INSTITUTO TECNOLÓGICO SUPERIOR AERONÁUTICO

CARRERA DE ELECTRÓNICA MENCIÓN INSTRUMENTACIÓN Y AVIÓNICA

"ELABORACIÓN DE GUÍAS DE PROGRAMACIÓN PARA EL CONTROL DE NAVEGACIÓN APLICADO AL MINI UAV TIPO QUADCOPTER"

POR:

CRISTIAN ANDRÉS ZAMBRANO CASTAÑEDA

Trabajo de Graduación como requisito previo para la obtención del Título de:

TECNÓLOGO EN ELECTRÓNICA MENCIÓN EN INSTRUMENTACIÓN Y AVIÓNICA

2012

CERTIFICACIÓN

Certifico que el presente Trabajo de Graduación fue realizado en su totalidad por el Sr. **ZAMBRANO CASTAÑEDA CRISTIAN ANDRÉS,** como requerimiento parcial para la obtención del título de **TECNÓLOGO EN ELECTRÓNICA MENCIÓN INSTRUMENTACIÓN Y AVIÓNICA.**

_____

SR. ING. JORGE PARDO

DIRECTOR DEL PROYECTO

Latacunga, 05 de Julio del 2012

DEDICATORIA

Se lo dedico muy personalmente a mi madre Alicia Castañeda, sin ella no estaría presente en el mundo, ella me ha enseñado que siempre hay que esforzarse y trabajar duro para conseguir lo que uno quiere, me ha enseñado a que uno mismo debe ganarse el nombre, gracias infinitamente por todo el apoyo que me has sabio brindar.

A mi hermana Nathaly Viracocha, ella ha sido mi ejemplo de tenacidad, mi ejemplo para no rendirme y seguir siempre adelante, ella me ha mostrado que si uno quiere ser algo más en la vida se puede lograr pero es necesario trabajar duro, todos los sacrificios tienen su recompensa, de igual manera gracias por todo el apoyo que me has dado.

Existen muchas personas a las cuales me gustaría dedicar el presente trabajo de graduación, cada una me ha enseñado diferentes cosas que me han ayudado a formarme en la vida, gracias a toda mi familia y a mis otras madres Rosa Castañeda, Guadalupe Castañeda, Margarita Castañeda y a mi abuelito Manuel Castañeda infinitas gracias a todos por su apoyo.

Cristian Zambrano.

AGRADECIMIENTOS

Primero ante todo debo agradecer muy personalmente al padre de todos nosotros, sus bendiciones me han mostrado el camino correcto, llenándome de fe para no dejarme caer, y si fuese el caso brindándome aliento para levantarme volver a empezar y seguir adelante.

A mis profesores los cuales me han mostrado el sendero del conocimiento, son las herramientas que dignifican al hombre y le permiten ser un ente útil para la humanidad, me disculpo sinceramente si en algún momento los he ofendido y me arrepiento si no supe aprovechar la oportunidad de conocimiento que ustedes me han brindado.

Agradezco muy personalmente a mis amigos de carrera, para mí son como mis hermanos, nos formamos juntos entre risas y preocupaciones, cada uno aportó un granito de arena en nuestra formación personal y profesional, nunca piensen que están solos porque siempre estaré allí si me necesitan.

A mi familia, les agradezco infinitamente todos, me han llenado de valores de aliento, de ánimo, nunca me han negado nada y siempre han sido mi apoyo, siempre han creído en mí y en mis capacidades para seguir adelante, de igual manera a mi nueva familia, la familia Moreno Cerón la cual me ha acogido, me ha brindado un nuevo hogar y me ha enseñado que nunca se debe olvidar que siempre debe haber amor en el hogar y finalmente a mi Katy gracias por enseñarme a volver a creer.

Agradezco al Instituto Tecnológico Superior Aeronáutico, siempre honraré su nombre.

Mi más sincero agradecimiento a todos.

Cristian Zambrano.

ÍNDICE DE CONTENIDOS

CAPÍTULO III
DESARROLLO DEL TEMA

CAPÍTULO IV
CONCLUSIONES Y RECOMENDACIONES

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE FOTOS

ÍNDICE DE ANEXOS

RESUMEN

En el presente proyecto se realizó el estudio del código de programación aplicado a un mini UAV tipo Quadcopter para el control de navegación desglosando cada uno de sus elementos como son su estructura, variables y funciones, estos elementos de programación fueron desarrollados siguiendo un nuevo concepto denominado como programación abierta, de esta manera se da cabida a la creación de una nueva tecnología lógica abierta al mundo actual.

En primer lugar se detallan los conceptos básicos de cómo está estructurado un dispositivo UAV tipo Quadcopter, además se detallan los diferentes sistemas incorporados en el mismo, cada una de sus partes realiza una función específica y el conjunto de todas estas funciones nos permiten obtener un dispositivo autónomo.

Posteriormente se mostrará muy específicamente desde un nivel básico hasta un nivel avanzado como se realiza la estructuración de un programa de código abierto utilizando el software compilador de Arduino, sus funciones lógicas, numéricas, de control, y de adquisición de datos, mostrando muy claramente sus ventajas y limitaciones.

El conocer todos estos conceptos permite al usuario comprender el funcionamiento del dispositivo, ya que está conformado por diferentes sensores y dispositivos analógico - digitales, estos elementos poseen protocolos de comunicación diferentes para optimizar su rendimiento y asegurar su compatibilidad con el código de control.

ABSTRACT

This project was the study of programming code applied to mini UAV Quadcopter type navigation control by breaking down each of its elements such as structure, variables, and functions, these programming elements were developed following a new concept called as open source, so it allows for the creation of a new logic technology open to the world.

First details the basic of how a device is structured such UAV Quadcopter also details the different systems incorporated there in, each of it´s parts performs a specific functions and the set of all these features allow us to obtain a device autonomous.

Later will be very specifically from entry level o an advanced level as is done to structure a program of open source software using the Arduino compiler, it´s logic function, numerical control and data acquisition, clearly showing their advantages and limitations.

Knowing all these concepts allows the use to understand the functioning of the device, which consist of various sensors and analog - digital, these elements have differents communication protocols to optimize performance and ensure compatibility with the control code.

CAPÍTULO I

TEMA

1.1. ANTECEDENTES

El presente proyecto de grado ha sido realizado en base a la investigación de nuevas tecnologías aplicadas al desarrollo de aeromodelos más sofisticados denominados UAV´s (Vehículo Aéreo no Tripulado), esto significa que el control del vehículo aéreo se lo realiza de forma remota mediante radio control por un operador.

En la actualidad se ha logrado excluir de forma parcial al operador y al mismo tiempo la dependencia de un radio transmisor de control, esto se ha logrado mediante la implementación de un sistema de navegación programable instalado en el aeromodelo el cual funciona conjuntamente con diversos dispositivos electrónicos dando como resultado una autonomía parcial al vehículo aéreo en lo que se refiere a la navegación ya que aún depende del operador para realizar otro tipo de funciones específicas.

