Apéndices y anexos

1. **Apéndice A: Diseño del circuito de alimentación y control de motores.**

Para realizar la conmutación a alta velocidad de cada motor en un solo sentido, se utilizó un MOSFET IRFZ44N y un diodo de rodada libre, para forzar la descarga de la inductancia del motor al cerrar el canal del MOSFET. Estos fueron seleccionados de entre todos los componentes disponibles en el mercado venezolano por su alta velocidad de conmutación, alta tolerancia y estabilidad ante valores altos de corriente y voltaje, y baja resistencia drenador-surtidor en el caso del IRFZ44N. Se realizaron pruebas para caracterizar la conductividad y resistencia interna del MOSFET ante distintos valores de voltaje aplicados en la compuerta del mismo, al encender uno de los motores del cuadricóptero de forma continua (Ciclo de trabajo de 100% con un voltaje Vgs fijo), con una alimentación de 12V -cercana al voltaje máximo de carga de la batería de polímero de litio seleccionada para alimentar el conjunto-.

Para realizar la caracterización de los MOSFET IRFZ44N se utilizó el siguiente circuito de prueba:



Ilustracion 4. Diagrama de circuito para estimación de Vgs óptimo para conmutar MOSFET con carga de motor DC en el surtidor.

Fuente: elaboración propia

A continuación se presenta la curva característica de corriente drenador-surtidor (Ids) en función del voltaje de compuerta (Vgs) del MOSFET IRFZ44N, identificada a partir de las pruebas:

Ilustracion 4. Relación **Voltaje de compuerta** – **Corriente drenador-surtidor** del MOSFET IRFZ44N, en presencia de carga del motor.

Fuente: elaboración propia.

De igual manera, a continuación se presenta la curva característica de resistencia drenador-surtidor (Ids) en función del voltaje de compuerta (Vgs) del MOSFET IRFZ44N:

Ilustracion 4. Relación **Voltaje de compuerta** - **Corriente drenador-surtidor** del MOSFET IRFZ44N, en presencia de carga del motor.

Fuente: elaboración propia.

A partir del análisis realizado se identificó el rango de valores entre 4,5 y 6 voltios como el rango de valores de tensión de compuerta óptimos para el funcionamiento del MOSFET con los motores del cuadricóptero como carga en el drenador. Considerando el efecto de descarga de la batería a utilizar, se decidió utilizar un voltaje de al menos 6V, siendo éste cota superior del rango de valores óptimos identificado, como tensión de activación de la compuerta del MOSFET IRFZ44N.

Para disminuir los efectos de ruido electromagnético que pudieran ser introducidos por la rápida conmutación de los MOSFETS y los motores al circuito de lógica, sensores y comunicación, el cual envía las señales de PWM para la regulación de velocidad de los motores, se decidió utilizar opto-acopladores para separar totalmente la etapa. Se seleccionó el modelo 4N26, presente en el mercado de componentes electrónicos venezolano, por su alta velocidad de conmutación, tolerancia a altos valores de voltaje y corriente, bajo precio, y simplicidad de configuración. Para obtener la tensión de salida de 6V para conmutar la compuerta del MOSFET IRFZ44N se diseñó un circuito que hace uso de un divisor de voltaje en el emisor del fototransistor, en configuración colector común, como se presenta a continuación:

****

Ilustracion 4. Diagrama de circuito para estimación de carga total de foto-transistor del opto-acoplador 4N26 en configuración colector común.

Fuente: elaboración propia.

Para calcular la resistencia total de la carga del colector común del fototransistor se evaluó el tiempo de respuesta del mismo ante impulsos de PWM de 490 Hz. Se realizaron pruebas con ayuda de un osciloscopio digital para medir el rendimiento del opto-acoplador 4N26, y se seleccionó un valor de 900Ω para la resistencia de carga total del opto-acoplador en configuración colector común, por obtenerse una alta velocidad de respuesta en el opto-acoplador. A partir del valor de resistencia total seleccionado se realizó el cálculo de las resistencias R1 y R2 para el circuito de conmutación con divisor de voltaje. El valor calculado para R1 fue de 390Ω, y para R2 fue de 510Ω.

En la figura que sigue se presenta la respuesta del circuito de conmutación diseñado al recibir una señal de PWM con ciclo de trabajo de 98%:



Ilustracion 4. Señal de salida del opto-acoplador ante una señal de PWM con un ciclo de trabajo de 98%.

Fuente: elaboración propia.

Posteriormente, se sometió al circuito a una señal de PWM con ciclo de trabajo de 23%, como se ilustra a continuación:



Ilustracion 4. Señal de salida del opto-acoplador ante una señal de PWM con un ciclo de trabajo de 23%.

Fuente: elaboración propia.

El circuito de regulación de velocidad diseñado, para el control de velocidad individual de cada motor se presenta en el diagrama que sigue:



Ilustracion 4. Diagrama de circuito de regulación de velocidad diseñado.

Fuente: elaboración propia.

Finalmente, se diseñó y construyó una placa de circuito impreso de una sóla capa consistente en un arreglo de cuatro módulos para regulación de velocidad desarrollados en el presente trabajo. A continuación se muestra el diseño del mismo:



Ilustracion 4. Diseño de circuito impreso del circuito de alimentación y control de motores.

Fuente: elaboración propia.

1. **Apéndice B: Descripción del modelo de estimación de posición angular y velocidad angular del cuadricóptero.**
   1. **Cálculo de velocidad angular**

El cálculo de velocidad angular se realizó a partir de las mediciones realizadas con el giroscopio. Partiendo de la información provista por la hoja de datos del sensor L3GD20, el mismo puede configurarse para obtener una sensibilidad de 8,75, 17,5 y 70 milésimas de grado por segundo por cada dígito de medición obtenido (mdps/digit – en inglés *millidegrees per second per digit*), y un rango de medición de 250, 500 y 2000 grados por segundo (dps – en inglés *degree per second*). Se decidió configurar el rango de medición a 250 grados por segundo y la senbilidad del sensor a 8,75 mdps/digit, por considerarse rango y senbilidad suficientes para medir las velocidades del cuadricóptero realizando movimientos simples en vuelo. En base a esto último se calculó la ganancia del giroscopio, para convertir todas las mediciones obtenidas mediante el mismo, en milésimas de grado por segundo, a grados por segundo, como se ilustra a continuación:

siendo:

* 1. **Estimación de ángulos de Pitch y Roll a partir del acelerómetro**

Se realizó un estimado del ángulo de inclinación del cuadricóptero a partir de las mediciones del acelerómero, el cual provee una descomposición de la fuerza de aceleración del cuadricóptero en tres (3) ejes perpendiculares (x, y, z). El acelerómetro puede detectar constantemente la fuerza de gravedad, en magnitud, dirección, y fuerza, en cada uno de sus tres (3) ejes, y en base a ello puede establecerse un marco de referencia absoluto a partir del cual calcular los ángulos de Pitch y Roll del sensor, y en consecuencia, del cuadricóptero. La estimación de ángulos se realizó siguiendo el procedimiento expuesto en [STMicroElectronics 2010], el cual es presentado a continuación:



Ilustracion 4. Modelo de estimación de angulos

Fuente: [STMicroElectronics]

siendo:

Las estimaciones de los ángulos de Pitch y Roll calculadas a partir de los datos del acelerómetro, a pesar de ser precisas y permitir mantener un marco de referencia absoluto, en base a la fuerza de gravedad de la tierra, presentan un alto porcentaje de ruido, ya que el acelerómetro es altamente sensible a perturbaciones provocadas por fuerzas externas que incidan sobre el mismo.

* 1. **Estimación de posición angular a partir del giroscopio**

Se realizó un estimado del ángulo de inclinación del cuadricóptero, mediante integración numérica de las velocidades de rotación de Yaw, Pitch y Roll, como se describe a continuación:

Durante un lapso corto de tiempo, este estimado del ángulo de inclinación en cada eje puede ser preciso, pero, tiende a presentar deriva constante y a alejarse de los valores reales a medir, por no realizarse la misma respecto a un marco de referencia absoluto.

* 1. **Combinación de las estimaciones de posición angular del acelerómetro y giroscopio**

Al combinar la precisión del acelerómetro para medir inclinación respecto al marco de referencia absoluto del planeta Tierra, con la sensibilidad y estabilidad de la estimación de ángulo realizada a partir de los datos del giroscopio para medir los movimientos de rotación alrededor de cada eje del sensor, puede obtenerse una estimación de ángulo precisa, estable, y de alta sensibilidad.

