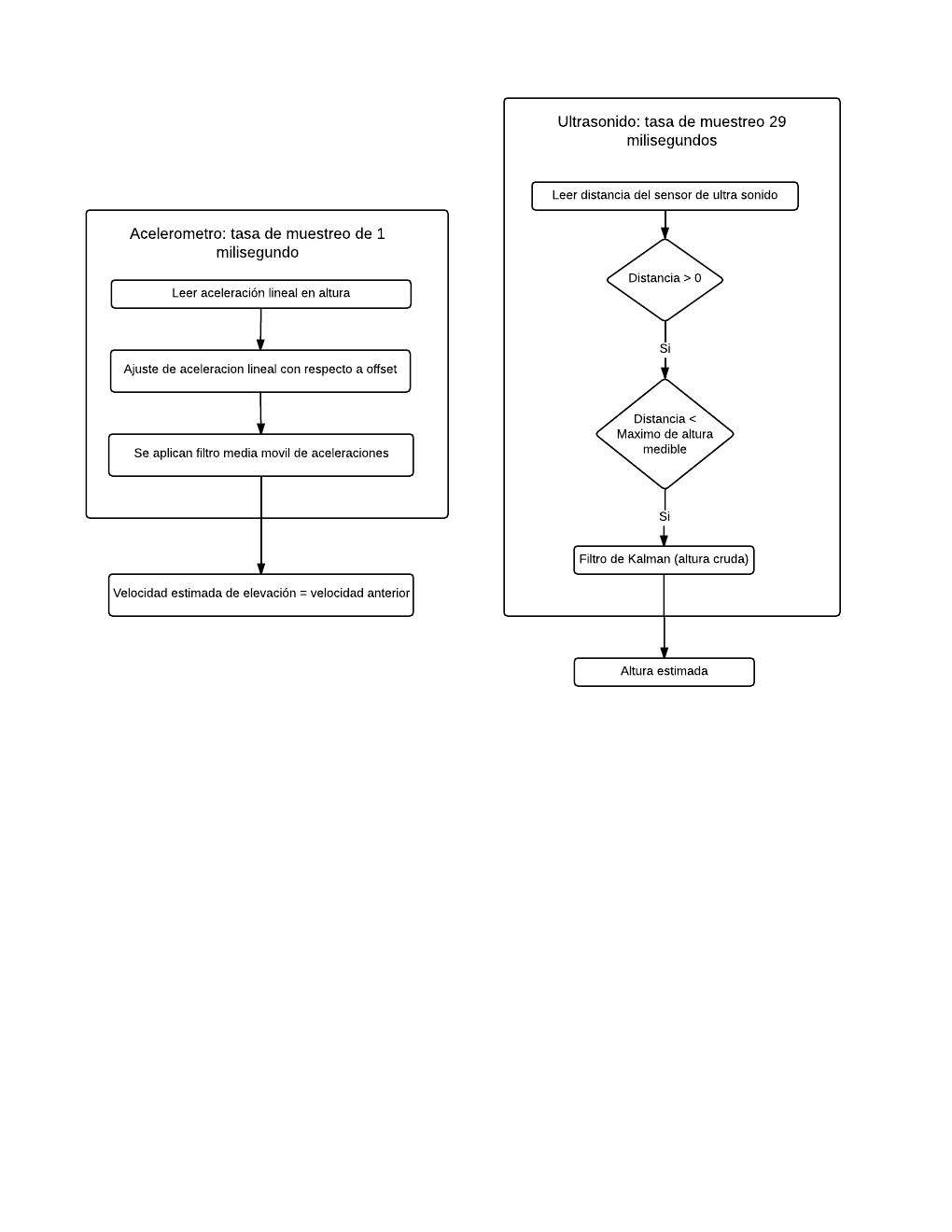
Ilustracion 4. : posición del sensor ultrasónico de distancia en el cuadricóptero.

Fuente: elaboración propia.

* 1. **Algoritmo de estimación de posición y velocidad en el eje z del cuadricóptero**

Se desarrolló un modelo matemático del proceso de estimación a partir de las características de los sensores utilizados. El mismo puede encontrarse en el *Apéndice C* del presente trabajo.

A continuación se describe el algoritmo desarrollado para la estimación de posición y velocidad en el eje z del cuadricóptero:

******

Ilustracion 4. : funcionamiento del algoritmo de estimación de posición y velocidad lineal en el eje z.

Fuente: elaboración propia.

1. ***Comunicación inalámbrica***

Para poder mover el cuadricóptero y conocer los datos de telemetría desde una computadora fue necesario definir una comunicación entre ambos. Esta comunicación debe ser ligera, rápida, sencilla, de bajo consumo y compatible tanto para la PC como para la tarjeta arduino que se instaló en el cuadricóptero, por esta razón se eligió usar el protocolo ZigBee.

* 1. ***Descripción de las características y configuración de los módulos XBEE utilizados***

Se utilizaron dos (2) tarjetas XBee 1mW Series 1 con antena de cable, una tarjeta XBee Explorer USB y una tarjeta XBee Explorer Regulated.



Ilustracion 4. Modulo XBee serie 1

Fuente: <http://examples.digi.com/wp-content/uploads/2012/06/1x-XBee-S1-300x300.jpg>

Especificaciones técnicas de los modulos XBee:

Tabla 4. Especificaciones del modulo XBee

|  |  |
| --- | --- |
| **Caracteristicas** | **ModuloXbee** |
| **Rendimiento** | |
| Rango con obstáculos | Hasta 100 pies (30 mts) |
| Rango sin obstaculos | Hasta 300 pies (90 mts) |
| Potencia de salida transmitida (seleccionable por software) | 1mW (0 dBm) |
| Tasa de datos en radio frecuencia | 250,000 bps |
| Frecuencia de datos por interfaz serial (seleccionable por software) | 1200 bps - 250 kbps (frecuencias sin estandar tambien son soportadas) |
| Sensibilidad de recepcion | -92 dBm (1% tasa de error de 1%) |
| **Requerimientos de poder** | |
| Voltaje de Entrada | 2.8 – 3.4 V |
| Corriente para enviar data (típica) | 45mA (@ 3.3 V) |
| Corriente para recibir data (típica) | 50mA (@ 3.3 V) |
| Corriente para apagarse | < 10 µA |
| **General** | |
| Frecuencia de operación | ISM 2.4 GHz |
| Dimensiones | 0,96” x 1,087” (2,438cm x 2,761cm) |
| Temperatura de operación | -40 to 85º C (industrial) |
| Opciones de antena | Integrated Whip, Chip or U.FL Connector, RPSMA Connector |
| **Redes y Seguridad** | |
| Topologias de red soportadas | Point-to-point, Point-to-multipoint & Peer-to-peer |
| Cantidad de canales (seleccionable por software) | 16 Direct Sequence Channels |
| Opciones de direccionamiento | PAN ID, Channel and Addresses |

Fuente: <https://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Datasheet.pdf>

Las tarjetas Sparkfun XBee *Explorer USB* y Xbee *Explorer Regulated* permiten usar las tarjetas de XBee como una interfaz serial, por ende, facilitan la programación para la comunicacion entre la PC y el cuadricóptero



 Ilustracion 4.  *XBee Explorer Regulated*

Fuente: <https://cdn.sparkfun.com//assets/parts/7/1/1/5/11373-02.jpg>

Ilustracion 4. *XBee Explorer USB*

Fuente: <https://cdn.sparkfun.com//assets/parts/8/1/4/0/11812-03.jpg>

* 1. ***Descripción del protocolo de comunicación desarrollado***

Por razones de velocidad y simplicidad en la implementación se utilizaron los modulos XBEE en modo AT (transparent mode), el cual permite diseñar los paquetes de comunicación pudiendo así reducir su longitud al enviar datos en forma serial y diseñar todo el protocolo de comunicación aumentando la velocidad de transmisión de los mensajes.