1.2. JUSTIFICACIÓN E IMPORTANCIA

Las nuevas tecnologías han desarrollado nuevos y mejores sistemas de control aplicado al campo del aeromodelismo, todos estos avances tecnológicos son la respuesta de un trabajo conjunto entre diferentes dispositivos electrónicos, de esta manera se puede obtener como resultado un sistema real de navegación controlado por cualquier usuario.

Estos tipos de avances integran sistemas GPS, Giróscopos Electrónicos y un sistema Acelerómetro, los cuales por sus características, sencillez y tamaño han logrado ser implementados en los aeromodelos de la actualidad. El aporte de conocimientos con la presente investigación está enfocado principalmente a los futuros profesionales en el campo de la Electrónica, de esta manera logramos mantener sus conocimientos en constate actualización.

1.3. OBJETIVOS

1.3.1. General

Elaborar una guía de programación para el control de navegación aplicada al mini UAV tipo Quadcopter.

1.3.2. Específicos

- Identificar las características y especificaciones del mini UAV tipo Quadcopter
- Analizar los parámetros básicos de programación abierta que ofrece la tecnología de Arduino
- Analizar los componentes que integran el sistema Quadcopter
- Realizar un estudio de la programación incorporada al mini UAV tipo Quadcopter
- Determinar la factibilidad de modificar la programación incorporada al mini UAV tipo Quadcopter
- Realizar una guía de procedimientos para la configuración y calibración básica del mini UAV tipo Quadcopter

1.4. ALCANCE

Esta investigación está dirigida en beneficio de los estudiantes y docentes de la Carrera de Electrónica, de esta manera se logrará elevar el nivel de conocimientos en el campo de la instrumentación aeronáutica ya que el modelo adquirido denominado UAV tipo Quadcopter posee cualidades únicas de tecnología, permitiendo al estudiante observar en tiempo real el comportamiento de los principales instrumentos de navegación mediante el software denominado APM Planner.

CAPÍTULO II

MARCO TEÓRICO

2.1. VEHÍCULO AÉREO NO TRIPULADO[1]

Un vehículo aéreo no tripulado, UAV por siglas en inglés (Unmanned Aerial Vehicle), es una aeronave que vuela sin tripulación humana a bordo. Son usados mayoritariamente en aplicaciones militares. Un UAV se define como un vehículo sin tripulación reutilizable capaz de mantener un nivel de vuelo controlado y sostenido propulsado principalmente por un motor a reacción o actualmente por motores eléctricos guiados remotamente.

Existe una amplia variedad de formas, tamaños, configuraciones y características en el diseño de los UAV. Históricamente los UAV eran simplemente aviones pilotados remotamente (en inglés: drones), pero cada vez más se está empleando el control autónomo de los UAV. En este sentido se han creado dos variantes, algunos son controlados desde una ubicación remota, y otros vuelan de forma autónoma basándose en planes de vuelo pre-programados usando sistemas más complejos de automatización.

Cabe destacar que las aeronaves piloteadas remotamente en realidad no califican para ser llamadas como UAV, ya que los vehículos aéreos piloteados remotamente (o por control remoto) se conocen como Aeronaves Radio controladas o Aeronaves RC; esto debido a que, precisamente, los UAV son también sistemas autónomos que pueden operar sin intervención humana, pueden despegar, volar y aterrizar automáticamente.

Actualmente los UAV también son utilizados en un pequeño pero creciente número de aplicaciones civiles, como en labores de lucha contra incendios o seguridad civil como la vigilancia de oleoductos.

[1] http://es.wikipedia.org/wiki/Veh%C3%ADculo_a%C3%A9reo_no_tripulado

Figura 2.1: Vehículo Aéreo No Tripulado
Fuente: http://lemonodor.com/archives/001317.html

2.2. QUADCOPTER

Un Quadcopter también denominado cuadricóptero en español, es una aeronave no tripulada que se eleva y se desplaza por el impulso de cuatro motores colocados en los extremos de una estructura rígida en forma de cruz tal como se presenta en la figura 2.2, el vehículo dispone de 4 motores con sus respectivas hélices de ángulo fijo, se utiliza la velocidad de los motores para controlar la estabilidad y movimientos del vehículo aéreo, en el centro posee una tarjeta electrónica la cual mediante una pre-programación realizada por el usuario comanda el cuadricóptero.

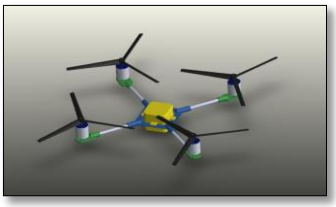

Figura 2.2: UAV Tipo Cuadricóptero
Fuente: www.vtol-technologies.com

2.2.1. Principales Características

Los cuadricópteros se sitúan en la categoría de mini UAV´s, esto significa que posee un peso menor a 25 kg obteniendo las siguientes características generales de vuelo:

Características:

- Alcance: 10Km
- Altitud en vuelo: 300m
- Autonomía: 2 Horas (dependiendo del tipo de batería utilizada)
- Carga máxima: 5Kg

Adicionalmente una de las principales características es la gran maniobrabilidad que pose este tipo de vehículo aéreo ya que el control de los cuatro motores es bastante exacto lo que ayuda a utilizarlo en aplicaciones donde la exactitud de vuelo estacionario es muy importante.

Al igual que un helicóptero estos vehículos disponen de una capacidad de vuelo vertical que los hacen únicos, esta función es ventajosa cuando no queremos tener mucha velocidad horizontal y cuando queremos tener una buena capacidad de vuelo estacionario, lo que ayuda a elegir este tipo de sistemas cuando se desea adquirir datos desde el vehículo.

El problema fundamental de los cuadricópteros es su control, el sistema debe incorporar mecanismos de estabilización para ayudar a mantener un correcto control de navegación. Esta característica hace posible el incorporar un gran número de sensores y una de las características más importantes a tener en cuenta en los sistemas de vuelo es la autonomía.

La autonomía de vuelo no suele ser muy buena por el alto consumo de energía que dispone para alimentar sus sistemas, la autonomía se ve reducida a menos de 2 horas, esta fue una de las limitaciones por la que los UAV tardaron un cierto tiempo en avanzar.

Actualmente se están realizando avances importantes en las fuentes de alimentación como son las baterías de polímero de Litio, proporcionado más capacidad de corriente y reduciendo los tamaños.

2.2.2. Diseño Arquitectónico del Sistema Quadcopter

A continuación se presenta la arquitectura de un sistema cuadricóptero, para ello se ha dividido todo el sistema en varias partes principales.

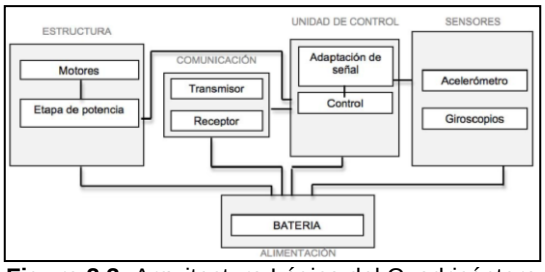

Figura 2.3: Arquitectura Lógica del Cuadricóptero
Fuente: http://es.scribd.com/doc/41621520/14/CAPITULO-2-ARQUITECTURA-DEL-SISTEMA-QUADROTOR

2.2.3. Dinámica de Movimiento

La dinámica o la relación de movimiento de un sistema cuadricóptero inicia a partir del momento de inercia de cada rotor. Cada rotor consta de un motor conjuntamente con una hélice de palas de ángulo fijo, una de las características más importantes del cuadricóptero es, que el motor delantero y el motor trasero rotan en sentido horario (motores 2 y 3), mientras que los otros dos motores rotan en sentido anti-horario (motores 0 y 1). De esta manera obtenemos como resultado que los efectos giroscópicos y los momentos aerodinámicos tienden a cancelarse en vuelo estacionario.