Como es descrito en [Burgard 2005],si es el estado estimado de un proceso, y una observación acerca del estado del mismo, puede aplicarse la Regla de Bayes para estimar la probabilidad de que el proceso se encuentre en el estado X a partir de Z:

En base a [Sturm 2013] y [Burgard 2005], por la Ley de Probabilidad Total, el término puede ser tratado como una constante de normalización , tal que:

Por lo cual:

Para el caso particular del presente trabajo en el que se realizó estimación de posición angular mediante la combinación de datos de los sensores acelerómetro y giroscopio, por lo cual se cuenta con observaciones y , por lo cual el modelo se reduce a:

Se asume además que:

+ )

+ )

Siendo el error de estimación a partir de las observaciones de ambos sensores. Luego:

Finalmente, se decidió utilizar una aproximación del ángulo bajo la suposición de que el error de estimación supone una cantidad despreciable:

+

El modelo derivado a partir de teoría de probabilidades para la combinación de las estimaciones de ángulo del giroscopio y acelerómetro, coincide con el de un filtro digital de uso altamente extendido conocido como *Filtro Complementario* [Gaydou 2007], [Colton 2007], el cual se fundamenta en la combinación de un filtro pasa bajos y un filtro pasa altos, ambos de primer orden, para la composición de los espectros de frecuencias de dos señales lineales invariantes en el tiempo en una tercera señal de salida. El filtro complementario puede ser descrito mediante las siguientes ecuaciones en el dominio de la frecuencia [Gaydou 2007]:



Ilustracion 4. Modelo del Filtro complementario

Fuente: [Gaydou 2007]

En donde:

La ganancia del filtro pasa altos puede ser descrita a partir de la siguiente ecuación:

Donde:

constante de tiempo del filtro.

diferencial de tiempo.

Tanto el modelo probabilístico como el modelo basado en el análisis de señales del proceso de estimación de ángulo coinciden en la relación fundamental que establecen entre la estimación final de ángulo, y la estimación del giroscopio y acelerómetro, por lo cual se puede considerar que, permiten analizar la influencia de los parámetros y sobre el espectro de frecuencias de las estimaciones de ángulos como funciones de tiempo continuo. Finalmente, que se cumple la siguiente relación: Mejorar

El cálculo de la ganancia del filtro pasa altos, y con ello, de y , se realizó mediante prueba y error, teniendo como criterio de selección la estabilidad, precisión y exactitud de la estimación de ángulo realizada, y el tiempo de respuesta de la misma. Se estableció un valor de k=0,03 para la ganancia del filtro.

Es importante recalcar que por las características de funcionamiento del acelerómetro, y el rango de la función , el procedimiento de estimación de ángulos de Pitch y Roll presentado sólo permite la aproximación de los mismos en un rango entre menos noventa y noventa grados, sin poder detectar si el cuadricóptero se encuentra en un ángulo fuera de ese rango o volteado. No obstante, el algoritmo desarrollado satisface las necesidades del proyecto realizado en el presente trabajo de investigación, ya que el algoritmo a desarrollar sólo apunta a estabilizar el cuadricóptero, más no a brindar posibilidades de realizar vuelo acrobático.

**Gráfica que demuestre precisión y estabilidad del ángulo calculado con el filtro complementario.**

1. **Apéndice C: Descripción del modelo de estimación de altura del cuadricóptero.**

Para la obtención de datos del sensor ultrasónico de distancia se utilizó la librería NewPing V1.3 para Arduino, la cual permite el manejo del sensor mediante la encapsulación del mismo como un objeto de la clase NewPing, y basa su funcionamiento en la emisión de un pulso del sensor, y la ejecución de una rutina para detección y medición del tiempo de retorno del pulso ultrasónico emitido. La librería NewPing ofrece una mayor precisión que el método de detección de pulsos utilizado por defecto en Arduino mediante la función pulseIn(), y no interfiere en la ejecución del resto del código, ya que el proceso de detección de pulsos se ejecuta como una rutina de interrupción sobre el temporizador timer2 de las tarjetas Arduino, a una frecuencia máxima de 34Hz.

La estimación de distancia a partir de las mediciones del sensor ultrasónico se realiza a partir del tiempo de retorno de los pulsos emitidos, como se expone a continuación:

Donde:

velocidad del sonido, constante, de valor 0,0343 cm/µs.

: yiempo de retorno del pulso, medido en microsegundos (µs).

Se modeló el cálculo de altura del cuadricóptero en vuelo, estando este sujeto a inclinaciones en los ejes de Pitch, Roll y Yaw. A continuación se expone el modelo y las ecuaciones que se derivaron del mismo:

**Diagrama del modelo de estimación de altura sujeto a inclinaciones.**

=

1. **Apéndice D: Modelo dinámico del cuadricóptero.**
   1. **Modelo dinámico del cuadricóptero**

Se utilizó una versión reducida del modelo físico del Draganflyer V desarrollado en [Kivrak 2006], por el alto nivel de detalle con el que describe la dinámica de rotación y traslación en el eje ‘z’ del mismo. Las asunciones realizadas por el modelo son:

1. La estructura de fibra de carbono del cuadricóptero es rígida.
2. El cuadricóptero posee una estructura completamente simétrica, y no se presenta acoplamiento cruzado entre los ejes del mismo, por lo cual la matriz de inercia del cuadricóptero es una matriz diagonal.
3. El cuadricóptero se encuentra en condición de vuelo.
4. Los movimientos del cuadricóptero pueden ser modelados de forma efectiva respecto a dos ejes de coordenadas: uno centrado en el suelo, debajo del cuadricóptero, y que representa el marco de referencia inercial de la tierra; y otro ubicado en el centro de gravedad del cuadricóptero.
5. La relación (voltaje aplicado) – (fuerza de empuje ejercida por las hélices) de cada uno de los motores del cuadricóptero es lineal.
6. El aleteo de las hélices, el efecto suelo, la fricción del aire, y el retardo característico de los motores del cuadricóptero tienen un efecto despreciable sobre la dinámica de vuelo del mismo en espacios cerrados.

La matriz de estados del sistema viene dada por:

Las ecuaciones que describen la dinámica del cuadricóptero son:

* .
* .
* .
* .

Así mismo, se utilizó el modelo lineal de la relación (voltaje aplicado) - (fuerza de empuje ejercida por las hélices) de cada uno de los motores del cuadricóptero Draganflyer V, desarrollado en [Kivrak 2006]. El modelo en cuestión es descrito mediante las siguientes ecuaciones:

Finalmente, los momentos de fuerza en los ejes x, y, z del cuadricóptero vienen dados por:

El convenio de numeración y la fuerza ejercida por cada motor, junto a las variables de estado del cuadricóptero se expone a continuación:



Ilustración 4.23: Diagrama simplificado del modelo dinámico del cuadricóptero Draganflyer V.

Fuente: [Kivrak 2006]

* 1. **Linealización del modelo dinámico**

Partiendo de la asunción de que las fuerzas externas en los ejes X e Y del cuadricóptero son despreciables, se puede representar la dinámica del cuadricóptero como la función vectorial f(X,U), la cual integra las ecuaciones del modelo físico del mismo; y las entradas de control del sistema por el vector . Luego:

Se llevó al cabo el cálculo de los parámetros de inercia del cuadricóptero, como es expuesto en el *Apéndice A*, y posteriormente, se linealizó el modelo dinámico descrito por el vector , mediante el cálculo de su matriz Jacobiana respecto al vector de estados y al vector de entradas de control Las matrices obtenidas se evaluaron en un punto de equilibrio , para obtener una representación del sistema de la forma , donde es una matriz que describe el comportamiento del sistema alrededor de un punto de equilibrio , y es una matriz que describe el comportamiento del sistema ante perturbaciones.

Se seleccionó el vector , para asegurar que el modelo lineal represente el comportamiento del cuadricóptero en vuelo, con angulos, velocidades angulares, y velocidades lineales iguales a cero (0), y una altura inicial de un (1) metro, cercana a la altura máxima a alcanzar por el cuadricóptero desarrollado en el presente Trabajo Especial de Grado. Las matrices A y B calculadas se exponen a continuación:

Se verificó la controlabilidad del sistema en base al rango de la matriz de controlabilidad , la cual puede hallarse a partir de las matrices A y B, como se expone a continuación:

Se obtuvo que el rango de la matriz es igual diez (10), el número de estados del sistema, por lo cual todos los estados del sistema con controlables.

Se modeló la estimación de estado del cuadricóptero como un sistema de ecuaciones diferenciales lineales de la forma , donde Y es la observación sobre el estado del cuadricóptero realizada a partir de los sensores disponibles en el mismo, X es el estado del cuadricóptero, y C es la matriz de salida del cuadricóptero, que representa el arreglo de sensores disponibles. Los sensores existentes en el cuadricóptero permiten la estimación de las velocidades y posiciones angulares, y la velocidad lineal y posición en el eje z, a partir de lo cual se modeló la matriz C como se presenta a continuación:

Para todo X, la estimación de estado del cuadricóptero realizada a partir de los sensores del mismo viene dada por:

Se verificó la observabilidad del sistema en base al rango de la matriz de observabilidad , la cual puede hallarse a partir de las matrices A y B, como se expone a continuación:

Se obtuvo que el rango de la matriz es igual a ocho (8), siendo diez (10) el número de estados del sistema, por lo cual dos estados del cuadricóptero no podrán ser estimados a partir del arreglo de sensores disponible. En particular, dichos estados son las velocidades lineales en los ejes x e y.