Se ha identificado un error de comunicación entre el Arduino Nano y los módulos XBee Series 1 cuando se configuran ambos para transmitir a 57600 baudios por segundo, que roza el máximo de tolerancia soportado por la UART (“***U****niversal****A****synchronous****R****eceiver-****T****ransmitter*” ,en español, *Transmisor-Receptor Asíncrono Universal)* [Colton 2011 - 1]. Con base en este hecho, se decidió utilizar una frecuencia de transmisión de 38400 baudios por segundo para la realización de pruebas de vuelo, y de 115200 baudios por segundo para la realización de telemetría con fines de análisis de datos de los sensores del cuadricóptero en reposo.

Para reducir al mínimo el tamaño de los paquetes a enviar, se decidió codificar los datos a transmitirse por las interfaces inalámbricas en bytes, lo cual limitó el rango de valores a enviar mediante la interfaz diseñada, pudiendo enviarse datos de tipo entero dentro del rango numérico de cero (0) a doscientos cincuenta y cinco (255). Para los casos en que se necesitaba un mayor rango, se decidió dividir los paquetes según su signo, pudiéndose enviar, finalmente, datos dentro del rango numérico de los enteros, de cero (0) a quinientos doce (512).

Para los mensajes se desarrolló la siguiente arquitectura:

Tabla 4. Arquitectura de los mensajes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Cantidad bytes** | 1 | 1 | 1-12 | 1 |
| **Contenido** | Cabecera del mensaje (255) | Código del mensaje | Contenido del mensaje | Checksum |

*Fuente: elaboración propia*

**La cabecera de mensaje:** es un identificador que indica cuando inicia un mensaje, siempre será el valor 255.

**El código de mensaje:** es el valor con el que se identifica cada tipo de mensaje.

**Contenido del mensaje:** es el que contiene la data útil dependiendo del mensaje, su longitud varia de 1 a 12 bytes dependiendo del tipo de mensaje.

**Checksum:** se utiliza a manera de validar que el mensaje llegue correctamente, se genera haciendo una operación XOR a todos los bytes anteriores de ese mensaje.

Para este TEG se diseñaron cuatro (4) tipos de mensajes diferentes, dos (2) mensajes que van del cuadricóptero a la PC y dos (2) mensajes que van desde la PC al cuadricóptero y son los siguientes:

**Mensaje de estado:** es el paquete que contiene la información de telemetría del cuadricóptero y su contenido está compuesto por 12 bytes, los cuales indican posiciones y velocidades para los tres (3) ejes del cuadricóptero con respecto a un eje fijo imaginario, además tiene información sobre la altura y si los motores se encuentran encendidos o apagados.

Debido a que tanto la posición como la velocidad pueden ser negativas o positivas y el mayor valor que se puede enviar en un byte es doscientos cincuenta y cinco (255) se decidió separar los datos cuyo rango de valores superara esta cota máxima.



Ilustracion 4. Diagrama de movimientos del cuadricóptero y signo que toman los valores de aceleración y posición.

Fuente: <http://3.bp.blogspot.com/-B8IImJJHUl0/U0cj-KWq95I/AAAAAAAAAIg/kaMEopdgCwc/s1600/Screen+Shot+2014-04-10+at+4.06.09+PM.png>

* Para el eje yaw la velocidad angular puede variar de -255 a 255 y la posición puede variar entre -180 y 180 grados
* Para el eje pitch la velocidad angular puede variar de -255 a 255 y la posición puede variar entre -90 y 90 grados.
* Para el eje roll la velocidad angular puede variar de -255 a 255 y la posición puede variar entre -90 y 90 grados.
* La altura siempre será un valor positivo debido a que representa la distancia entre el cuadricóptero y el piso, su máximo valor es de dos metros y medio (2,5 metros).
* El estado de los motores indica si estos están encendidos (0) o apagados (1).

**Mensaje de recibido:** es el mensaje que envía el cuadricóptero a la pc para indicar que se ha recibido un mensaje y lo que contiene es el código del mensaje recibido.

**Mensaje de motores:** es el comando que envía la PC al cuadricóptero para apagar o encender los motores.

**Mensaje de movimiento:** es el comando que se envía para mover al cuadricóptero en el eje roll (derecha o izquierda), Pitch (adelante o atrás) o altura (para subir y bajar el cuadricóptero).

A continuación se expone la estructura de los mensajes de las interfaces de comunicación inalámbrica para ejecución de comandos y telemetría del cuadricóptero desarrolladas:

Ilustración 4. Mensajes del cuadricóptero a la estación de control

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Tipo de mensaje** | **Código de mensaje** | **Contenido del mensaje** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **Estado** | 6 | Posición YAW positiva | Posición YAW negativa | Posición PITCH (de 0 a 180 siendo el valor 90 igual a 0 grados) | Posición ROLL (de 0 a 180 siendo el valor 90 igual a 0 grados) | Velocidad YAW positiva | Velocidad YAW negativa | Velocidad PITCH positiva | Velocidad PITCH Negativa | Velocidad ROLL positiva | Velocidad ROLL negativa | Altura | Estado Motores |  | | | | | | | | | | | | | | | | | | | | | | | | | |
| **Acknowledge** | 7 | Código mensaje recibido | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **Telemetría completa** | 8 | Estimado | | | | | | Bruto | | | | | | Filtrado | | | | | | Bruto | | | | | | Filtrado | | | | | | Bruto | Filtrado | Estimado | | PWM motor delantero | PWM motor trasero | PWM motor derecho | PWM motor izquierdo |
| Posición YAW positiva | Posición YAW negativa | Posición PITCH positiva | Posición PITCH negativa | Posición ROLL positiva | Posición ROLL negativa | Velocidad YAW positiva | Velocidad YAW negativa | Velocidad PITCH positiva | Velocidad PITCH positiva | Velocidad ROLL positiva | Velocidad ROLL positiva | Velocidad YAW positiva | Velocidad YAW negativa | Velocidad PITCH positiva | Velocidad PITCH negativa | Velocidad ROLL positiva | Velocidad ROLL negativa | Aceleración YAW positiva | Aceleración YAW positiva | Aceleración PITCH positiva | Aceleración PITCH positiva | Aceleración ROLL positiva | Aceleración ROLL positiva | Aceleración YAW positiva | Aceleración YAW negativa | Aceleración PITCH positiva | Aceleración PITCH negativa | Aceleración ROLL positiva | Aceleración ROLL negativa | Altura | Altura | Velocidad Z positiva | Velocidad Z negativa |

Fuente: elaboración propia.

Ilustración 4. Mensajes de la estación de control al cuadricóptero

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tipo de mensaje** | **Código de mensaje** | **Contenido del mensaje** | | |
| **Encendido de motores** | 0 | Para apagar motores: 0  Para apagar motores: 1 | | |
| **Movimiento** | 1 | Movimiento Eje Pitch | Movimiento Eje Roll | Movimiento de altura |

Fuente: elaboración propia

En la tabla que se presenta a continuación se expone el retardo producido por el envío de cada mensaje de las interfaces de comunicación desarrolladas:

Tabla 4. retardo de mensajes

|  |  |  |
| --- | --- | --- |
| **Mensaje** | **Retardo (medido en milisegundos)** | |
| **38400 bdps** | **115200 bdps** |
| Encendido | 0,83 | 0,27 |
| Movimiento | 1,25 | 0,41 |
| Estado | 3,125 | 1,04 |
| TelemetriaTotal | 8,75 | 2,91 |
| Acknowledge | 0,83 | 0,27 |

Fuente: elaboración propia

Se realizaron pruebas para medir la tasa de transmisión de mensajes en función de la distancia entre los módulos XBee, a partir de las cuales se obtuvieron los siguientes resultados:

Tabla 4. 6 Pruebas realizadas con el XBee a diferentes distancias, cada prueba envía un total de 200 mensajes.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Distancia | Prueba #1 | Prueba #2 | Prueba #3 | Media de mensajes recibidos | % de mensajes recibidos | % de mensajes no recibidos |
| Mensajes recibidos | Mensajes recibidos | Mensajes recibidos |
| 0 | 200 | 200 | 200 | 200 | 100 | 0 |
| 1 | 200 | 199 | 200 | 200 | 99.83333333 | 0.166666667 |
| 3 | 199 | 200 | 199 | 199 | 99.66666667 | 0.333333333 |
| 5 | 197 | 194 | 200 | 197 | 98.5 | 1.5 |
| 7 | 196 | 172 | 198 | 189 | 94.33333333 | 5.666666667 |
| 10 | 148 | 200 | 191 | 180 | 89.83333333 | 10.16666667 |

Fuente: elaboración propia

En la gráfica que se expone a continuación se presenta la media de mensajes recibidos en función de la distancia entre los módulos XBee:

Ilustracion 4. Porcentaje de mentajes recibidos en distancia

Fuente: elaboración propia

1. ***Sistemas de control***
   1. **Arquitectura de control**

Se diseñó una arquitectura de control multi-capa para la regulación en vuelo de la posición angular, velocidad angular, velocidad en el eje z, y posición en el eje z (altura) del mismo, mediante algoritmos de control Proporcional-Integral-Derivativo. La arquitectura de control propuesta se describe a continuación:

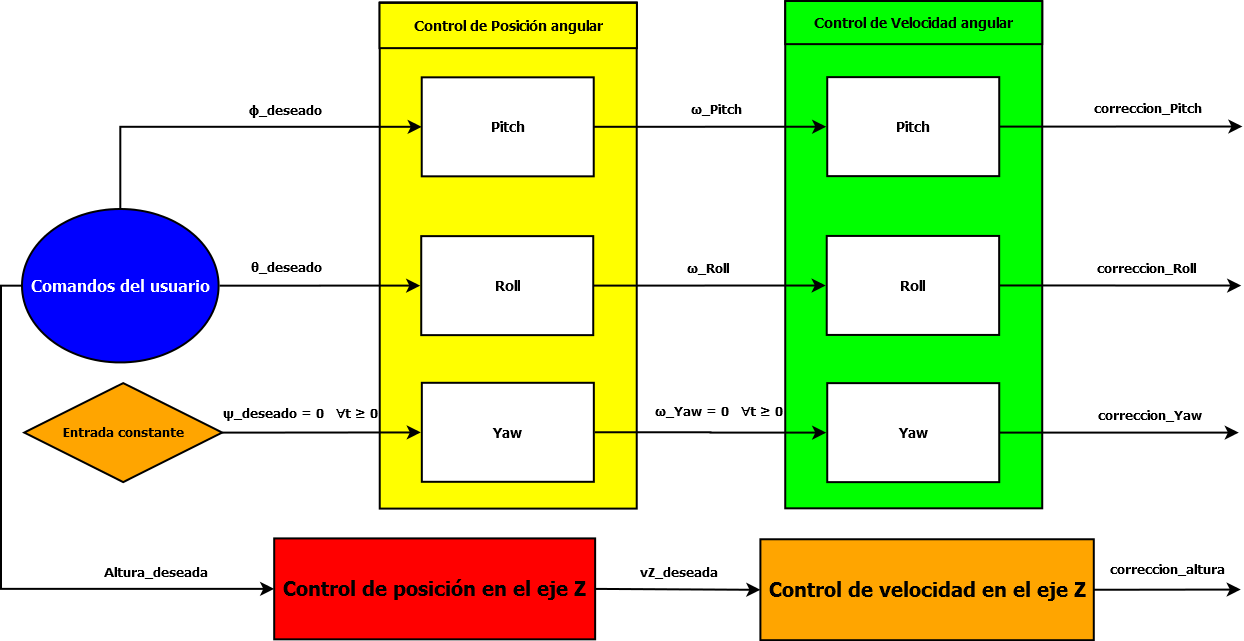


Ilustración 4. Arquitectura de control propuesta

Fuente: elaboración propia.

La arquitectura de control propuesta se desglosa en:

1. Sistema de control de posición en el eje z (Altura).
   1. Entrada: estimación de altura del cuadricóptero.
   2. Referencia: altura deseada, especificada por el usuario en tiempo real.
   3. Salida: velocidad lineal en z para alcanzar la altura deseada.
   4. Intervalo de ejecución: cada 30 milisegundos.
2. Sistema de control de velocidad lineal en el eje z.
   1. Entrada: velocidad lineal del cuadricóptero.
   2. Referencia: velocidad lineal en z deseada.
   3. Salida: corrección de altura.
   4. Intervalo de ejecución: cada 30 milisegundos.
3. Sistema de control de posición angular de Pitch.
   1. Entrada: ángulo de Pitch del cuadricóptero.
   2. Referencia: ángulo de Pitch deseado, especificado por el usuario en tiempo real.
   3. Salida: velocidad angular para alcanzar la posición de Pitch deseada.
   4. Intervalo de ejecución: cada 20 milisegundos.
4. Sistema de control de posición angular de Roll.
   1. Entrada: ángulo de Roll del cuadricóptero.
   2. Referencia: ángulo de Roll deseado, especificado por el usuario en tiempo real.
   3. Salida: velocidad angular para alcanzar la posición de Roll deseada.
   4. Intervalo de ejecución: cada 20 milisegundos.
5. Sistema de control de posición angular de Yaw.
   1. Entrada: ángulo de Yaw del cuadricóptero.
   2. Referencia: ángulo de Yaw=0, constante.
   3. Salida: velocidad angular para alcanzar la posición de Yaw deseada.
   4. Intervalo de ejecución: cada 20 milisegundos.
6. Sistema de control de velocidad angular de Pitch.
   1. Entrada: velocidad angular del cuadricóptero en el eje de Yaw.
   2. Referencia: velocidad angular de Pitch deseada, obtenida como salida del sistema de control de posición angular de Pitch.
   3. Salida: corrección de Pitch.
   4. Intervalo de ejecución: cada 10 milisegundos.
7. Sistema de control de velocidad angular de Roll.
   1. Entrada: velocidad angular del cuadricóptero en el eje de Yaw.
   2. Referencia: velocidad angular de Roll deseada, obtenida como salida del sistema de control de posición angular de Pitch.
   3. Salida: corrección de Pitch.
   4. Intervalo de ejecución: cada 10 milisegundos.
8. Sistema de control de velocidad angular de Yaw.
   1. Entrada: velocidad angular del cuadricóptero en el eje de Yaw.
   2. Referencia: velocidad angular de Yaw deseada, obtenida como salida del sistema de control de posición angular de Yaw.
   3. Salida: corrección de Yaw.
   4. Intervalo de ejecución: cada 10 milisegundos.

Debido a la magnitud de las vibraciones en el chasis del cuadricóptero, sólo se pudo obtener un rendimiento satisfactorio en los sistemas de control PID de velocidad angular de Yaw, Pitch y Roll, y en el sistema de control PID de Yaw. Por ello, se modificó la arquitectura de control propuesta inicialmente, para que el usuario pudiera emitir consignas de velocidad angular y velocidad global de los motores en tiempo real (Tarea que debían realizar los sistemas de control de posición angular y altura en la arquitectura propuesta). La arquitectura final implementada se presenta a continuación:

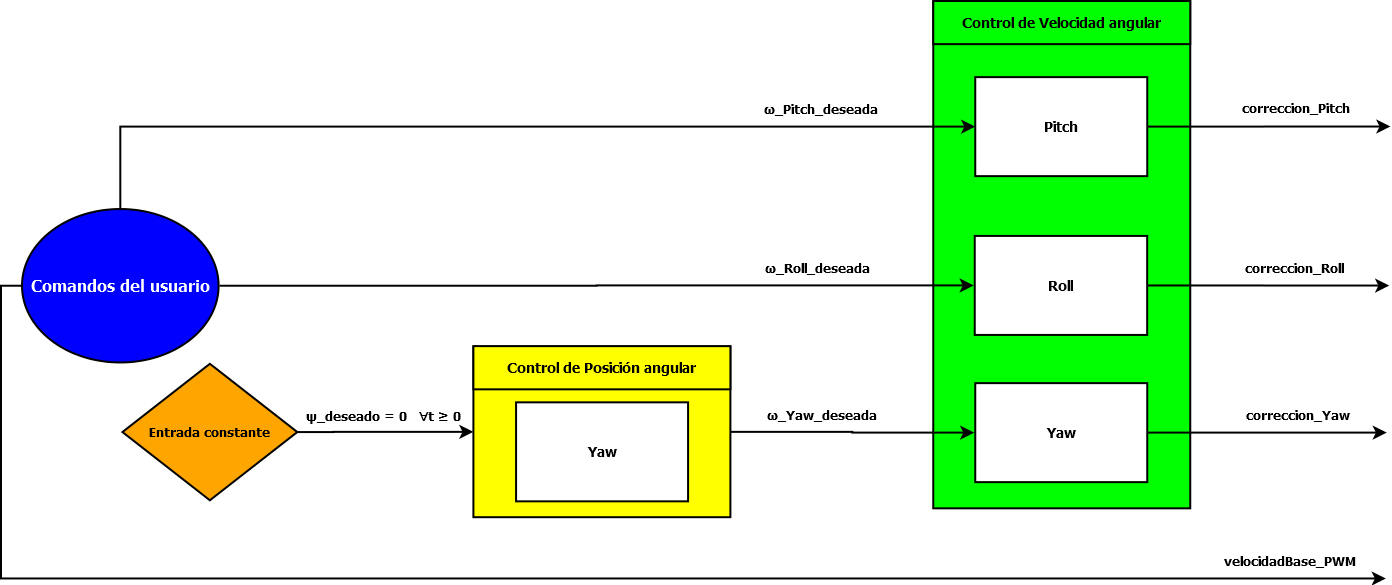


Ilustración 4. Arquitectura de control simplificada

Fuente: elaboración propia.

* 1. ***Simulación***

El modelo linealizado del cuadricóptero fue implementado en el software MATLAB, y se realizaron simulaciones para verificar la factibilidad de realizar el control de posición angular, velocidad angular y altura del cuadricóptero mediante la arquitectura de control propuesta. Las simulaciones se llevaron a cabo mediante el método de Euler, ya que se partía del modelo linealizado del cuadricóptero y el algoritmo Proporcional-Integral-Derivativo modela la retroalimentación en lazo cerrado como una ecuación diferencial lineal.

**Diagrama del algoritmo de simulacion**

El cálculo de los parámetros de ejecución de los algoritmos PID se realizó por prueba y error, y se obtuvo un rendimiento satisfactorio de la arquitectura de control propuesta, con lo cual se verificó la efectividad de la misma. A continuación se exponen los resultados obtenidos en la ejecución de los sistemas de control de posición angular, velocidad angular y altura (posición y velocidad lineal en el eje z):

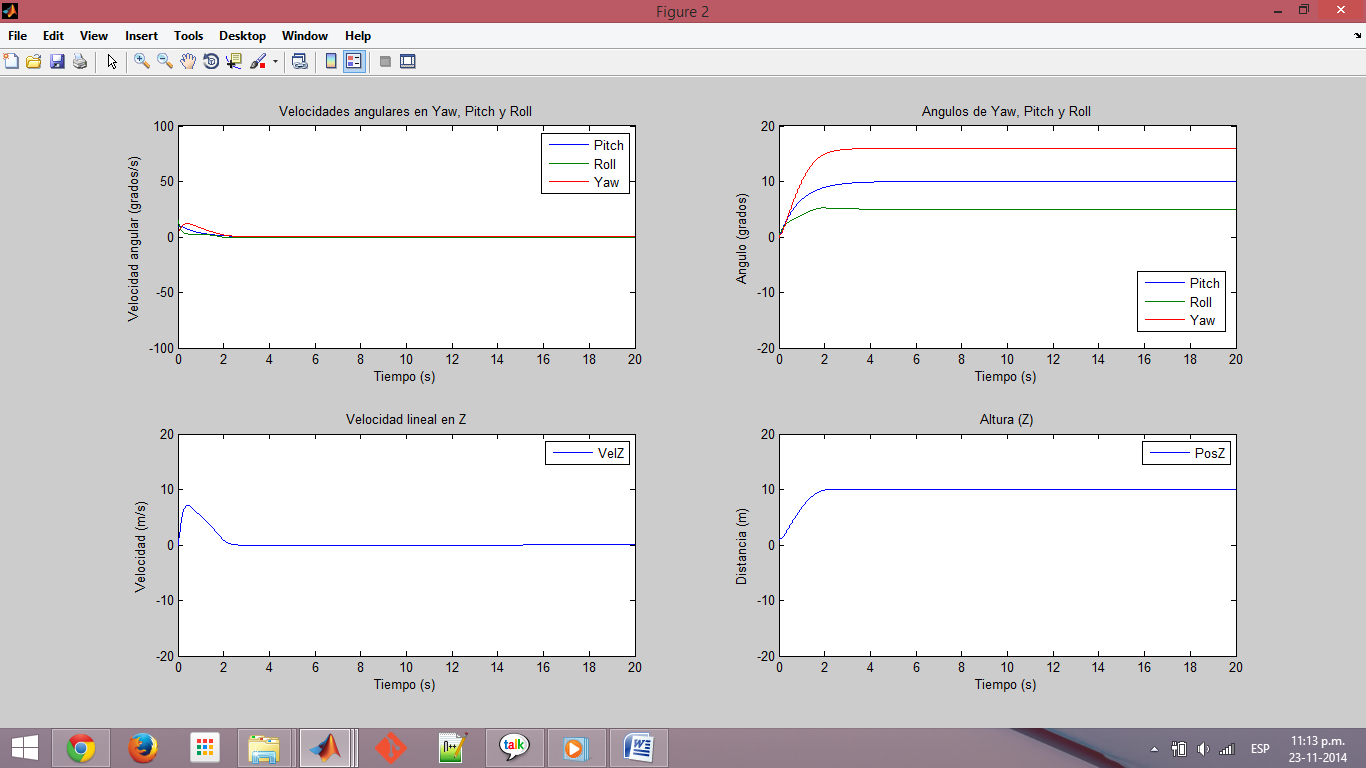


Ilustración 4. 23 Rendimiento de los sistemas de control PID de la arquitectura propuesta en simulación

Fuente: elaboración propia.

De igual manera, se obtuvo un rendimiento safisfactorio de la arquitectura de control simplificada que finalmente fue implementada en el cuadricóptero, como se expone a continuación:

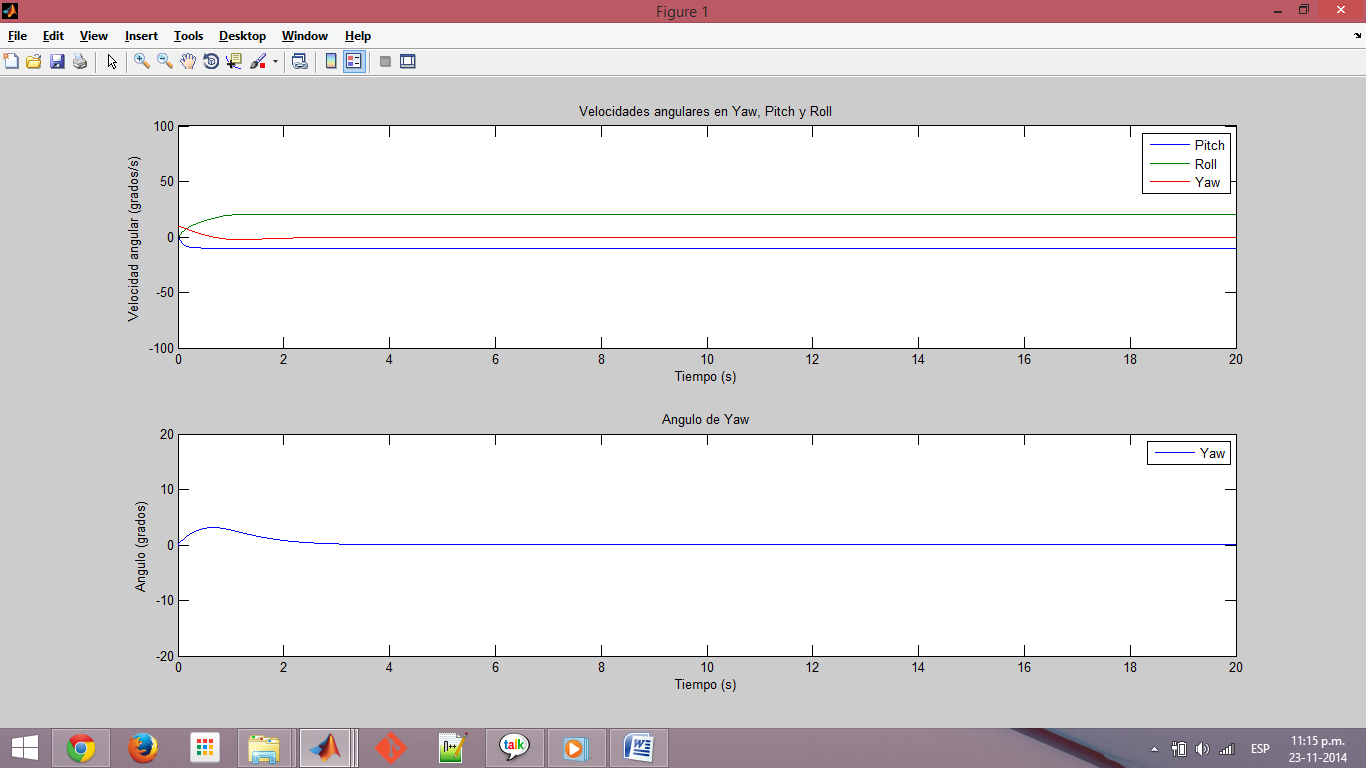


Ilustración 4. 23 Rendimiento de los sistemas de control PID de la arquitectura simplificada en simulación

Fuente: elaboración propia.

* 1. ***Implementación en el cuadricóptero***

Se definió una velocidad base constante, para mantener homogeneidad en el comportamiento dinámico del cuadricóptero durante el proceso de calibración, pruebas, y vuelo. Las señales de Módulación por Ancho de Pulsos (PWM) a enviar a los motores se calcularon de la siguiente manera:

Para la implementación de los sistemas de control se utilizó la librería PID de Arduino, la cual permite encapsular en una clase todos los atributos y métodos suficientes para la ejecución de un sistema de control Proporcional-Integral-Derivativo. El cálculo de las constantes de los sistemas de control se realizó mediante prueba y error.

A continuación se expone el flujo de ejecución de la rutina de control implementada en el microcontrolador Arduino Nano, a bordo del cuadricóptero:

**DIAGRAMA DEL ALGORITMO**

Se realizaron pruebas con la arquitectura de control propuesta, en las cuales no se obtuvo un resultado exitoso por la inestabilidad de la estimación de posición angular del cuadricóptero producida por las vibraciones del chásis. A consecuencia de ello, tampoco se pudo obtener un rendimiento adecuado de los sistemas de control de altura, ya que la estimación de altura del cuadricóptero se realiza en base a los ángulos de inclinación del mismo.

**Gráfica que demuestre rendimiento de los sistemas de control de posición angular, velocidad angular y altura.**

Se realizaron pruebas de vuelo con la arquitectura de control simplificada, y se obtuvieron resultados satisfactorios de seguimiento de consignas fuera de la banda de error producida por las vibraciones del chásis. El manejo del cuadricóptero requirió de un lapso de práctica, ya que requiere una realimentación constante por parte del usuario para ajustar la inclinación y altura del cuadricóptero en vuelo, y sólo puede asegurarse un seguimiento de consignas superiores a 20 grados/segundo.

**Gráfica que demuestre rendimiento de los sistemas de control de velocidad angular y posición angular en Yaw del cuadricóptero en vuelo.**

1. ***Plataforma de pruebas***
   1. ***Software de telemetría y comandos***

El software de telemetría y comandos se desarrolló en 3 fases, las cuales se describen a continuación:

1. **Primera fase:** se desarrolló un script en MATLAB para la lectura y escritura del puerto serial, graficación de datos recibidos y el envío de comandos de movimiento al cuadricóptero. Se encontraron numerosas dificultades para modularizar el software desarrollado, una alta abstracción que dificultaba la realización exhaustiva de validaciones, y problemas en la ejecución del recolector de basura del intérprete en tiempo de ejecución de MATLAB que llevaban a tener que reiniciar MATLAB continuamente para crear exitosamente un objeto manejador del puerto serial.
2. **Segunda fase:** se desarrollaron módulos en el lenguaje de programación Python para lectura y escritura del puerto serial, graficación de datos recibidos, y detección de comandos del control Logitech Rumblepad II, con las librerías pyserial, pyqtgraph y pygame. Los módulos desarrollados para la lectura del puerto serial y la detección de comandos del control Logitech Rumblepad II ofrecieron un rendimiento safisfactorio en pruebas de ejecución, pero no así el módulo de graficación desarrollado, requiriendo una implementación multi-hilo exhaustiva. Por otro lado, se encontraron problemas a la hora de realizar la comunicación entre los hilos de ejecución de los módulos desarrollados.
3. **Tercera fase (final):** en vista de los problemas de robustez, comunicación entre módulos y rendimiento del software de graficación de datos recibidos por telemetría, se decidió adaptar los módulos de detección de comandos del control Logitech Rumblepad II y de lectura del puerto serial desarrollados para funcionar sobre la plataforma de software para robótica Robot Operating System (ROS).

El software se rediseñó y desarrolló como una serie de paquetes para ROS, que llevan a cabo de forma independiente la transmisión de datos entre el cuadricóptero y la PC, la obtención de comandos del usuario mediante un control Logitech Gamepad II y la exportación de los mensajes de estado del cuadricóptero a formato CSV. La codificación de los mismos se realizó en el lenguaje de programación Python, haciendo uso de la librería rospy, y siguiendo los estándares de programación de ROS.