Para lograr que el aeromodelo realice un movimiento hacia adelante la velocidad del rotor trasero (motor 3) debe aumentar y simultáneamente la velocidad del rotor delantero (motor 2) debe disminuir. El desplazamiento lateral se ejecuta siguiendo el mismo procedimiento, pero usando los rotores de la derecha y de la izquierda.

El movimiento de rotación sobre el mismo eje se obtiene con cada par de rotores, es decir, se acelera los dos rotores con sentido horario mientras se desacelera los rotores con sentido anti-horario y siguiendo el mismo principio para girar de lado contrario.

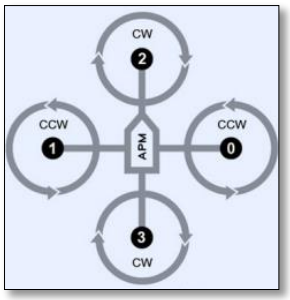

Figura 2.4: Movimiento de Motores
Fuente: Investigación de Campo

2.2.4. Motores

Los motores más utilizados en aeromodelismo son motores de corriente contínua DC y los motores Brushless.

2.2.4.1. Motores DC

Los motores de corriente continua son muy utilizados en aplicaciones de aeromodelismo, de igual manera para controles pequeños como mini robots y accionamientos varios. El rotor es el mecanismo que gira en el centro del motor DC y está compuesto de enrodamientos de cable conductores de corriente. Esta corriente es suministrada al rotor por medio de las "escobillas" generalmente fabricadas de carbón. La fuerza con la que el motor gira es proporcional a la corriente que hay por los conductores. A mayor tensión, mayor corriente y mayor par o fuerza de torque en el motor.

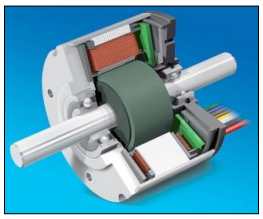

Figura 2.5: Motor DC
Fuente: www.dvcg22a.blogspot.com

2.2.4.2. Motor Brushless

Actualmente se utilizan mucho los motores brushless o trifásicos, esto se debe a la superioridad y eficiencia que ofrece sobre los motores DC en dos aspectos fundamentales: relación potencia-peso (también menor tamaño para la misma potencia). Esto implica que la cantidad de energía eléctrica que se transforma en energía mecánica es mucho mayor, este tipo de motor carece de colector y escobillas, este tipo de motor funciona en AC y la mayoría se alimenta con una señal trifásica, esta señal idealmente debería ser sinusoidal, pero en la práctica son señales de pulsos (PWM).

Ventajas:

- Mayor eficiencia con menos perdida por emisión de potencia

- Mayor potencia con menos consumo de corriente
- Menor peso para la misma potencia
- Requieren menos mantenimiento al no tener escobillas
- Relación velocidad/par (torque) del motor es casi constante
- Mayor potencia para el mismo tamaño
- Rango de velocidad elevado al no tener limitaciones mecánicas

Desventajas:

- Mayor coste de construcción
- El control es caro y complejo
- Es necesario un controlador de velocidad para que funcione

2.2.5. Control Electrónico de Velocidad (ESC & BEC)[2]

El ESC o control electrónico de velocidad tiene como finalidad mantener una velocidad estable en el rotor, sin importar la carga o las perturbaciones que tenga, en general una hélice se considera una carga que posee un conjunto de perturbaciones dinámicas lo cual hace de este un problema de control complejo.

El ESC recibe la señal PWM con una frecuencia de 50 Hz y dependiendo de la longitud del ancho de pulso entregará más o menos potencia al motor, esta señal será emitida por nuestro controlador. La mayoría de los ESC son programables algunos mediante comunicación serie y otros con una tarjeta programadora diseñada por el fabricante.

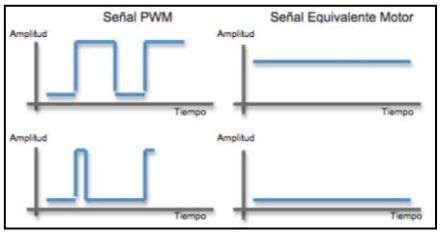

Figura 2.6: Señal PWM/Señal Equivalente que llega al Motor
Fuente: Investigación de Campo

El BEC o circuito eliminador de batería está diseñado para proporcionar energía eléctrica a otros circuitos como servomotores y al circuito receptor eliminando la

[2] www.wikipedia.org/wiki/Battery_eliminator_circuit

necesidad de una batería adicional, en algunos casos el BEC es parte del receptor.

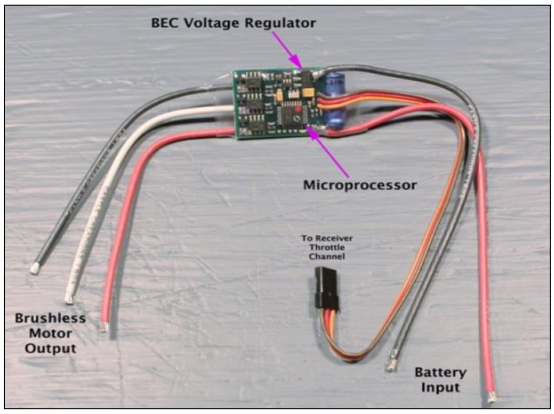

Figura 2.7: ESC & BEC
Fuente: http://www.ikkaro.com/node/714

2.2.6. Fuente de Alimentación

La alimentación es uno de los factores más importantes para el funcionamiento de un aeromodelo autónomo, actualmente existen dos alternativas para solucionar este inconveniente, una de estas alternativas es mediante la utilización de almacenadores, en el mercado podemos encontrar dos clases de almacenadores: el primero denominado “pila” pero con un rápido desgaste energético, y el secundario las denominadas “baterías”, las cuales dependiendo su composición química resisten al desgaste energético por un mayor tiempo prolongado su uso, además tienen la opción de ser recargadas sometiéndose a un estado de alimentación con corriente eléctrica continua. La capacidad de una batería es la cantidad total de carga producida en la reacción electroquímica y se define en unidades o amperios hora (Ah).

Los parámetros a considerar de una batería son los siguientes:

- La tensión de salida medida en voltios.
- La capacidad eléctrica, se mide por referencia a los tiempos de carga o de descarga en amperios hora (Ah). Este parámetro es muy importante y debemos fijarnos en él cuando compremos la batería pues cuanto mayor sea, más tiempo tardará en descargarse por el uso. El precio está en relación directa con este parámetro.