1. **Apéndice E: Código de estimación, control y comunicación a bordo del cuadricóptero.**

**#include <FiltroLP\_Giroscopio.h>**

**#include <FiltroLP\_Acelerometro.h>**

**#include <FiltroMediaMovil\_Acelerometro.h>**

**#include <FiltroMediaMovil\_Giroscopio.h>**

**#include <Wire.h>**

**#include <L3G.h>**

**#include <LSM303.h>**

**#include <PID\_v1.h>**

**#include <NewPing.h>**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* CONSTANTES y VARIABLES \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**//MOTORES:**

**#define PUERTOMOTORDERECHO 5 //puerto de PWM del motor derecho**

**#define PUERTOMOTORIZQUIERDO 9 //puerto de PWM del motor izquierdo**

**#define PUERTOMOTORINFERIOR 10 //puerto de PWM del motor inferior**

**#define PUERTOMOTORSUPERIOR 11 //puerto de PWM del motor superior**

**#define PWM\_MAXIMO 240 //maximo PWM que puede enviar el arduino a los motores**

**int velocidadBasePWM = 120;**

**char modoEjecucion = '\_';**

**int motorDerecho = 0;**

**int motorIzquierdo = 0;**

**int motorDelantero = 0;**

**int motorTrasero = 0;**

**//FIN MOTORES**

**//CODIGOS DE COMUNICACION:**

**#define modoTelemetriaTotal 1**

**#define DT\_envioDatosTelemetriaTotal 5**

**#define DT\_envioDatosEstado 30**

**#define LED\_ENCENDIDO 13**

**#define CODIGO\_INICIO\_MENSAJE 255**

**#define CODIGO\_ENCENDIDO 0**

**#define CODIGO\_MOVIMIENTO 1**

**#define CODIGO\_ACK 6**

**#define CODIGO\_ESTADO 7**

**#define CODIGO\_TELEMETRIA\_TOTAL 8**

**#define MAXIMO\_ANGULO\_COMANDO 70**

**unsigned char headerMensaje;**

**unsigned char codigoRecibido;**

**unsigned char comandoEncendidoRecibido;**

**unsigned char comandoPitch;**

**unsigned char comandoRoll;**

**unsigned char comandoAltura;**

**unsigned char checksum;**

**unsigned char mensajeEstado[15];**

**unsigned char mensajeTelemetriaTotal[42];**

**unsigned char ack[4];**

**long tiempoUltimoEnvio = 0;**

**//FIN CODIGOS DE COMUNICACION**

**//ULTRASONIDO:**

**#define USPIN 15 //puerto de datos del ultradonido.**

**#define ALTURA\_MAXIMA 150**

**#define INCREMENTO\_ALTURA\_COMANDO 5 ///////////////////////////// CAMBIAR A 5????**

**NewPing sonar(USPIN, USPIN, ALTURA\_MAXIMA);**

**long tiempoUltimoMuestreoAltura = 0;**

**unsigned int uS;**

**double alturaDeseada = 0;**

**double distancia = 0;**

**double USAltura = 0; // Distancia medida por el sensor de ultrasonido**

**double correccionAltura = 0;**

**double covarianzaProcesoFisicoAltura = 0.1;**

**double covarianzaRuidoSensorAltura = 10.0;**

**double estimacionAltura = 0.0;**

**double covarianzaRuidoEstimacionAltura = 3.0;**

**double gananciaKalman = 0.0;**

**double velocidad\_Z = 0.0;**

**double Z\_previo = 0.0;**

**//FIN ULTRASONIDO**

**//IMU**

**#define ToRad(x) ((x)\*0.01745329252) // \*pi/180**

**#define ToDeg(x) ((x)\*57.2957795131) // \*180/pi**

**#define G\_GYRO 0.00875**

**#define G\_ACC 0.015874**

**#define K\_COMP 0.97**

**#define DT\_sensor\_altura 29**

**#define DT\_acelerometro 1**

**#define DT\_giroscopio 6**

**#define DT\_PID\_altura 50**

**#define DT\_PID\_posicionAngular 20**

**#define DT\_PID\_velocidadAngular 6**

**L3G gyro;**

**LSM303 compass;**

**char report[80];**

**double yaw\_offset = 0;**

**double pitch\_offset = 0;**

**double roll\_offset = 0;**

**double G\_offsetYPR [3] = {**

**0, 0, 0**

**};**

**double A\_offsetYPR [3] = {**

**14946, -355, -1957**

**};**

**double anguloDeseadoYPR[3] = {**

**0, 0, 0**

**};**

**double G\_velocidadYPR\_filtrada[3] = {**

**0, 0, 0**

**};**

**double G\_velocidadYPR[3] = {**

**0, 0, 0**

**};**

**double G\_anguloYPR[3] = {**

**0, 0, 0**

**};**

**double A\_aceleracionYPR[3] = {**

**0, 0, 0**

**};**

**double A\_aceleracionYPR\_filtrada[3] = {**

**0, 0, 0**

**};**

**double A\_anguloYPR[3] = {**

**0, 0, 0**

**};**

**double A\_anguloYPR\_filtrado[3] = {**

**0, 0, 0**

**};**

**double anguloYPR[3] = {**

**0, 0, 0**

**};**

**double anguloYPR\_filtrado[3] = {**

**0, 0, 0**

**};**

**double velocidadDeseadaYPR[3] = {**

**0, 0, 0**

**};**

**double correccionPWM\_YPR[3] = {**

**0, 0, 0**

**};**

**FiltroMediaMovil\_Giroscopio filtroVelocidadYPR [3];**

**FiltroMediaMovil\_Acelerometro filtroAceleracionYPR [3];**

**double DT = 0;**

**long tiempoUltimoMuestreoGiroscopio = 0;**

**long tiempoUltimoMuestreoAcelerometro = 0;**

**long tiempoUltimoMuestreoAngulos = 0;**

**//FIN IMU**

**//SISTEMAS DE CONTROL PID**

**PID PID\_vAngular\_Yaw(&G\_velocidadYPR[0], &correccionPWM\_YPR[0], &velocidadDeseadaYPR[0], 0, 0, 0, DIRECT);**

**PID PID\_vAngular\_Pitch(&G\_velocidadYPR\_filtrada[1], &correccionPWM\_YPR[1], &velocidadDeseadaYPR[1], 0, 0, 0, DIRECT);**

**PID PID\_vAngular\_Roll(&G\_velocidadYPR\_filtrada[2], &correccionPWM\_YPR[2], &velocidadDeseadaYPR[2], 0, 0, 0, DIRECT);**

**PID PID\_pAngular\_Yaw(&anguloYPR\_filtrado[0], &velocidadDeseadaYPR[0], &anguloDeseadoYPR[0], 0, 0, 0, DIRECT);**

**PID PID\_pAngular\_Pitch(&anguloYPR\_filtrado[1], &velocidadDeseadaYPR[1], &anguloDeseadoYPR[1], 0, 0, 0, DIRECT);**

**PID PID\_pAngular\_Roll(&anguloYPR\_filtrado[2], &velocidadDeseadaYPR[2], &anguloDeseadoYPR[2], 0, 0, 0, REVERSE);**

**PID PID\_altura(&estimacionAltura, &correccionAltura, &alturaDeseada, 0, 0, 0, DIRECT);**

**//FIN PID**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FIN DE CONSTANTES y VARIABLES \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**void setup() {**