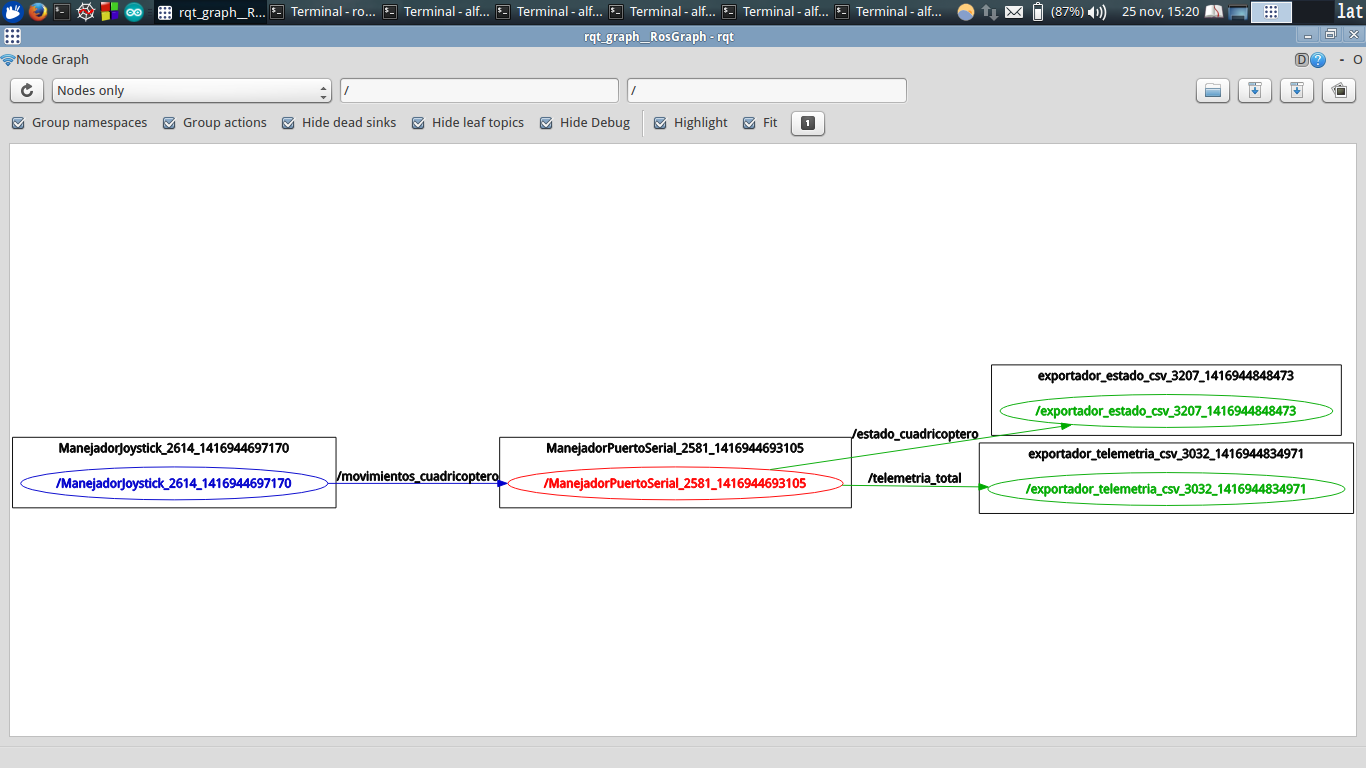
****

Ilustración 4. 24: grafo de los nodos desarrollados en ejecución.

Fuente: elaboración propia.

Para realizar la comunicación entre los paquetes de software en tiempo de ejecución se definieron formatos de mensaje personalizados, los cuales se describen a continuación:

Al realizarse la comunicación entre los paquetes de forma asíncrona, como es natural en los paquetes de software desarrollados para ROS, la plataforma de software desarrollada tuvo un muy bajo acoplamiento entre sus componentes, lo cual aseguró una alta robustez del sistema ante fallas.

Tabla 4. 6. Formatos de mensaje utilizados por los nodos desarrollados

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nodo** | **Encendido** | **ComandoMovimiento** | **EstadoCuadricoptero** | **TelemetriaTotal** |
| **comunicacion\_serial** | X | X | X | X |
| **logitech\_rumblepad\_ii** |  | X |  |  |
| **exportador\_estado\_csv** |  |  | X |  |
| **exportador\_telemetria\_csv** |  |  |  | X |

*Fuente: elaboración propia.*

A continuación se describen los paquetes de software desarrollados:

* **comunicación\_serial:** Al iniciar su ejecución instancia un objeto de la clase handler\_serial, el cual encapsula el manejo del puerto serial, y permite al programa principal realizar llamados simples a funciones para llevar a cabo el envío y recepción de mensajes de encendido o apagado, estado, y comandos de movimiento. Posteriormente, son inicializadas rutinas de subscripción a los tópicos encendido\_cuadricoptero y movimientos\_cuadricoptero, y rutinas para publicación en el tópico estado\_cuadricoptero.

Durante su ejecución, el nodo del paquete publica todos los mensajes de estado recibidos por el objeto handler\_serial, y realiza el envío de los comandos de encendido o apagado, y y de movimiento. Si se finaliza su ejecución de forma abrupta, o se desconecta el cable de comunicación del puerto serial, el nodo automáticamente detiene la comunicación y libera el puerto serial como recurso del sistema.

El manejo del puerto serial en la clase handler\_serial se realiza mediante la librería pyserial, el objeto Serial, y sus métodos correspondientes.

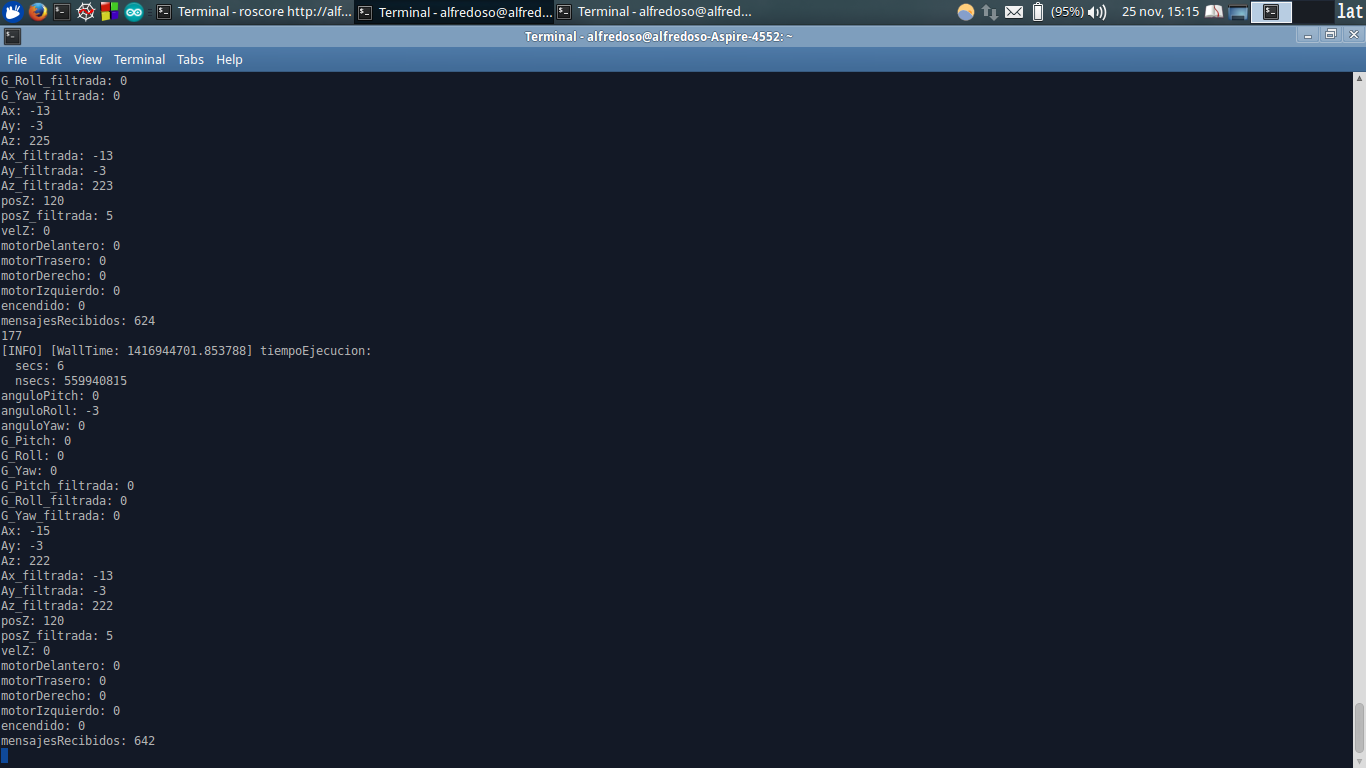


Ilustración 4. 26 Nodo comunicación\_serial en funcionamiento.

Fuente: elaboración propia.

* **logitech\_rumblepad\_ii:** Al iniciar su ejecución instancia un objeto de la clase handler\_joystick, el cual encapsula la detección y manejo de los eventos del control. Igualmente, el nodo inicializa rutinas para la publicación de los comandos del usuario en los tópicos encendido\_cuadricoptero y movimientos\_cuadricoptero.