2.2.6.1. Batería de Polímero de Litio (Li-Po)

Son una variación de las baterías de Ion-Litio, este nuevo tipo de batería es el más utilizado para aeromodelos y helicópteros, son más ligeras y tienen una densidad de energía entre 5 y 12 veces mayor a las baterías de Ni-Cd o las de Ni-MH. La desventaja es que necesitan un tiempo de recarga mucho más lenta pero son ideales para alimentar motores muy potentes. Se debe cuidar de que el consumo máximo del motor sea menor que la descarga máxima de la batería para evitar que la vida de ésta se acorte demasiado, además posee una resistencia interna pequeña, lo que hace que se pueda aprovechar casi el 100% de la energía disponible.

2.2.7. Unidad de Control[3]

Las salidas de la unidad de control se encargan de controlar la actividad y funcionamiento de nuestros dispositivos. La unidad de control es la circuitería que controla el flujo de datos a través del procesador y coordina las actividades de las otras unidades dentro de él, de cierta manera es el "cerebro dentro del cerebro", pues controla lo que sucede dentro del procesador. La edad de la información moderna no sería posible sin diseños complejos de la unidad de control.

2.2.7.1. Unidad de Control Microprogramada[4]

La idea de microprogramación fue introducida por M.V. Wilkes en 1951 como un nivel intermediario para ejecutar instrucciones de programa de computador. Los microprogramas fueron organizados como una secuencia de microinstrucciones y almacenados en una memoria de control especial, la ventaja principal de la unidad de control microprogramada es la simplicidad de su estructura, las salidas del controlador son organizadas en microinstrucciones y pueden ser reemplazadas fácilmente.

[3] http://es.wikipedia.org/wiki/Unidad_de_control
[4] http://es.wikipedia.org/wiki/Unidad_de_control/Microprogramación

2.2.7.2. Microcontrolador[5]

Un microcontrolador (µC,UC o MCU) es un circuito integrado programable, capaz de ejecutar las órdenes grabadas en su memoria. Está compuesto de varios bloques funcionales los cuales cumplen una tarea específica, un microcontrolador incluye en su interior las tres unidades funcionales principales de una computadora.

- Unidad central de procesamiento
- Memoria
- Periféricos de entrada y salida

Al ser fabricados, la memoria EEPROM del microcontrolador no posee datos, para que pueda controlar algún proceso es necesario generar o crear un grupo de instrucciones (programa o código) y luego ser grabado en la memoria EEPROM del microcontrolador, el cual puede ser escrito en lenguaje ensamblador u otro lenguaje para microcontroladores; sin embargo, para que el programa pueda ser grabado en la EEPROM del microcontrolador, debe ser codificado en sistema numérico hexadecimal que es finalmente el sistema que hace trabajar al microcontrolador cuando éste es alimentado con el voltaje adecuado para su funcionamiento.

Son diseñados para reducir costos y el consumo de energía de un sistema en particular. Por eso el tamaño de la unidad central de procesamiento, la cantidad de memoria y los periféricos incluidos dependerán de la aplicación.

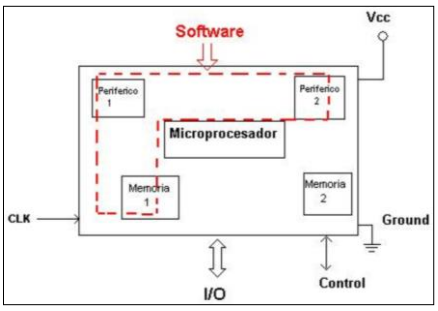

Figura 2.8: Esquema de un Microcontrolador
Fuente: http://es.wikipedia.org/wiki/Microcontrolador

[5] http://es.wikipedia.org/wiki/Microcontrolador

2.2.7.3. Microcontrolador Atmega2560

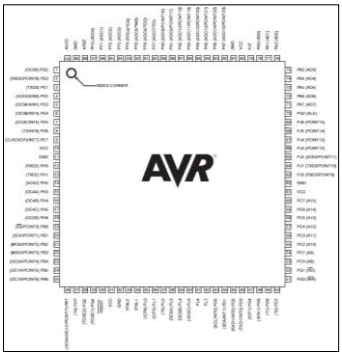

Figura 2.9: Atmega 2560
Fuente: Datasheet Atmega 2560

El Atmega2560 posee las siguientes características:

- 256Kbytes de memoria Flash programable con capacidad de lectura y escritura
- 4Kbytes de memoria EEPROM
- 8Kbytes de RAM
- 54 Puertos E/S
- 14 Puertos destinados a señal PWM con resolución de 16bits
- Contador de tiempo real (RTC)
- 2 Puertos USARTs
- 4 Interfaces de comunicación serie
- 8 Puertos ADC
- Puerto de comunicación serie SPI
- Puerto de comunicación serie I2C
- Atmel ofrece la biblioteca de QTOUCH para utilizar botones táctiles capacitivos, y deslizadores.

Descripción de Puertos

Port A (PA0 - PA7)

Posee puertos E/S bidireccionales de 8 bits con resistencias internas pull-up.

Tabla 2.1: Puerto A: Función alternativa de pines

Puerto	Función Alternativa
PA7	**AD7** Dirección de interface con la memoria externa bit 7
PA6	**AD6** Dirección de interface con la memoria externa bit 6
PA5	**AD5** Dirección de interface con la memoria externa bit 5
PA4	**AD4** Dirección de interface con la memoria externa bit 4
PA3	**AD3** Dirección de interface con la memoria externa bit 3
PA2	**AD2** Dirección de interface con la memoria externa bit 2
PA1	**AD1** Dirección de interface con la memoria externa bit 1
PA0	**AD0** Dirección de interface con la memoria externa bit 0

Fuente: Investigación de Campo

Port B (PB0 - PB7)

Posee puertos E/S bidireccionales de 8 bits con resistencias internas pull-up.

Tabla 2.2: Puerto B: Función alternativa de pines

Puerto	Función Alternativa
PB7	**OC0A/OC1C/PCINT7** Salida PWM A para Timer/Counter0 Comparador de salida y salida PWM A para Timer/Counter1 Pin interruptor de cambio 7)
PB6	**OC1B/PCINT6** Comparador de salida Salida PWM B para Timer/Counter1 Pin interruptor de cambio 6)
PB5	**OC1A/PCINT5** Comparador de salida Salida PWM A para Timer/Counter1 Pin interruptor de cambio 5)
PB4	**OC1A/PCINT5** Salida PWM A para Timer/Counter2 Pin interruptor de cambio 4)
PB3	**MISO/PCINT3** SPI Bus Master de entrada/salida del esclavo Pin interruptor de cambio 3
PB2	**MISO/PCINT2** SPI Bus Master de entrada/salida del esclavo Pin interruptor de cambio 2
PB1	**SCK/PCINT1** SPI Bus Serial Clock Pin interruptor de cambio 1
PB0	**SS/PCINT0** SPI Selector de entrada esclavo Pin interruptor de cambio 0

Fuente: Investigación de Campo

Port C (PC0 - PC7)

Posee puertos E/S bidireccionales de 8 bits con resistencias internas pull-up.

Tabla 2.3: Puerto C: Función alternativa de pines.