**// Modo FAST PWM en los pines 9, 10 y 11 //**

**TCCR2A = \_BV(COM2A1) | \_BV(COM2B1) | \_BV(WGM21) | \_BV(WGM20);**

**TCCR2B = \_BV(CS22);**

**TCCR1A = \_BV(COM1A1) | \_BV(COM1B1) | \_BV(WGM10);**

**TCCR1B = \_BV(CS11) | \_BV(CS10) | \_BV(WGM12);**

**///////////////////////////////////////////**

**// Configuracion de los puertos para sensores, motores y bandera de encendido//**

**pinMode(LED\_ENCENDIDO, OUTPUT);**

**pinMode(PUERTOMOTORDERECHO, OUTPUT);**

**pinMode(PUERTOMOTORIZQUIERDO, OUTPUT);**

**pinMode(PUERTOMOTORSUPERIOR, OUTPUT);**

**pinMode(PUERTOMOTORINFERIOR, OUTPUT);**

**analogWrite(PUERTOMOTORDERECHO, 0);**

**analogWrite(PUERTOMOTORIZQUIERDO, 0);**

**analogWrite(PUERTOMOTORSUPERIOR, 0);**

**analogWrite(PUERTOMOTORINFERIOR, 0);**

**//////////////////////////////////////////////////////////**

**// Inicializacion de la comunicacion Serial, I2C y acelerometro/giroscopio //**

**if (modoTelemetriaTotal == 1)**

**{**

**Serial.begin(115200);**

**}**

**else**

**{**

**Serial.begin(38400);**

**}**

**Wire.begin();**

**if (!gyro.init())**

**{**

**Serial.println("Failed to autodetect gyro type!");**

**while (1);**

**}**

**gyro.enableDefault();**

**compass.init();**

**compass.enableDefault();**

**CalcularOffsetGiroscopio();**

**// CalcularOffsetAcelerometro();**

**/////////////////////////////////////////////////////////////////////////////**

**// Parametros de los Algoritmos PID //**

**PID\_pAngular\_Yaw.SetSampleTime(DT\_PID\_posicionAngular);**

**PID\_pAngular\_Pitch.SetSampleTime(DT\_PID\_posicionAngular);**

**PID\_pAngular\_Roll.SetSampleTime(DT\_PID\_posicionAngular);**

**PID\_vAngular\_Yaw.SetSampleTime(DT\_PID\_velocidadAngular);**

**PID\_vAngular\_Pitch.SetSampleTime(DT\_PID\_velocidadAngular);**

**PID\_vAngular\_Roll.SetSampleTime(DT\_PID\_velocidadAngular);**

**PID\_altura.SetSampleTime(DT\_PID\_altura);**

**PID\_pAngular\_Yaw.SetOutputLimits(-180.0, 180.0);**

**PID\_pAngular\_Pitch.SetOutputLimits(-180.0, 180.0);**

**PID\_pAngular\_Roll.SetOutputLimits(-180.0, 180.0);**

**PID\_vAngular\_Yaw.SetOutputLimits(-PWM\_MAXIMO, PWM\_MAXIMO);**

**PID\_vAngular\_Pitch.SetOutputLimits(-PWM\_MAXIMO, PWM\_MAXIMO);**

**PID\_vAngular\_Roll.SetOutputLimits(-PWM\_MAXIMO, PWM\_MAXIMO);**

**PID\_altura.SetOutputLimits(-PWM\_MAXIMO, PWM\_MAXIMO);**

**PID\_pAngular\_Yaw.SetMode(AUTOMATIC);**

**PID\_pAngular\_Pitch.SetMode(AUTOMATIC);**

**PID\_pAngular\_Roll.SetMode(AUTOMATIC);**

**PID\_vAngular\_Yaw.SetMode(AUTOMATIC);**

**PID\_vAngular\_Pitch.SetMode(AUTOMATIC);**

**PID\_vAngular\_Roll.SetMode(AUTOMATIC);**

**PID\_altura.SetMode(AUTOMATIC);**

**PID\_pAngular\_Yaw.SetTunings(1, 0.01, 0);**

**PID\_vAngular\_Yaw.SetTunings(0.4, 0, 0);**

**PID\_vAngular\_Pitch.SetTunings(0.95, 0.01, 0.005);**

**PID\_vAngular\_Roll.SetTunings(0.95, 0.1, 0.005);**

**//////////////////////////////////////**

**// Inicio de conteo para manejo de frecuencia de envio de datos y DT de muestreo //**

**tiempoUltimoMuestreoAngulos = micros();**

**tiempoUltimoMuestreoAltura = millis();**

**tiempoUltimoEnvio = millis();**

**///////////////////////////////////////////////////////////////////////////////////**

**}**

**/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////**

**///////////////////////////////////////// PROCEDIMIENTO PRINCIPAL /////////////////////////////////////////**

**/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////**

**/\* El procedimiento loop se ejecuta de forma constante e ininterrumpida, y realiza llamados a las rutinas de**

**recepcion de comandos, secuencia de inicio, y secuencia de vuelo.**

**\*/**

**void loop()**

**{**

**modoEjecucion = '\_';**

**RecibirComando();**

**SecuenciaDeInicio();**

**SecuenciaDeVuelo();**

**}**

**////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////**

**///////////////////////////////////////// PROCEDIMIENTO PARA LLEVAR A CABO ESTIMACION DE ESTADO, TELEMETRIA E INICIO PAULATINO DE LOS MOTORES DEL CUADRICOPTERO /////////////////////////////////////////**

**////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////**

**/\***

**Se realiza muestreo de los sensores en espera de recibir un comando de encendido de motores desde la PC. En cuanto se encienden los motores se ejecuta una secuencia de encendido**

**gradual de los motores.**

**\*/**

**void SecuenciaDeInicio()**

**{**

**//Ciclo para realizar muestreo y telemetria del cuadricoptero de forma constante, en espera de comandos de encendido de motores desde la PC.**

**int i = 0;**

**while (i < 50)**

**{**

**FiltroComplementario();**

**CalcularAltura();**

**EnviarMensajesTelemetriaPC();**

**RecibirComando();**

**anguloDeseadoYPR[1] = 0;**

**anguloDeseadoYPR[2] = 0;**

**alturaDeseada = 0;**

**i++;**

**}**

**//Ciclo para iniciar lentamente los motores, hasta la mitad de la velocidad base de PWM por defecto, con un retardo de 1ms por cada escalon de velocidad.**

**//Se ejecuta si y solo si ha sido enviado un mensaje de encendido de motores desde la PC. (modoEjecucion == 'T')**

**i = 0;**

**if (modoEjecucion != '\_')**

**{**

**while (i < velocidadBasePWM / 2)**

**{**

**if (modoEjecucion == '\_')**

**{**

**motorDerecho = 0;**

**motorIzquierdo = 0;**

**motorDelantero = 0;**

**motorTrasero = 0;**

**}**

**if (modoEjecucion == 'T')**

**{**

**motorDerecho = i;**

**motorIzquierdo = i;**

**motorDelantero = i;**

**motorTrasero = i;**

**PIDAltura();**

**PID\_VelocidadAngular();**

**AplicarPWMmotores(i);**

**}**

**FiltroComplementario();**

**CalcularAltura();**

**PID\_PosicionAngular();**

**PID\_VelocidadAngular();**

**AplicarPWMmotores(velocidadBasePWM);**

**EnviarMensajesTelemetriaPC();**

**RecibirComando();**

**i++;**

**delay(1);**

**}**

**}**

**else**

**{**

**analogWrite(PUERTOMOTORDERECHO, 0);**

**analogWrite(PUERTOMOTORIZQUIERDO, 0);**

**analogWrite(PUERTOMOTORSUPERIOR, 0);**

**analogWrite(PUERTOMOTORINFERIOR, 0);**

**}**

**}**

**////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////**

**///////////////////////////////////////// PROCEDIMIENTO PARA LLEVAR A CABO ESTIMACION DE ESTADO, CONTROL Y TELEMETRIA DEL CUADRICOPTERO EN VUELO /////////////////////////////////////////////////**

**////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////**

**/\* Ciclo que se ejecuta mientras no se envie comando de apagado de los motores (mientras modoEjecucion sea distinto de '\_').**

**Secuencialmente, se reciben comandos, se estima el estado del cuadricoptero (posicion angular, velocidad angular y altura), se ejecutan los algoritmos PID, se aplican sus salidas a los**

**motores, y se envian mensajes de telemetria a la PC.**

**\*/**

**void SecuenciaDeVuelo()**

**{**

**while (modoEjecucion != '\_')**

**{**

**RecibirComando();**

**FiltroComplementario();**

**CalcularAltura();**

**//PIDAltura();**

**PID\_PosicionAngular();**

**PID\_VelocidadAngular();**

**AplicarPWMmotores(velocidadBasePWM);**

**EnviarMensajesTelemetriaPC();**

**}**

**}**

**////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////**

**///////////////////////////////////////// PROCEDIMIENTO PARA ESTIMAR LA MEDIA DE RUIDO POR CONDICIONES INICIALES DE VOLTAJE Y TEMPERATURA DEL GIROSCOPIO /////////////////////////////////////////**

**////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////**

**/\***

**Se realizan 500 muestreos del giroscopio y se calcula la media de los mismos, la cual representa el offset del sensor debido a las condiciones iniciales de voltaje de alimentacion y temperatura.**

**\*/**

**void CalcularOffsetGiroscopio() {**

**int numMuestras = 500;**

**for (int n = 0; n < numMuestras ; n++) {**

**gyro.read();**

**G\_offsetYPR[0] += gyro.g.z;**

**G\_offsetYPR[1] += gyro.g.x;**

**G\_offsetYPR[2] += gyro.g.y;**

**}**

**G\_offsetYPR [0] = G\_offsetYPR[0] / numMuestras;**

**G\_offsetYPR [1] = G\_offsetYPR[1] / numMuestras;**

**G\_offsetYPR [2] = G\_offsetYPR[2] / numMuestras;**

**}**

**///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////**

**///////////////////////////////////////// PROCEDIMIENTO PARA ESTIMAR LA MEDIA DE RUIDO POR INCLINACION DEL ACELEROMETRO SOBRE EL CHASIS /////////////////////////////////////////**

**///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////**

**/\***

**Se realizan 500 muestreos del acelerometro y se calcula la media de los mismos, la cual representa el offset del sensor debido a la inclinacion del mismo sobre el chasis del cuadricoptero.**

**\*/**

**void CalcularOffsetAcelerometro() {**

**int numMuestras = 500;**

**for (int n = 0; n < numMuestras ; n++) {**

**compass.read();**

**A\_offsetYPR[0] += compass.a.z;**

**A\_offsetYPR[1] += compass.a.y;**

**A\_offsetYPR[2] += compass.a.x;**

**}**

**A\_offsetYPR [0] = A\_offsetYPR[0] / numMuestras;**

**A\_offsetYPR [1] = A\_offsetYPR[1] / numMuestras;**

**A\_offsetYPR [2] = A\_offsetYPR[2] / numMuestras;**

**}**

**///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////**

**///////////////////////////////////////// PROCEDIMIENTO PARA ESTIMAR LA VELOCIDAD ANGULAR, Y LA POSICION ANGULAR MEDIANTE FILTRO COMPLEMENTARIO /////////////////////////////////////////**

**///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////**

**/\* El procedimiento FiltroComplementario() adquiere datos desde el acelerometro y el giroscopio, los somete a filtrado por medias moviles, y calcula la posicion angular del**

**cuadricoptero en Yaw, Pitch y Roll mediante filtro complementario. Se maneja estrictamente la frecuencia de muestreo de los sensores a la hora de muestrear los datos y**