Durante su ejecución el nodo del paquete verifica constantemente la emisión de comandos por parte del usuario, y en caso de haber sido emitido un comando desde el control, se calcula la consigna de movimiento mediante un polinomio de tercer grado, el nodo publica dicho comando al tópico que corresponda. Si se finaliza su ejecución de forma abrupta, o se desconecta el cable del control, el nodo automáticamente detiene la publicación de mensajes y libera el control como recurso del sistema.

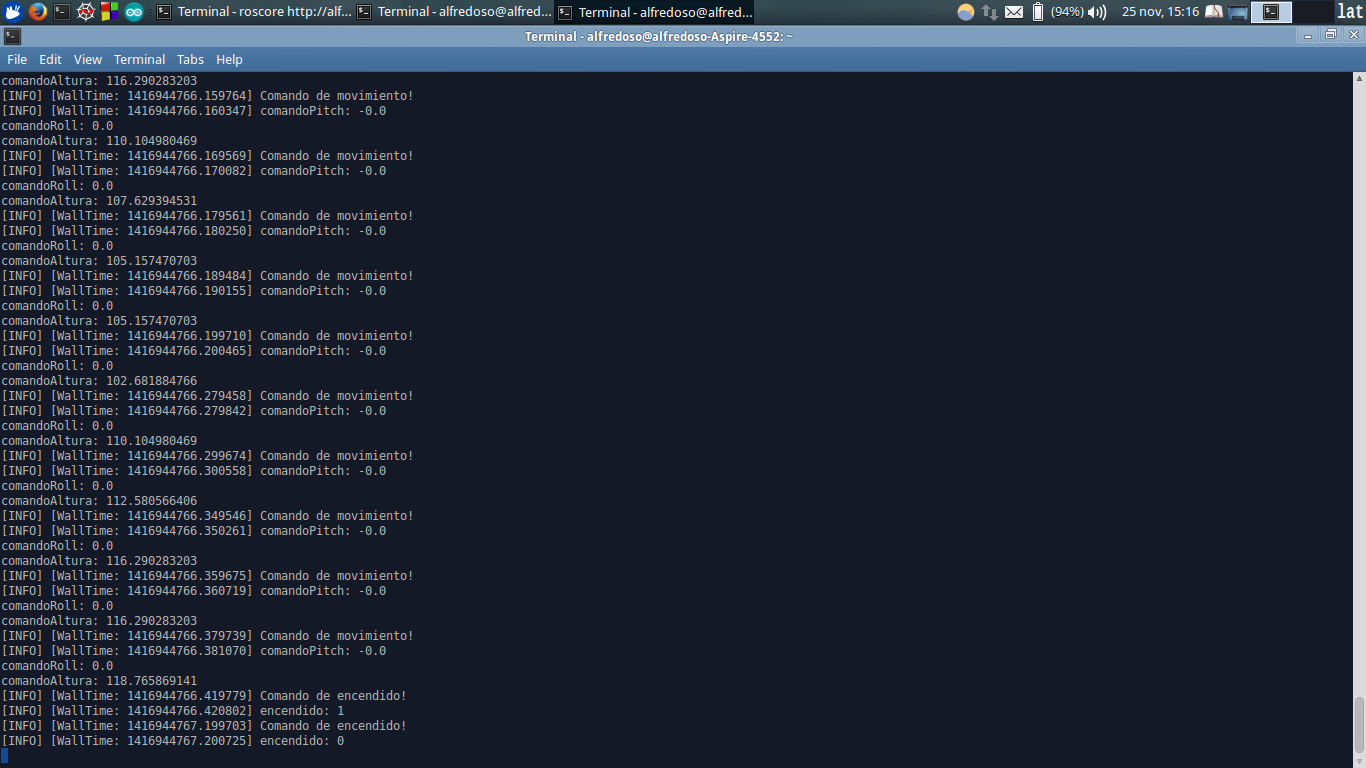


Ilustración 4. 26 Nodo logitech\_rumblepad\_ii en funcionamiento.

Fuente: elaboración propia.

El manejo del puerto serial en la clase handler\_joystick se realiza mediante la librería pygame, el objeto Joystick, y sus métodos correspondientes.

Ilustración 4. Arquitectura de control implementada

Fuente: elaboración propia.



* **exportador\_estado\_csv:** el nodo consta de una rutina de subscripción al tópico estado\_cuadricoptero, que escribe todos los mensajes publicados en el mismo en un archivo en formato CSV (Siguiendo las pautas de codificación de archivos CSV para su lectura con MATLAB).
* **exportador\_telemetria\_csv:** el nodo consta de una rutina de subscripción al tópico telemetría\_total, que escribe todos los mensajes publicados en el mismo en un archivo en formato CSV (Siguiendo las pautas de codificación de archivos CSV para su lectura con MATLAB).

No se desarrolló un paquete para la visualización de datos obtenidos mediante telemetría por considerarse que los paquetes para visualización de gráficas en tiempo real para ROS existentes (rxplot y rqt\_plot) brindan un rendimiento óptimo y una variedad de opciones de visualización adecuadas para la visualización del estado del cuadricóptero en tiempo real.

****

Ilustración 4. 25 Visualización de datos en rqt\_plot.

Fuente: elaboración propia.

Finalmente, en el diagrama que se presenta a continuación se describe a detalle el flujo de ejecución del software desarrollado:

**Poner un diagrama de flujo del programa, que sea sencillo pero muy explicativo.**

* 1. ***Scripts para análisis de respuesta en frecuencia de los sensores***

Se desarrollaron una serie de scripts de MATLAB para realizar el análisis de respuesta en frecuencia de los datos crudos y filtrados de los sensores, a partir de archivos CSV generados por los nodos exportador\_estado\_csv y exportador\_telemetria\_csv, desarrollados como parte de la plataforma de software de telemetría y comandos.

Los scripts desarrollados aplican un algoritmo de transformada rápida de Fourier sobre los datos de los sensores, realizan la traslación de los mismos en el dominio de la frecuencia, y muestran al usuario gráficas de los datos obtenidos en el dominio del tiempo y de la frecuencia, para su análisis.

A continuación se presenta una captura de pantalla de la ventana de visualización de ángulos, que forma parte del conjunto de ventanas que se muestran al ejecutar los scripts desarrollados, con un formato similar:

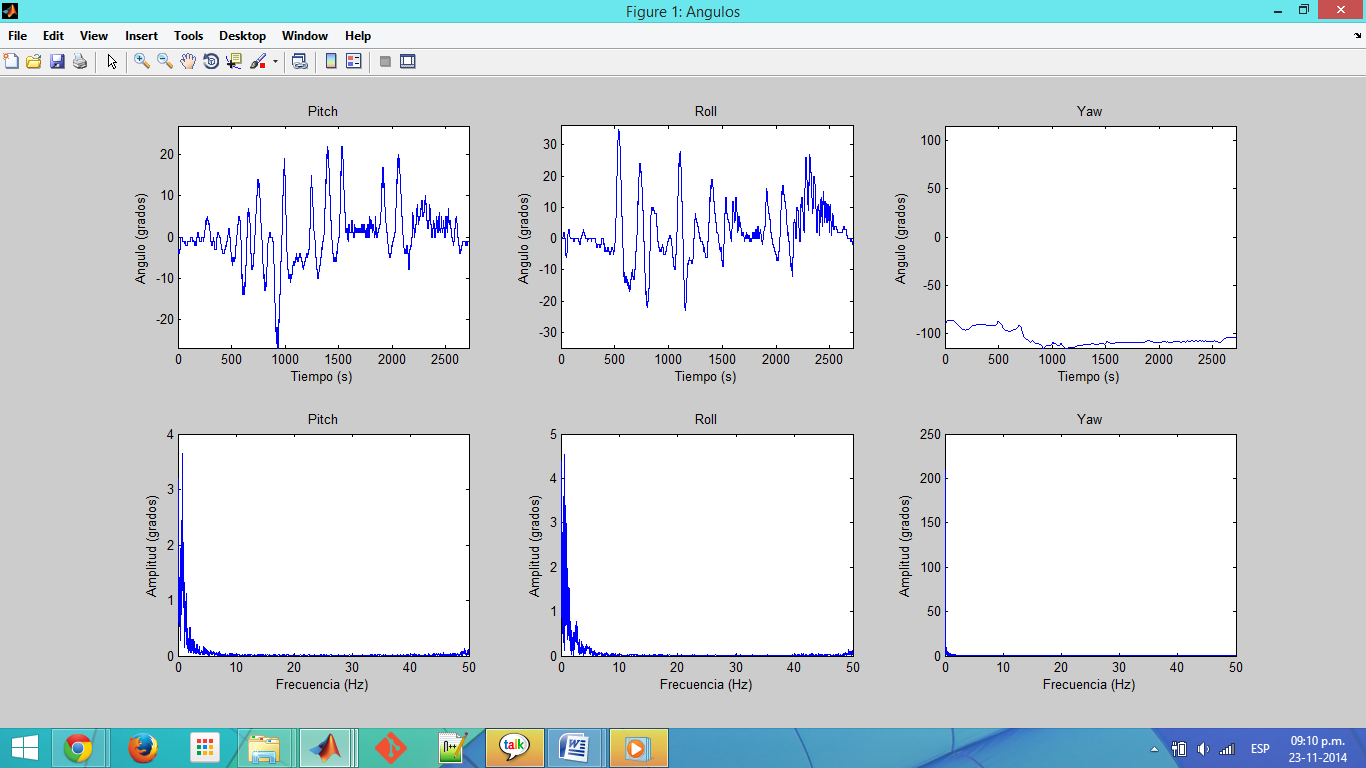


Ilustración 4. 25 Captura de pantalla de la ventana de visualización de datos de ángulos en MATLAB

Fuente: elaboración propia

* 1. ***Montaje para la ejecución de pruebas en un solo eje del cuadricóptero***

Una vez comprobado el montaje de todos los componentes sobre el cuadricóptero se procedió a trabajar con el algoritmo de estabilización de manera independiente para los ejes roll y pitch, esto se logro utilizando una base de madera la cual restringía el movimiento del cuadricóptero a solo uno de los ejes.



Ilustración 4. Ejemplo de base utilizada para restringir el movimiento del cuadricóptero en un eje

Fuente: elaboración propia

* 1. ***Montaje para la ejecución de pruebas en vuelo restringido***

Una vez estabilizados los ejes de roll y pitch de forma independiente se empezó a calibrar el eje yaw, para esto se sujeto al cuadricóptero de una cuerda, la cual fue amarrada al centro de la estructura, y se colgó en un lugar suficientemente alto y amplio como para que pudiera moverse sin chocar contra otros objetos.

Además se pudo probar el ajuste de roll y pitch trabajando de forma simultánea y la robustez del sistema de control frente a perturbaciones del ambiente.

Durante estas pruebas el ciclo de trabajo base estuvo por debajo del 70% para evitar que despegase y se enredara con la cuerda.



Ilustracion 4. Diagrama de movimiento sobre el eje yaw.

*Fuente:* [*http://blog.oscarliang.net/wp-content/uploads/2013/06/quadcopter-rotating.png*](http://blog.oscarliang.net/wp-content/uploads/2013/06/quadcopter-rotating.png)

1. ***Perspectiva general del sistema implementado***

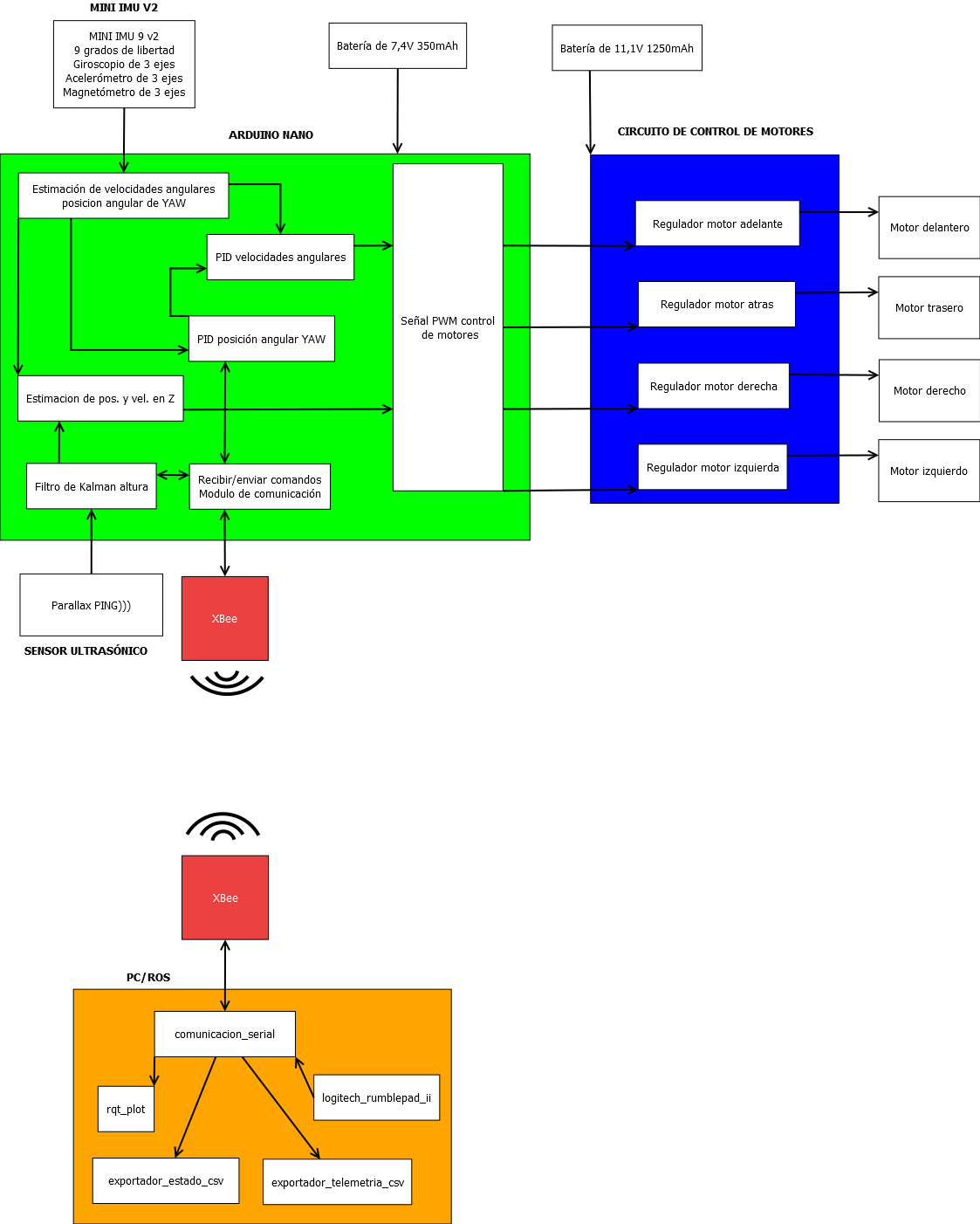
Los circuitos desarrollados, la Unidad de Medición Inercial, el sensor ultrasónico de distancia y las baterías fueron montados en el chasis de cuadricóptero Draganflyer V como se ilustra a continuación:



Ilustracion 4. 27 Cuadricóptero ensamblado

*Fuente: elaboración propia.*

Finalmente, en el diagrama que se presenta a continuación, se describe la arquitectura general del sistema implementado, incluyendo las interfaces de comunicación desarrolladas, los algoritmos de estimación, la plataforma de software en Robot Operating System y los sistemas de control:

****

Ilustracion 4. 28 Diagrama general del sistema.

*Fuente: elaboración propia.*