Puerto	Función Alternativa
PC7	**A15** Dirección de interface con la memoria externa bit 15
PC6	**A14** Dirección de interface con la memoria externa bit 14
PC5	**A13** Dirección de interface con la memoria externa bit 13
PC4	**A12** Dirección de interface con la memoria externa bit 12
PC3	**A11** Dirección de interface con la memoria externa bit 11
PC2	**A10** Dirección de interface con la memoria externa bit 10
PC1	**A9** Dirección de interface con la memoria externa bit 9
PC0	**A8** Dirección de interface con la memoria externa bit 8

Fuente: Investigación de Campo

Port D (PD0 - PD7)

Posee puertos E/S bidireccionales de 8 bits con resistencias internas pull-up.

Tabla 2.4: Puerto D: Función alternativa de pines.

Puerto	Función Alternativa
PD7	**T0** Timer/Counter0 Entrada de reloj 0
PD6	**T1** Timer/Counter0 Entrada de reloj
PD5	**XCK1** USART1 Reloj externo I/O
PD4	**ICP1** Timer/Counter1 Capturador de entrada del disparador (Trigger)
PD3	**INT3/TXD1** Entrada del interruptor externo 3 Puerto de transmisión USART1
PD2	**INT2/RXD1** Entrada del interruptor externo 2 Puerto de recepción USART1
PD1	**INT1/SDA** Entrada del interruptor externo 1 TWI Serial Data
PD0	**INT0/SCL** Entrada del interruptor externo 0 TWI Serial CLock

Fuente: Investigación de Campo

Port E (PE0 - PE7)

Posee puertos E/S bidireccionales de 8 bits con resistencias internas pull-up.

Tabla 2.5: Puerto E: Función alternativa de pines.

Puerto	Función Alternativa
PE7	**INT7/ICP3/CLK0** Entrada del interruptor externo 7 Timer/Counter3 Capturador de entrada del disparador (Trigger) Sistema dividido de reloj
PE6	**INT6/T3** Entrada del interruptor externo 6 Timer/Counter3 Entrada de reloj
PE5	**INT5/OC3C** Entrada del interruptor externo 5 Comparador de salida Salida de señal PWM C para Timer/Counter3
PE4	**INT4/OC3B** Entrada del interruptor externo 4 Comparador de salida Salida de señal PWM B para Timer/Counter3
PE3	**AIN1/OC3A** Entrada negativa del comparador análogo Comparador de salida Salida de señal PWM A para Timer/Counter3
PE2	**AIN0/XCK0** Entrada positiva del comparador análogo USART0 Reloj externo I/O
PE1	**TXD0** Puerto de transmisión
PE0	**RXD0/PCINT8** USART0 Puerto de recepción Pin interruptor de cambio 8

Fuente: Investigación de Campo

Port F (PF0 - PF7)

El Puerto F funciona como entrada analógica para el convertidor analógico-digital, también funciona como puertos de E/S bidireccional de 8 bits con resistencias internas pull-up si el convertidor analógico-digital no es usado.

Tabla 2.6: Puerto F: Función alternativa de pines.

Puerto	Función Alternativa
PF7	**ADC7/TDI** Canal de entrada ADC7 JTAG Test Data Input
PF6	**ADC6/TDO** Canal de entrada ADC6 JTAG Test Data Output

PF5	**ADC5/TMS** Canal de entrada ADC5 JTAG Test Mode Select
PF4	**ADC4/TCK** Canal de entrada ADC4 JTAG Test Clock
PF3	**ADC3** Canal de entrada ADC3
PF2	**ADC2** Canal de entrada ADC2
PF1	**ADC1** Canal de entrada ADC1
PF0	**ADC0** Canal de entrada ADC0

Fuente: Investigación de Campo

Port G (PG0 – PG5)

Posee puertos E/S bidireccionales de 6 bits con resistencias internas pull-up.

Tabla 2.7: Puerto G: Función alternativa de pines.

Puerto	Función Alternativa
PG5	**OC0B** Comparador de salida Salida de señal PWM B para el Timer/Counter0
PG4	**TOSC1** RTC Oscilador Timer/Counter2
PG3	**TOSC2** RTC Oscilador Timer/Counter2
PG2	**ALE** Habilita la memoria externa
PG1	**RD** Lectura rápida de memoria externa
PG0	**WR** Escritura rápida de memoria externa
PG0	**ADC0** Canal de entrada ADC0

Fuente: Investigación de Campo

Port H (PH0 – PH7)

Posee puertos E/S bidireccionales de 8 bits con resistencias internas pull-up.

Tabla 2.8: Puerto H: Función alternativa de pines.

Puerto	Función Alternativa
PH7	**T4** Timer/Counter 4 Entrada de reloj
PH6	**OC2B** Comparador de salida Salida de señal PWM B para el Timer/Counter 2
PH5	**OC4C**

	Comparador de salida Salida de señal PWM C para el Timer/Counter 4
PH4	**OC4B** Comparador de salida Salida de señal PWM B para el Timer/Counter 4
PH3	**OC4A** Comparador de salida Salida de señal PWM A para el Timer/Counter 4
PH2	**XCK2** USART2 Reloj externo
PH1	**TXD2** USART2 Puerto de transmisión
PH0	**RXD2** USART2 Puerto de recepción

Fuente: Investigación de Campo

Port J (PJ0 – PJ7)

Posee puertos E/S bidireccionales de 6 bits con resistencias internas pull-up.

Tabla 2.9: Puerto J: Función alternativa de pines.

Puerto	Función Alternativa
PJ7	**-**
PJ6	**PCINT15** Puerto interruptor de cambio 15
PJ5	**PCINT14** Puerto interruptor de cambio 14
PJ4	**PCINT13** Puerto interruptor de cambio 13
PJ3	**PCINT12** Puerto interruptor de cambio 12
PJ2	**XCK3/PCINT11** USART3 Reloj externo Puerto interruptor de cambio 11
PJ1	**TXD3/PCINT10** USART3 Puerto de transmisión Puerto interruptor de cambio 10
PJ0	**RXD3/PCINT9** USART3 Puerto de recepción Puerto interruptor de cambio 10

Fuente: Investigación de Campo

Port K (PK0 – PK7)

El Puerto K funciona como entrada analógica para el convertidor analógico-digital, también funciona como puertos de E/S bidireccional de 8 bits con resistencias internas pull-up si el convertidor analógico-digital no es usado.

Tabla 2.10: Puerto J: Función alternativa de pines.

Puerto	Función Alternativa
PK7	**ADC15/PCINT23** Canal de entrada ADC 15 Puerto interruptor de cambio 23
PK6	**ADC14/PCINT22** Canal de entrada ADC 14 Puerto interruptor de cambio 22
PK5	**ADC13/PCINT21** Canal de entrada ADC 13 Puerto interruptor de cambio 21
PK4	**ADC12/PCINT20** Canal de entrada ADC 12 Puerto interruptor de cambio 20
PK3	**ADC11/PCINT19** Canal de entrada ADC 11 Puerto interruptor de cambio 19
PK2	**ADC10/PCINT18** Canal de entrada ADC 10 Puerto interruptor de cambio 18
PK1	**ADC9/PCINT17** Canal de entrada ADC 9 Puerto interruptor de cambio 17
PK0	**ADC8/PCINT16** Canal de entrada ADC 8 Puerto interruptor de cambio 16

Fuente: Investigación de Campo

Port L (PL0 – PL7)

Posee puertos E/S bidireccionales de 6 bits con resistencias internas pull-up.