**filtrar los datos.**

**\*/**

**void FiltroComplementario() {**

**gyro.read();**

**compass.read();**

**DT = (double)(micros() - tiempoUltimoMuestreoAngulos) / 1000000;**

**if (millis() - tiempoUltimoMuestreoGiroscopio >= DT\_giroscopio)**

**{**

**G\_velocidadYPR[0] = (double) ((gyro.g.z - G\_offsetYPR[0]) \* G\_GYRO );**

**G\_velocidadYPR[1] = (double) ((gyro.g.x - G\_offsetYPR[1]) \* G\_GYRO );**

**G\_velocidadYPR[2] = (double) ((gyro.g.y - G\_offsetYPR[2]) \* G\_GYRO );**

**G\_velocidadYPR\_filtrada[0] = filtroVelocidadYPR [0].step ((double) G\_velocidadYPR[0]);**

**G\_velocidadYPR\_filtrada[1] = filtroVelocidadYPR [1].step ((double) G\_velocidadYPR[1]);**

**G\_velocidadYPR\_filtrada[2] = filtroVelocidadYPR [2].step ((double) G\_velocidadYPR[2]);**

**tiempoUltimoMuestreoGiroscopio = millis();**

**anguloYPR[0] = (double) (anguloYPR[0] + G\_velocidadYPR[0] \* DT);**

**anguloYPR[1] = (double) (K\_COMP \* (anguloYPR[1] + G\_velocidadYPR[1] \* DT) + (1 - K\_COMP) \* A\_anguloYPR[1]);**

**anguloYPR[2] = (double) (K\_COMP \* (anguloYPR[2] + G\_velocidadYPR[2] \* DT) + (1 - K\_COMP) \* A\_anguloYPR[2]);**

**anguloYPR\_filtrado[0] = (double) (anguloYPR\_filtrado[0] + G\_velocidadYPR\_filtrada[0] \* DT);**

**anguloYPR\_filtrado[1] = (double) (K\_COMP \* (anguloYPR\_filtrado[1] + G\_velocidadYPR\_filtrada[1] \* DT) + (1 - K\_COMP) \* A\_anguloYPR\_filtrado[1]);**

**anguloYPR\_filtrado[2] = (double) (K\_COMP \* (anguloYPR\_filtrado[2] + G\_velocidadYPR\_filtrada[2] \* DT) + (1 - K\_COMP) \* A\_anguloYPR\_filtrado[2]);**

**}**

**if (millis() - tiempoUltimoMuestreoAcelerometro >= DT\_acelerometro)**

**{**

**A\_aceleracionYPR[0] = (double) (compass.a.z) \* G\_ACC;**

**A\_aceleracionYPR[1] = (double) (compass.a.y - A\_offsetYPR[1]) \* G\_ACC;**

**A\_aceleracionYPR[2] = (double) (compass.a.x - A\_offsetYPR[2]) \* G\_ACC;**

**A\_aceleracionYPR\_filtrada[0] = filtroAceleracionYPR[0].step((double) A\_aceleracionYPR[0]);**

**A\_aceleracionYPR\_filtrada[1] = filtroAceleracionYPR[1].step((double) A\_aceleracionYPR[1]);**

**A\_aceleracionYPR\_filtrada[2] = filtroAceleracionYPR[2].step((double) A\_aceleracionYPR[2]);**

**A\_anguloYPR[0] = 0;**

**A\_anguloYPR[1] = (double) atan2(A\_aceleracionYPR[1], sqrt(A\_aceleracionYPR[0] \* A\_aceleracionYPR[0] + A\_aceleracionYPR[2] \* A\_aceleracionYPR[2]));**

**A\_anguloYPR[1] = ToDeg(A\_anguloYPR[1]);**

**A\_anguloYPR[2] = (double) atan2(A\_aceleracionYPR[2], sqrt(A\_aceleracionYPR[0] \* A\_aceleracionYPR[0] + A\_aceleracionYPR[1] \* A\_aceleracionYPR[1]));**

**A\_anguloYPR[2] = ToDeg(A\_anguloYPR[2]);**

**A\_anguloYPR\_filtrado[0] = 0;**

**A\_anguloYPR\_filtrado[1] = (double) atan2(A\_aceleracionYPR\_filtrada[1], sqrt(A\_aceleracionYPR\_filtrada[0] \* A\_aceleracionYPR\_filtrada[0] + A\_aceleracionYPR\_filtrada[2] \* A\_aceleracionYPR\_filtrada[2]));**

**A\_anguloYPR\_filtrado[1] = ToDeg(A\_anguloYPR\_filtrado[1]);**

**A\_anguloYPR\_filtrado[2] = (double) atan2(A\_aceleracionYPR\_filtrada[2], sqrt(A\_aceleracionYPR\_filtrada[0] \* A\_aceleracionYPR\_filtrada[0] + A\_aceleracionYPR\_filtrada[1] \* A\_aceleracionYPR\_filtrada[1]));**

**A\_anguloYPR\_filtrado[2] = ToDeg(A\_anguloYPR\_filtrado[2]);**

**tiempoUltimoMuestreoAcelerometro = millis();**

**}**

**anguloYPR[0] = ToRad(anguloYPR[0]);**

**anguloYPR[0] = (double) atan2(sin(anguloYPR[0]), cos(anguloYPR[0]));**

**anguloYPR[0] = ToDeg(anguloYPR[0]);**

**anguloYPR[1] = ToRad(anguloYPR[1]);**

**anguloYPR[1] = (double) atan2(sin(anguloYPR[1]), cos(anguloYPR[1]));**

**anguloYPR[1] = ToDeg(anguloYPR[1]);**

**anguloYPR[2] = ToRad(anguloYPR[2]);**

**anguloYPR[2] = (double) atan2(sin(anguloYPR[2]), cos(anguloYPR[2]));**

**anguloYPR[2] = ToDeg(anguloYPR[2]);**

**anguloYPR\_filtrado[0] = ToRad(anguloYPR\_filtrado[0]);**

**anguloYPR\_filtrado[0] = (double) atan2(sin(anguloYPR\_filtrado[0]), cos(anguloYPR\_filtrado[0]));**

**anguloYPR\_filtrado[0] = ToDeg(anguloYPR\_filtrado[0]);**

**anguloYPR\_filtrado[1] = ToRad(anguloYPR\_filtrado[1]);**

**anguloYPR\_filtrado[1] = (double) atan2(sin(anguloYPR\_filtrado[1]), cos(anguloYPR\_filtrado[1]));**

**anguloYPR\_filtrado[1] = ToDeg(anguloYPR\_filtrado[1]);**

**anguloYPR\_filtrado[2] = ToRad(anguloYPR\_filtrado[2]);**

**anguloYPR\_filtrado[2] = (double) atan2(sin(anguloYPR\_filtrado[2]), cos(anguloYPR\_filtrado[2]));**

**anguloYPR\_filtrado[2] = ToDeg(anguloYPR\_filtrado[2]);**

**tiempoUltimoMuestreoAngulos = micros();**

**}**

**//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////**

**///////////////////////////////////////// PROCEDIMIENTOS PARA CALCULAR LA ALTURA DEL CUADRICOPTERO /////////////////////////////////////////**

**//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////**

**/\***

**\* En CalcularAltura() se calcula la distancia del sensor ultrasonico al suelo emitiendo un pulso, y calculando el tiempo de respuesta del mismo.**

**Conociendo previamente la velocidad del sonido, puede calcularse la distancia que recorrió la onda, ida y vuelta. El procedimiento se apoya en**

**la libreria NewPing para Arduino, que utiliza rutinas con temporizadores.**

**\* En FiltroKalmanAltura() se estima la distancia real del sensor ultrasonico al suelo mediante un filtro de Kalman de primer orden.**

**\*/**

**void CalcularAltura()**

**{**

**if (millis() - tiempoUltimoMuestreoAltura > DT\_sensor\_altura)**

**{**

**uS = sonar.ping();**

**distancia = (double) (uS / US\_ROUNDTRIP\_CM);**

**if ((distancia > 0) && (distancia < ALTURA\_MAXIMA))**

**{**

**USAltura = distancia;**

**FiltroKalmanAltura();**

**mensajeEstado[8] = estimacionAltura;**

**velocidad\_Z = estimacionAltura - Z\_previo;**

**Z\_previo = estimacionAltura;**

**}**

**tiempoUltimoMuestreoAltura = millis();**

**}**

**}**

**void FiltroKalmanAltura()**

**{**

**covarianzaRuidoEstimacionAltura = covarianzaRuidoEstimacionAltura + covarianzaProcesoFisicoAltura;**

**gananciaKalman = covarianzaRuidoEstimacionAltura / (covarianzaRuidoEstimacionAltura + covarianzaRuidoSensorAltura);**

**estimacionAltura = estimacionAltura + gananciaKalman \* (USAltura - estimacionAltura);**

**covarianzaRuidoEstimacionAltura = (1 - gananciaKalman) \* covarianzaRuidoEstimacionAltura;**

**}**

**///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////**

**///////////////////////////////////////// RUTINAS PARA EJECUTAR LOS PID DE POSICION ANGULAR, VELOCIDAD ANGULAR Y ALTURA /////////////////////////////////////////**