Tabla 2.11: Puerto L: Función alternativa de pines.

Puerto	Función Alternativa
PL7	**-**
PL6	**-**
PL5	**OC5C** Comparador de salida Salida de señal PWM C para el Timer/Counter5
PL4	**OC5B** Comparador de salida Salida de señal PWM B para el Timer/Counter5
PL3	**OC5A** Comparador de salida Salida de señal PWM A para el Timer/Counter5
PL2	**T5** Timer/Counter5 Entrada de reloj
PL1	**ICP5** Timer/Counter5 Capturador de entrada del disparador (Trigger)
PL0	**ICP4** Timer/Counter4 Capturador de entrada del disparador (Trigger)

Fuente: Investigación de Campo

2.2.8. Sistema de Comunicación[6]

El control más utilizado en este tipo de sistemas es la emisora de radio control, el radio control “RC” es la técnica que permite el gobierno de un objeto a distancia y de manera inalámbrica por medio de ondas de radio.

El radiocontrol posee tres elementos tecnológicos fundamentales:

- La electrónica que se encarga de transformar los comandos en ondas de radio para transmisor y a la inversa en el receptor.
- La electricidad, encargada de proporcionar la energía necesaria a los dispositivos tanto el comando o transmisor como el receptor.
- La mecánica encargada de mover los actuadores, servomotores o motores según la señal demoduladas o decodificadas que proviene del receptor.

Las principales limitaciones y causa de problemas es el uso de la misma frecuencia por más de un usuario, la segunda causa de problemas es la falta de estabilidad de emisoras y receptores, a los que pueden influir canales adyacentes.

Utilizando buenos sistemas receptores este problema se logra minimizar en un alto grado, pero si una emisora adyacente emite con mucha potencia y con bastante desviación del canal entonces terminará influyendo en el vehículo y al alejarse de nuestra emisora se puede llegar a tener serios problemas de control.

Esta es una tecnología muy simple y antigua comercializada hace varias décadas buscando la simplicidad para poder limitar el precio, tamaño y peso de los receptores, en ningún momento se esperaba que un microprocesador fuese a estar involucrado así que no tenía sentido hacerla más compleja.

2.2.8.1. Señal PPM

La señal PPM (modulación por posición de pulso), es el método de codificación más usado en radiocontrol. El estándar utilizado en RC es que el servomotor

[6] http://es.wikipedia.org/wiki/Aeromodelismo#Radiocontrol

tenga la posición máxima en un extremo cuando la posición del pulso que recibe es de una duración de 1ms, el punto medio está ubicado en 1.5ms y el máximo opuesto esta cuando el pulso tiene una duración de 2ms, este pulso debe repetirse en un tiempo de entre 20 a 30 veces. Utilizando esta información se realizan los circuitos que sirven para comprobar el funcionamiento de servomotores, todos los circuitos de radiocontrol utilizan este estándar.

La señal PPM es una señal de baja frecuencia que codifica la posición de las palancas de mando y demás controles, esta señal es transformada en radiofrecuencia para que se envié a distancia al receptor que se encarga luego en su etapa decodificadora en volverla entendible para los dispositivos de movimiento. Es importante conocer en términos de compatibilidad que algunas radios trabajan con señal PPM en modo normal y otras trabajan en modo invertido. Esto hace que los decodificadores interpreten de manera diferente la información.

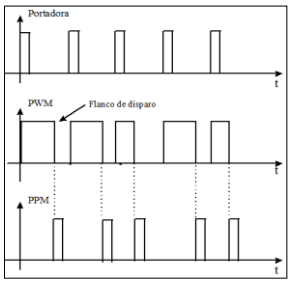

Figura 2.10: Señal PPM & PWM
Fuente: Investigación de Campo

2.3. ARDUINO[7]

Arduino es una plataforma electrónica abierta para la creación de prototipos basada en software y hardware libres fáciles de usar. Se creó para artistas, diseñadores, aficionados y cualquier interesado en crear entornos u objetos electrónicos interactivos.

Arduino puede tomar información del entorno a través de sus pines de entrada utilizando toda una gama de sensores y puede controlar todo aquello que lo rodea controlando luces, motores y otros dispositivos periféricos. El microcontrolador en

[7] http://www.arduino.cc/es/

la placa Arduino se programa mediante el lenguaje de programación Arduino basado en Wiring[8] (código abierto) y el entorno de desarrollo Arduino basado en Processing[9] (programación abierta). Los proyectos hechos con Arduino pueden ejecutarse sin la necesidad de conectarse a un ordenador, si bien tienen la posibilidad de hacerlo y comunicarse con diferentes tipos de software.

Las placas pueden ser hechas a mano o compradas montadas de fábrica; el software puede ser descargado de forma gratuita. Los ficheros de diseño de referencia (CAD) están disponibles bajo una licencia abierta, así el usuario es libre de adaptarlo a sus necesidades.

2.4. ARDUCOPTER[10]

El ArduCopter es un vehículo aéreo no tripulado siguiendo el diseño de un cuadricóptero, fácil de configurar y fácil de volar mediante la transmisión de radio control RC o un control de vuelo autónomo programable que incluye puntos de referencia y un planificador de misiones. El proyecto se basa en el desarrollo del software de navegación automática denominado Ardupilot creado por la comunidad DIYdrones en base a la utilización del hardware y software libre de Arduino, de igual manera se incorporan otros elementos adicionales como son los sensores de tecnología IMU, un magnetómetro y un GPS.

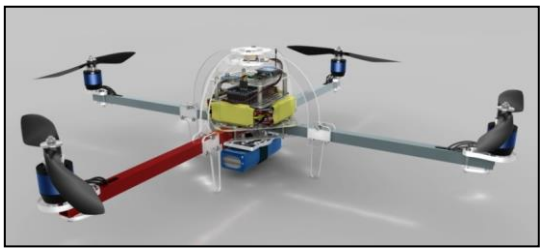

Foto 1: ArduCopter
Fuente: investigación de Campo

2.4.1. Estructura Física del ArduCopter

La estructura se compone de dos ejes metálicos de aluminio perpendiculares formando una cruz simétrica. En sus extremos se incorporan unos pequeños

[8] http://wiring.org.co
[9] http://www.processing.org
[10] http://code.google.com/p/arducopter/wiki/ArduCopter

soportes de plástico rígido para los motores, toda la electrónica está colocada en el centro de la estructura.

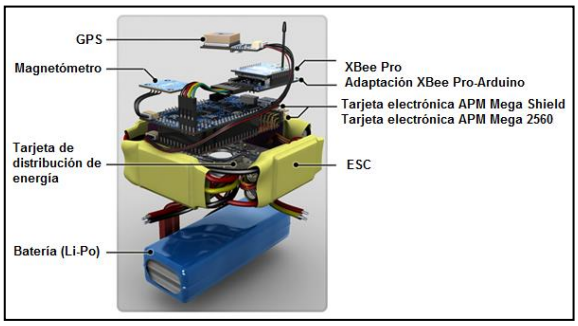

Figura 2.11: Dispositivos Electrónicos que conforman el Arducopter
Fuente: http://code.google.com/p/ardupilot-mega/wiki/IMUHardware

2.4.2. Motores

Los motores empleados para el desarrollo del ArduCopter son motores Brushless de la casa fabricante jDrones con las siguientes características:
Motor Brushless modelo jDrones AC2836-358, 880Kv:

- Tamaño: 28 x 36 mm
- RPM / V: 880 Kv
- Eje: 4mm
- Peso: 72gr
- Carga: 1000-2500gr

Figura 2.12: Motor Brushless
Fuente: http://www.canadadrones.com/product-p/jd-ac2836-358-880kv.htm

2.4.3. Control Electrónico de Velocidad (ESC & BEC)

ESC características del modelo ESC jDrondes AC20-1.0:

- Resistencia de salida extremadamente bajo
- Múltiples funciones de protección de baja tensión, de corte y protección de sobrecalentamiento
- Regulador de voltaje

- Alimenta la salida del motor (máx) 210.000 RPM (2 polos), 70.000 RPM (6 polos), 35.000 RPM (12 polos)
- Consumo de corriente contínua de 20mA
- Salida de BEC: 5V/2A
- Peso: 21gr

Figura 2.13: Motor Brushless
Fuente: http://www.canadadrones.com/product-p/jd-ac2836-358-880kv.htm

2.4.4. Tarjeta Electrónica APM1 Mega Shield - IMU

Esta tarjeta electrónica está conformada por varios sensores los cuales se encargarán de realizar el trabajo de adquisición y acondicionamiento de las señales físicas analógicas (presión atmosférica, temperatura, movimiento) adquiridas del entorno natural, adicionalmente posee el puerto de comunicación entre el usuario y la tarjeta electrónica (puerto mini USB). Estas señales analógicas posteriormente son transformadas a un lenguaje digital entendible por la tarjeta electrónica APM Mega 2560 la cual posee la información necesaria para el control del sistema de navegación interpretando los datos obtenidos, la tarjeta electrónica primaria es denominada Ardupilot Mega Shield u Oilpan APM, es denominada de esta forma por estar localizada sobre la tarjeta electrónica APM Mega 2560 ofreciendo adicionalmente una protección a daños externos.

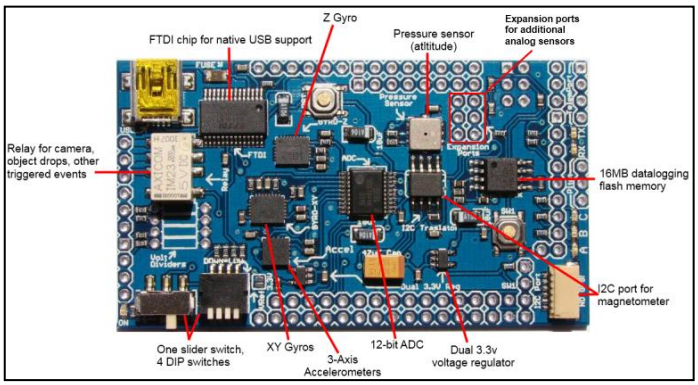

Figura 2.14: Componentes Ardupilot Mega Shield
Fuente: http://code.google.com/p/ardupilot-mega/wiki/IMUHardware

Características:

- Regulador de 3.3V doble (dedicado a los sensores analógicos)
- 1 Interruptor de relé
- Resolución de 12-bits ADC enfocados al sistema giróscopo y acelerómetro
- Memoria interna de 16MB
- Adaptador FTDI para la implementación de comunicación USB
- Puerto de comunicación serial I2C para brújula electrónica
- Botón de reinicio
- Opcional: divisores de tensión fáciles de soldar
- Sistema de Giróscopos (ejes X, Y, Z)
- Dispositivo analógico ADX330 (acelerómetro)
- Dispositivos analógicos IDG500 ISZ500 (giróscopos)
- Sensor Bosch de presión absoluta y temperatura para marcación de altitud
- Entradas analógicas adicionales para la implementación de otros dispositivos

2.4.5. Sensores

Los sensores utilizados por el sistema ArduCopter poseen el principio de unidad de medición inercial (IMU), estos dispositivos electrónicos miden e informan datos de velocidad, orientación y fuerzas gravitacionales que afectan al dispositivo, usando una combinación entre el acelerómetro y el giróscopo. Las unidades de medición inercial son normalmente usadas para maniobrar aviones, incluyendo vehículos aéreos no tripulados, también incluye un sensor barométrico, y una brújula digital para de esta manera obtener la estabilización y control deseado.

2.4.5.1. Acelerómetro ADXL330

El acelerómetro ADXL330 es un sensor de aceleración capacitivo de tres ejes de detección (X, Y, Z), utiliza la variación de los valores de dos capacidades entre las cuales existe un material dieléctrico, para medir las aceleraciones los valores de ambas capacidades son equivalentes.

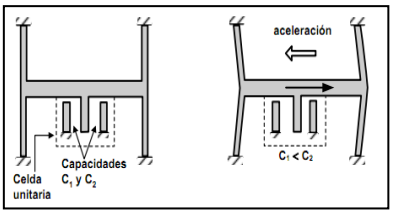

Figura 2.15: Principio de un Sensor Capacitivo
Fuente: http://coolab.umh.es/sea/instrumentacion/2_sensores_new.pdf

Esto significará que la placa central o el material dieléctrico se encuentra justo a la misma distancia entre ambos capacitores y al variar uno de estos valores capacitivos indicará al sensor que se ha producido un movimiento.

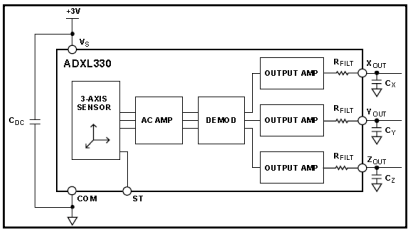

Figura 2.16: Diagrama de Funcionamiento ADXL330
Fuente: Datasheet ADXL330

Este sensor es capaz de medir aceleraciones de ± 3g y cambios de aceleración de hasta 10.000g. Esto se debe a que en la mayoría de aplicaciones estos dispositivos son utilizados como medidores de desaceleración, además este dispositivo puede medir la aceleración de la gravedad estática en las aplicaciones de detección de inclinación, así como la aceleración dinámica resultante de movimiento, choque o vibraciones.

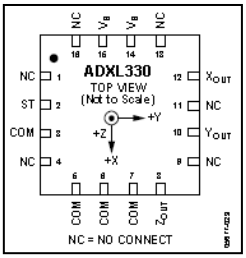

Figura 2.17: Acelerómetro ADXL330
Fuente: http://es.aliexpress.com/product-fm/356365208-ADX

Descripción de Puertos

Tabla 2.12: Descripción de Puertos ADXL 330

Puerto	Nomenclatura	Descripción
1	NC	No Conectado
2	ST	Test
3	COM	Common GND
4	NC	No Conectado
5	COM	Common GND

6	COM	Common GND
7	COM	Common GND
8	Zout	
9	NC	No Conectado
10	Yout	
11	NC	No Conectado
12	Xout	
13	NC	No Conectado
14	Vs	Voltaje (2.0-3.6V)
15	Vs	Voltaje (2.0-3.6V)
16	NC	No Conectado

Fuente: Investigación de Campo

2.4.5.2. Magnetómetro HMC5843[11]

Es comúnmente denominada como brújula digital, está diseñado para la detección del campo magnético terrestre y de esta manera indicarnos cuál es nuestra localización con respecto al norte geográfico.