**///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////**

**void PID\_PosicionAngular()**

**{**

**PID\_pAngular\_Yaw.Compute();**

**// PID\_pAngular\_Pitch.Compute();**

**// PID\_pAngular\_Roll.Compute();**

**}**

**void PID\_VelocidadAngular()**

**{**

**PID\_vAngular\_Yaw.Compute();**

**PID\_vAngular\_Pitch.Compute();**

**PID\_vAngular\_Roll.Compute();**

**}**

**void PIDAltura()**

**{**

**PID\_altura.Compute();**

**}**

**///////////////////////////////////////// RUTINA DE APLICACION DE PWM A LOS MOTORES /////////////////////////////////////////**

**// Valida el estado de la variable modoEjecucion:**

**// \* Si es igual a '\_', envía PWM=0 a todos los puertos.**

**// \* Si es igual a 'T', se calcula el valor de PWM a enviar a los motores a partir de la velocidad base del PWM establecida y las**

**// salidas de los sistemas de control PID de velocidad angular y altura.**

**//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////**

**void AplicarPWMmotores(int velocidadMotoresPWM)**

**{**

**if (modoEjecucion == '\_')**

**{**

**motorDerecho = 0;**

**motorIzquierdo = 0;**

**motorDelantero = 0;**

**motorTrasero = 0;**

**}**

**if (modoEjecucion == 'T')**

**{**

**motorDerecho = velocidadMotoresPWM - correccionPWM\_YPR[2] + correccionPWM\_YPR[0] + correccionAltura;**

**motorIzquierdo = velocidadMotoresPWM + correccionPWM\_YPR[2] + correccionPWM\_YPR[0] + correccionAltura;**

**motorDelantero = velocidadMotoresPWM + correccionPWM\_YPR[1] - correccionPWM\_YPR[0] + correccionAltura;**

**motorTrasero = velocidadMotoresPWM - correccionPWM\_YPR[1] - correccionPWM\_YPR[0] + correccionAltura;**

**}**

**if (motorDelantero > PWM\_MAXIMO)**

**{**

**motorDelantero = PWM\_MAXIMO;**

**}**

**else if (motorDelantero < 0)**

**{**

**motorDelantero = 0;**

**}**

**if (motorTrasero > PWM\_MAXIMO)**

**{**

**motorTrasero = PWM\_MAXIMO;**

**}**

**else if (motorTrasero < 0)**

**{**

**motorTrasero = 0;**

**}**

**if (motorDerecho > PWM\_MAXIMO)**

**{**

**motorDerecho = PWM\_MAXIMO;**

**}**

**else if (motorDerecho < 0)**

**{**

**motorDerecho = 0;**

**}**

**if (motorIzquierdo > PWM\_MAXIMO)**

**{**

**motorIzquierdo = PWM\_MAXIMO;**

**}**

**else if (motorIzquierdo < 0)**

**{**

**motorIzquierdo = 0;**

**}**

**analogWrite(PUERTOMOTORDERECHO, motorDerecho);**

**analogWrite(PUERTOMOTORIZQUIERDO, motorIzquierdo);**

**analogWrite(PUERTOMOTORSUPERIOR, motorDelantero);**

**analogWrite(PUERTOMOTORINFERIOR, motorTrasero);**

**}**

**////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////**

**///////////////////////////////////////// RUTINA DE PREPARACION DEL PAQUETE DE MENSAJE DE ESTADO /////////////////////////////////////////**

**////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////**

**/\* Procedimiento para preparar paquetes de mensaje de estado. (14 bytes)**

**posicion 0 = HEADER (255)**

**posicion 1 = CODIGO DE MENSAJE (7)**

**posicion 2 = POSICION ANGULAR YAW (Valor positivo o == 0)**

**posicion 3 = POSICION ANGULAR YAW (Valor negativo)**

**posicion 4 = POSICION ANGULAR PITCH**

**posicion 5 = POSICION ANGULAR ROLL**

**posicion 6 = VELOCIDAD ANGULAR YAW (Valor positivo o == 0)**

**posicion 7 = VELOCIDAD ANGULAR YAW (Valor negativo)**

**posicion 8 = VELOCIDAD ANGULAR PITCH (Valor positivo o == 0)**

**posicion 9 = VELOCIDAD ANGULAR PITCH (Valor negativo)**

**posicion 10 = VELOCIDAD ANGULAR ROLL (Valor positivo o == 0)**

**posicion 11 = VELOCIDAD ANGULAR ROLL (Valor negativo)**

**posicion 12 = ALTURA**

**posicion 13 = CHECKSUM (HECHO CON XOR DE LOS BYTES 0 AL 12)**

**\*/**

**void PrepararPaqueteMensajeEstado()**

**{**

**mensajeEstado[0] = CODIGO\_INICIO\_MENSAJE; //HEADER**

**mensajeEstado[1] = CODIGO\_ESTADO; //Codigo del mensaje**

**if (anguloYPR\_filtrado[0] >= 0)**

**{**

**mensajeEstado[2] = anguloYPR\_filtrado[0];**

**mensajeEstado[3] = 0;**

**}**

**else**

**{**

**mensajeEstado[3] = abs(anguloYPR\_filtrado[0]);**

**mensajeEstado[2] = 0;**

**}**

**mensajeEstado[4] = anguloYPR\_filtrado[1] + 90;**

**mensajeEstado[5] = anguloYPR\_filtrado[2] + 90;**

**if (G\_velocidadYPR[0] >= 0)**

**{**

**mensajeEstado[6] = G\_velocidadYPR[0];**

**mensajeEstado[7] = 0;**

**}**

**else**

**{**

**mensajeEstado[7] = abs(G\_velocidadYPR[0]);**

**mensajeEstado[6] = 0;**

**}**

**if (G\_velocidadYPR\_filtrada[1] >= 0)**

**{**

**mensajeEstado[8] = G\_velocidadYPR\_filtrada[1];**

**mensajeEstado[9] = 0;**

**}**

**else**

**{**

**mensajeEstado[9] = abs(G\_velocidadYPR\_filtrada[1]);**

**mensajeEstado[8] = 0;**

**}**

**if (G\_velocidadYPR\_filtrada[2] >= 0)**

**{**

**mensajeEstado[10] = G\_velocidadYPR\_filtrada[2];**

**mensajeEstado[11] = 0;**

**}**

**else**

**{**

**mensajeEstado[11] = abs(G\_velocidadYPR\_filtrada[2]);**

**mensajeEstado[10] = 0;**

**}**

**mensajeEstado[12] = velocidadBasePWM;**

**// mensajeEstado[12] = estimacionAltura;**

**// mensajeEstado[12] = alturaDeseada;**

**if (modoEjecucion == 'T')**

**{**

**mensajeEstado[13] = 1;**

**}**

**else**

**{**

**mensajeEstado[13] = 0;**

**}**

**mensajeEstado[14] = (mensajeEstado[0] ^ mensajeEstado[1] ^ mensajeEstado[2] ^ mensajeEstado[3] ^ mensajeEstado[4] ^ mensajeEstado[5] ^ mensajeEstado[6] ^ mensajeEstado[7] ^ mensajeEstado[8] ^ mensajeEstado[9] ^ mensajeEstado[10] ^ mensajeEstado[11] ^ mensajeEstado[12] ^ mensajeEstado[13]); //CHECKSUM**

**}**

**/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////**

**///////////////////////////////////////// RUTINA DE PREPARACION DEL PAQUETE DE MENSAJE DE TELEMETRIA TOTAL /////////////////////////////////////////**

**/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////**

**/\* Procedimiento para preparar paquetes de mensaje de telemetria total. (42 bytes)**

**posicion 0 = HEADER (255)**

**posicion 1 = CODIGO DE MENSAJE (7)**

**posicion 2 = POSICION ANGULAR YAW (Valor positivo o == 0)**

**posicion 3 = POSICION ANGULAR YAW (Valor negativo)**

**posicion 4 = POSICION ANGULAR PITCH (Valor positivo o == 0)**

**posicion 5 = POSICION ANGULAR PITCH (Valor negativo)**

**posicion 6 = POSICION ANGULAR ROLL (Valor positivo o == 0)**

**posicion 7 = POSICION ANGULAR ROLL (Valor negativo)**

**posicion 8 = VELOCIDAD ANGULAR YAW (Valor positivo o == 0)**

**posicion 9 = VELOCIDAD ANGULAR YAW (Valor negativo)**

**posicion 10 = VELOCIDAD ANGULAR PITCH (Valor positivo o == 0)**

**posicion 11 = VELOCIDAD ANGULAR PITCH (Valor negativo)**

**posicion 12 = VELOCIDAD ANGULAR ROLL (Valor positivo o == 0)**

**posicion 13 = VELOCIDAD ANGULAR ROLL (Valor negativo)**

**posicion 14 = VELOCIDAD ANGULAR YAW FILTRADA (Valor positivo o == 0)**

**posicion 15 = VELOCIDAD ANGULAR YAW FILTRADA (Valor negativo)**

**posicion 16 = VELOCIDAD ANGULAR PITCH FILTRADA (Valor positivo o == 0)**

**posicion 17 = VELOCIDAD ANGULAR PITCH FILTRADA (Valor negativo)**

**posicion 18 = VELOCIDAD ANGULAR ROLL FILTRADA (Valor positivo o == 0)**

**posicion 19 = VELOCIDAD ANGULAR ROLL FILTRADA (Valor negativo)**

**posicion 20 = ACELERACION LINEAL Z (Valor positivo o == 0)**

**posicion 21 = ACELERACION LINEAL Z (Valor negativo)**

**posicion 22 = ACELERACION LINEAL Y (Valor positivo o == 0)**

**posicion 23 = ACELERACION LINEAL Y (Valor negativo)**

**posicion 24 = ACELERACION LINEAL X (Valor positivo o == 0)**

**posicion 25 = ACELERACION LINEAL X (Valor negativo)**

**posicion 26 = ACELERACION LINEAL Z FILTRADA (Valor positivo o == 0)**

**posicion 27 = ACELERACION LINEAL Z FILTRADA (Valor negativo)**

**posicion 28 = ACELERACION LINEAL Y FILTRADA (Valor positivo o == 0)**

**posicion 29 = ACELERACION LINEAL Y FILTRADA (Valor negativo)**

**posicion 30 = ACELERACION LINEAL X FILTRADA (Valor positivo o == 0)**

**posicion 31 = ACELERACION LINEAL X FILTRADA (Valor negativo)**

**posicion 32 = ALTURA - POSICION EN Z**

**posicion 33 = ALTURA FILTRADA - POSICION EN Z FILTRADA**

**posicion 34 = VELOCIDAD EN Z (Valor positivo o == 0)**

**posicion 35 = VELOCIDAD EN Z (Valor negativo)**

**posicion 36 = PWM MOTOR DELANTERO**

**posicion 37 = PWM MOTOR TRASERO**

**posicion 38 = PWM MOTOR DERECHO**

**posicion 39 = PWM MOTOR IZQUIERDO**

**posicion 40 = ESTADO DE ENCENDIDO DE MOTORES**

**posicion 41 = CHECKSUM (HECHO CON XOR DE LOS BYTES 0 AL 40)**

**\*/**

**void PrepararPaqueteMensajeTelemetriaTotal()**

**{**

**mensajeTelemetriaTotal[0] = CODIGO\_INICIO\_MENSAJE; //HEADER**

**mensajeTelemetriaTotal[1] = CODIGO\_TELEMETRIA\_TOTAL; //Codigo del mensaje**

**if (anguloYPR\_filtrado[0] >= 0)**

**{**

**mensajeTelemetriaTotal[2] = anguloYPR\_filtrado[0];**

**mensajeTelemetriaTotal[3] = 0;**

**}**

**else**

**{**

**mensajeTelemetriaTotal[3] = abs(anguloYPR\_filtrado[0]);**

**mensajeTelemetriaTotal[2] = 0;**

**}**

**if (anguloYPR\_filtrado[1] >= 0)**

**{**

**mensajeTelemetriaTotal[4] = anguloYPR\_filtrado[1];**

**mensajeTelemetriaTotal[5] = 0;**

**}**

**else**

**{**

**mensajeTelemetriaTotal[5] = abs(anguloYPR\_filtrado[1]);**

**mensajeTelemetriaTotal[4] = 0;**

**}**

**if (anguloYPR\_filtrado[2] >= 0)**

**{**

**mensajeTelemetriaTotal[6] = anguloYPR\_filtrado[2];**

**mensajeTelemetriaTotal[7] = 0;**

**}**

**else**

**{**

**mensajeTelemetriaTotal[7] = abs(anguloYPR\_filtrado[2]);**

**mensajeTelemetriaTotal[6] = 0;**

**}**

**if (G\_velocidadYPR[0] >= 0)**

**{**

**mensajeTelemetriaTotal[8] = G\_velocidadYPR[0];**

**mensajeTelemetriaTotal[9] = 0;**

**}**

**else**

**{**

**mensajeTelemetriaTotal[9] = abs(G\_velocidadYPR[0]);**

**mensajeTelemetriaTotal[8] = 0;**

**}**

**if (G\_velocidadYPR[1] >= 0)**

**{**

**mensajeTelemetriaTotal[10] = G\_velocidadYPR[1];**

**mensajeTelemetriaTotal[11] = 0;**

**}**

**else**

**{**

**mensajeTelemetriaTotal[11] = abs(G\_velocidadYPR[1]);**

**mensajeTelemetriaTotal[10] = 0;**

**}**

**if (G\_velocidadYPR[2] >= 0)**

**{**

**mensajeTelemetriaTotal[12] = G\_velocidadYPR[2];**

**mensajeTelemetriaTotal[13] = 0;**

**}**

**else**

**{**

**mensajeTelemetriaTotal[13] = abs(G\_velocidadYPR[2]);**

**mensajeTelemetriaTotal[12] = 0;**

**}**

**if (G\_velocidadYPR\_filtrada[0] >= 0)**

**{**

**mensajeTelemetriaTotal[14] = G\_velocidadYPR\_filtrada[0];**

**mensajeTelemetriaTotal[15] = 0;**

**}**

**else**

**{**

**mensajeTelemetriaTotal[15] = abs(G\_velocidadYPR\_filtrada[0]);**

**mensajeTelemetriaTotal[14] = 0;**

**}**

**if (G\_velocidadYPR\_filtrada[1] >= 0)**

**{**

**mensajeTelemetriaTotal[16] = G\_velocidadYPR\_filtrada[1];**

**mensajeTelemetriaTotal[17] = 0;**

**}**

**else**

**{**

**mensajeTelemetriaTotal[17] = abs(G\_velocidadYPR\_filtrada[1]);**

**mensajeTelemetriaTotal[16] = 0;**

**}**

**if (G\_velocidadYPR\_filtrada[2] >= 0)**

**{**

**mensajeTelemetriaTotal[18] = G\_velocidadYPR\_filtrada[2];**

**mensajeTelemetriaTotal[19] = 0;**

**}**

**else**

**{**

**mensajeTelemetriaTotal[19] = abs(G\_velocidadYPR\_filtrada[2]);**

**mensajeTelemetriaTotal[18] = 0;**

**}**

**if (A\_aceleracionYPR[2] >= 0)**

**{**

**mensajeTelemetriaTotal[20] = A\_aceleracionYPR[2];**

**mensajeTelemetriaTotal[21] = 0;**

**}**

**else**

**{**

**mensajeTelemetriaTotal[21] = abs(A\_aceleracionYPR[2]);**

**mensajeTelemetriaTotal[20] = 0;**

**}**

**if (A\_aceleracionYPR[1] >= 0)**

**{**

**mensajeTelemetriaTotal[22] = A\_aceleracionYPR[1];**

**mensajeTelemetriaTotal[23] = 0;**

**}**

**else**

**{**

**mensajeTelemetriaTotal[23] = abs(A\_aceleracionYPR[1]);**

**mensajeTelemetriaTotal[22] = 0;**

**}**

**if (A\_aceleracionYPR[0] >= 0)**

**{**

**mensajeTelemetriaTotal[24] = A\_aceleracionYPR[0];**

**mensajeTelemetriaTotal[25] = 0;**

**}**

**else**

**{**

**mensajeTelemetriaTotal[25] = abs(A\_aceleracionYPR[0]);**

**mensajeTelemetriaTotal[24] = 0;**

**}**

**if (A\_aceleracionYPR\_filtrada[2] >= 0)**

**{**

**mensajeTelemetriaTotal[26] = A\_aceleracionYPR\_filtrada[2];**

**mensajeTelemetriaTotal[27] = 0;**

**}**

**else**

**{**

**mensajeTelemetriaTotal[27] = abs(A\_aceleracionYPR\_filtrada[2]);**

**mensajeTelemetriaTotal[26] = 0;**

**}**

**if (A\_aceleracionYPR\_filtrada[1] >= 0)**

**{**

**mensajeTelemetriaTotal[28] = A\_aceleracionYPR\_filtrada[1];**

**mensajeTelemetriaTotal[29] = 0;**

**}**

**else**

**{**

**mensajeTelemetriaTotal[29] = abs(A\_aceleracionYPR\_filtrada[1]);**

**mensajeTelemetriaTotal[28] = 0;**

**}**

**if (A\_aceleracionYPR\_filtrada[0] >= 0)**

**{**

**mensajeTelemetriaTotal[30] = A\_aceleracionYPR\_filtrada[0];**

**mensajeTelemetriaTotal[31] = 0;**

**}**

**else**

**{**

**mensajeTelemetriaTotal[31] = abs(A\_aceleracionYPR\_filtrada[0]);**

**mensajeTelemetriaTotal[30] = 0;**

**}**

**// mensajeTelemetriaTotal[32] = USAltura;**

**mensajeTelemetriaTotal[32] = velocidadBasePWM;**

**mensajeTelemetriaTotal[33] = estimacionAltura;**

**if (velocidad\_Z >= 0)**

**{**

**mensajeTelemetriaTotal[34] = velocidad\_Z;**

**mensajeTelemetriaTotal[35] = 0;**

**}**

**else**

**{**

**mensajeTelemetriaTotal[35] = abs(velocidad\_Z);**

**mensajeTelemetriaTotal[34] = 0;**

**}**

**mensajeTelemetriaTotal[36] = motorDelantero;**

**mensajeTelemetriaTotal[37] = motorTrasero;**

**mensajeTelemetriaTotal[38] = motorDerecho;**

**mensajeTelemetriaTotal[39] = motorIzquierdo;**

**if (modoEjecucion == 'T')**

**{**

**mensajeTelemetriaTotal[40] = 1;**

**}**

**else**

**{**

**mensajeTelemetriaTotal[40] = 0;**

**}**

**mensajeTelemetriaTotal[41] = (mensajeTelemetriaTotal[0] ^ mensajeTelemetriaTotal[1] ^ mensajeTelemetriaTotal[2] ^ mensajeTelemetriaTotal[3] ^ mensajeTelemetriaTotal[4] ^ mensajeTelemetriaTotal[5] ^ mensajeTelemetriaTotal[6] ^ mensajeTelemetriaTotal[7] ^ mensajeTelemetriaTotal[8] ^ mensajeTelemetriaTotal[9] ^ mensajeTelemetriaTotal[10] ^ mensajeTelemetriaTotal[11] ^ mensajeTelemetriaTotal[12] ^ mensajeTelemetriaTotal[13] ^ mensajeTelemetriaTotal[14] ^ mensajeTelemetriaTotal[15] ^ mensajeTelemetriaTotal[16] ^ mensajeTelemetriaTotal[17] ^ mensajeTelemetriaTotal[18] ^ mensajeTelemetriaTotal[19] ^ mensajeTelemetriaTotal[20] ^ mensajeTelemetriaTotal[21] ^ mensajeTelemetriaTotal[22] ^ mensajeTelemetriaTotal[23] ^ mensajeTelemetriaTotal[24] ^ mensajeTelemetriaTotal[25] ^ mensajeTelemetriaTotal[26] ^ mensajeTelemetriaTotal[27] ^ mensajeTelemetriaTotal[28] ^ mensajeTelemetriaTotal[29] ^ mensajeTelemetriaTotal[30] ^ mensajeTelemetriaTotal[31] ^ mensajeTelemetriaTotal[32] ^ mensajeTelemetriaTotal[33] ^ mensajeTelemetriaTotal[34] ^ mensajeTelemetriaTotal[35] ^ mensajeTelemetriaTotal[36] ^ mensajeTelemetriaTotal[37] ^ mensajeTelemetriaTotal[38] ^ mensajeTelemetriaTotal[39] ^ mensajeTelemetriaTotal[40]); //CHECKSUM**

**}**

**//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////**

**///////////////////////////////////////// PROCEDIMIENTO PARA ENVIAR MENSAJES DE TELEMETRIA A LA PC /////////////////////////////////////////**

**//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////**

**/\* Se valida el estado de la bandera modoTelemetriaTotal.**

**\* Si modoTelemetriaTotal==0, se envian mensajes de estado.**

**\* Si modoTelemetriaTotal==1, se envian mensajes de telemetria total.**

**\*/**

**void EnviarMensajesTelemetriaPC()**

**{**

**if (modoTelemetriaTotal == 0)**

**{**

**if (millis() - tiempoUltimoEnvio >= DT\_envioDatosEstado)**

**{**

**PrepararPaqueteMensajeEstado();**

**Serial.write(mensajeEstado, 14);//ENVIAR EL PAQUETE DE 14 BYTES**

**tiempoUltimoEnvio = millis();**

**}**

**}**

**else**

**{**

**if (millis() - tiempoUltimoEnvio >= DT\_envioDatosTelemetriaTotal)**

**{**

**PrepararPaqueteMensajeTelemetriaTotal();**

**Serial.write(mensajeTelemetriaTotal, 42);//ENVIAR EL PAQUETE DE 42 BYTES**

**tiempoUltimoEnvio = millis();**

**}**

**}**

**}**

**/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////**

**///////////////////////////////////////// PROCEDIMIENTO PARA RECIBIR MENSAJES DESDE LA PC /////////////////////////////////////////**

**/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////**

**/\* Se lee el puerto serial, y se verifica, de forma sucesiva que:**

**1) El primer dato recibido sea un char de codigo 255 (Header de los mensajes del protocolo).**

**2) El valor del segundo dato recibido (Codigo del mensaje), a partir de lo cual se recibe:**

**a) Comando de encendido/apagado (Codigo==0).**

**b) Comando de movimiento (Codigo==1).**

**3) Se realizar un checksum con todos los datos recibidos, y se compara con el ultimo dato recibido en cada caso (Checksum recibido).**

**4) Si el checksum recibido es igual al checksum calculado, el mensaje ha llegado correctamente, y se ejecuta el comando modificando**

**las variables modoEjecucion, velocidadDeseadaYPR y velocidadBasePWM segun corresponda. Si se recibio un mensaje de encendido, se**

**envia un mensaje de acknowledge a la PC.**

**Si en algun momento no se recibe un mensaje, o si el checksum recibido no coincide con el checksum calculado, se detiene la recepcion**

**del mensaje, y la ejecucion del procedimiento llega a su fin.**

**\*/**

**void RecibirComando()**

**{**

**if (Serial.available() > 0)**

**{**

**if (Serial.available() > 0)**

**{**

**headerMensaje = Serial.read();**

**delay(1);**

**if (headerMensaje == CODIGO\_INICIO\_MENSAJE)**

**{**

**if (Serial.available() > 0)**

**{**

**codigoRecibido = Serial.read();**

**delay(1);**

**if (codigoRecibido == CODIGO\_ENCENDIDO)**

**{**

**if (Serial.available() > 0)**

**{**

**comandoEncendidoRecibido = Serial.read();**

**delay(1);**

**if (Serial.available() > 0)**

**{**

**checksum = Serial.read();**

**delay(1);**

**if (CODIGO\_INICIO\_MENSAJE ^ CODIGO\_ENCENDIDO ^ comandoEncendidoRecibido == checksum)**

**{**

**if (comandoEncendidoRecibido == 1)**

**{**

**modoEjecucion = 'T';**

**digitalWrite(LED\_ENCENDIDO, HIGH);**

**}**

**if (comandoEncendidoRecibido == 0)**

**{**

**modoEjecucion = '\_';**

**velocidadDeseadaYPR[1] = 0.0;**

**velocidadDeseadaYPR[2] = 0.0;**

**alturaDeseada = 0;**

**digitalWrite(LED\_ENCENDIDO, LOW);**

**}**

**EnviarAcknowledge(CODIGO\_ENCENDIDO);**

**}**

**}**

**}**

**}**

**if (codigoRecibido == CODIGO\_MOVIMIENTO)**

**{**

**if (Serial.available() > 0)**

**{**

**comandoPitch = Serial.read();**

**delay(1);**

**if (Serial.available() > 0)**

**{**

**comandoRoll = Serial.read();**

**delay(1);**

**if (Serial.available() > 0)**

**{**

**comandoAltura = Serial.read();**

**delay(1);**

**if (Serial.available() > 0)**

**{**

**checksum = Serial.read();**

**delay(1);**

**if (CODIGO\_INICIO\_MENSAJE ^ CODIGO\_MOVIMIENTO ^ comandoPitch ^ comandoRoll ^ comandoAltura == checksum)**

**{**

**if ((abs(comandoPitch - MAXIMO\_ANGULO\_COMANDO) < MAXIMO\_ANGULO\_COMANDO) && (abs(comandoRoll - MAXIMO\_ANGULO\_COMANDO) < MAXIMO\_ANGULO\_COMANDO))**

**{**

**velocidadDeseadaYPR[1] = -(comandoPitch - MAXIMO\_ANGULO\_COMANDO);**

**velocidadDeseadaYPR[2] = (comandoRoll - MAXIMO\_ANGULO\_COMANDO);**

**if (comandoAltura <= PWM\_MAXIMO)**

**{**

**velocidadBasePWM = comandoAltura;**

**}**

**}**

**}**

**}**

**}**

**}**

**}**

**}**

**}**

**}**

**}**

**}**

**}**

**//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////**

**///////////////////////////////////////// PROCEDIMIENTO PARA ENVIAR MENSAJE DE ACKNOWLEDGE A LA PC /////////////////////////////////////////**

**//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////**

**/\* Se prepara y envia un mensaje de acknowledge, cuyo contenido viene dado por codigoMensaje.**

**\*/**

**void EnviarAcknowledge(unsigned char codigoMensaje)**

**{**

**ack[0] = CODIGO\_INICIO\_MENSAJE;**

**ack[1] = CODIGO\_ACK;**

**ack[2] = codigoMensaje;**

**ack[3] = (ack[0] ^ ack[1] ^ ack[2]);**

**Serial.write(ack, 4);**

**}**

1. **Apéndice F: Especificación de mensajes del software de control implementado en ROS.**
2. **Apéndice G: Código del paquete de ROS comunicación\_serial.**
3. **Apéndice H: Código del paquete de ROS logitech\_gamepad\_ii.**
4. **Apéndice I: Código del paquete de ROS exportador\_estado\_csv.**
5. **Apéndice J: Código del paquete de ROS exportador\_telemetria\_csv.**
6. **Apéndice K: Código del script de MATLAB para análisis de respuesta en frecuencia a partir de la interfaz de telemetría, adaptado al tópico de ROS telemetría\_total.**
7. **Apéndice K: Código del script de MATLAB para análisis de respuesta en frecuencia a partir de la interfaz de telemetría, adaptado al tópico de ROS estado\_cuadricoptero.**
8. **Apéndice L: Código del script de MATLAB para realizar la simulación de la arquitectura de control propuesta.**