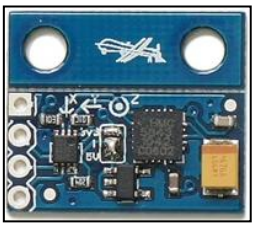

Figura 2.18: Magnetómetro HMC5843
Fuente: http://store.diydrones.com

El sensor tiene un rango de escala completa de ± 4g y una resolución de hasta 7mg. La tensión de alimentación debe estar entre 2,5 y 3.3VDC. La comunicación con el HMC5843 se la realiza mediante el protocolo de comunicación I2C.

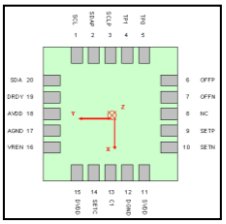

Figura 2.19: Magnetómetro HMC5843
Fuente: Datasheet HMC5843

Tabla 2.13: Descripción de Puertos HMC5843

Puerto	Nomenclatura	Descripción
1	SCL	Serial Clock-I2C Master/Slave Clock
2	SSDAP	Serial Data Pull-up 50Kohm to VDD
3	SCLP	Serial Clcok Pull-up 50Kohm to VDD
4	NC	No Conectado
5	NC	No Conectado

[11] http://www.sparkfun.com/products/9441

6	OFFP	Offset Strap Positive
7	OFFN	Offset Strap Negative
8	NC	No Conectado
9	SETP	Set/Reset Strap Positive – S/R Capacitor (C2) Connection
10	SETN	Set/Reset Strap Negative – Test Point
11	SVDD	Sensor Supply – Test Point
12	DGND	Digital Supply Ground/Return
13	C1	Reservoir Capacitor (C1) Connection
14	SETC	S/R Capacitor (C2) Connection – Driver Side
15	DVDD	Digital Positive Supply
16	VREN	Voltage Regulator Enable, (GND = Dual Supply, AVDD = Single Supply)
17	AGND	Analog Supply Ground/Return
18	AVDD	Analog Positive Supply
19	DRDY	Data Ready – Test Point
20	SDA	Serial Data – I2C Master/Slave Data

Fuente: Investigación de Campo

2.4.5.3. Sensor Digital de Presión Absoluta BMP085

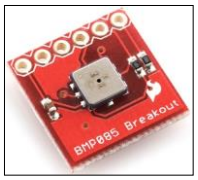

Figura 2.20: Sensor de Presión Absoluta BMP085
Fuente: http://www.sparkfun.com/products/9694

Este sensor barométrico posee cualidades de alta precisión y un bajo consumo de energía. El BMP085 ofrece un rango de medición de 300 a 1100hPa, con una precisión de hasta 0,03hPa. Se basa en la tecnología piezo-resistentes lo que significa que es un sensor cuya resistencia varia en relación a la fuerza que se aplica, un sensor piezo-resistente recibe un voltaje constante de referencia y devuelve una variable con relación a su resistencia variada con una alta precisión y linealidad, así como la estabilidad a largo plazo. Este sensor es compatible con una alimentación de tensión entre 1,8 y 3.6VDC, está diseñado para ser conectado directamente a un micro-controlador a través del puerto de comunicación I2C.

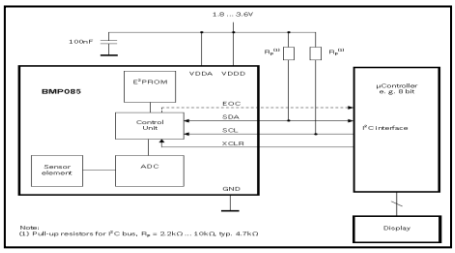

Figura 2.21: Diagrama de Funcionamiento BMP085
Fuente: Datasheet BMP085

2.4.5.4. GPS Mediatek MT3329[12]

Este receptor GPS ofrece alta confiabilidad de posicionamiento y precisión, el módulo GPS puede soportar hasta 66 canales ya que posee un alto nivel de sensibilidad para receptar las señales emitidas del sistema de satélites.

- Dimensiones: 16mm x 16mm x 6mm
- Parche tamaño de la antena: 15 mm x 15 mm x 4 mm
- Alta sensibilidad hasta: -165 dBm

Figura 2.22: Mediatek MT3329 GPS
Fuente: https://store.diydrones.com

2.4.5.5. Giróscopos IDG500 - ISZ500[13]

El sensor giróscopo IDG500 provee la información necesaria de los ejes X e Y, mientras que el sensor ISZ500 provee la información del eje Z, los sensores giróscopos detectan la velocidad de rotación sobre el eje X, Y y Z, dando como resultado un giroscopio integrado de tres ejes.

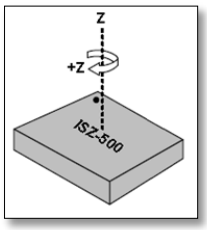

Figura 2.23: Sensor Giróscopo ISZ500
Fuente: Datasheet ISZ-500

El IDG-500 incluye los componentes electrónicos necesarios para obtener una funcionalidad de aplicaciones, incorpora también filtros de paso bajo para el eje X e Y, posee también una memoria EEPROM la cual guarda la información

[12] http://store.diydrones.com/ProductDetails.asp?ProductCode=MT3329-02
[13] http://www.riabelectronics.cl/dual-axis-gyro-idg500.html

necesaria para calibrar el sensor. Los factores de escala por defecto eliminan la necesitad de usar componentes activos externos y que el usuario final pueda realizar la calibración.

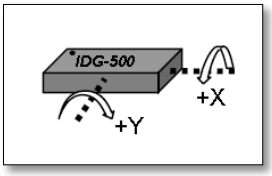

Figura 2.24: Sensor Giróscopo IDG500
Fuente: Datasheet IDG-500

2.4.6. Tarjeta Electrónica APM1 Mega 2560[14]

La tarjeta lógica principal o ArduPilot Mega 2560 es la nueva versión de piloto automático, esta última revisión utiliza el microcontrolador Atmega2560 Rev3 en lugar del microcontrolador Atmega1280.

ArduPilot Mega es un piloto automático totalmente programable que requiere un módulo GPS y sensores para crear un vehículo de funcionamiento aéreo no tripulado (UAV). El piloto automático se encarga tanto de la estabilización y la navegación, eliminando la necesidad de un sistema de estabilización por separado. También es compatible con un sistema "fly-by-wire" de modo que se pueda estabilizar a la aeronave durante el vuelo de forma manual bajo el control de RC, por lo que es más fácil y más seguro para volar. El hardware y el software son de código abierto. El Firmware RC de procesamiento ya está cargado, pero el software de piloto automático debe ser descargado y cargado en la tarjeta por el usuario mediante el compilador de Arduino.

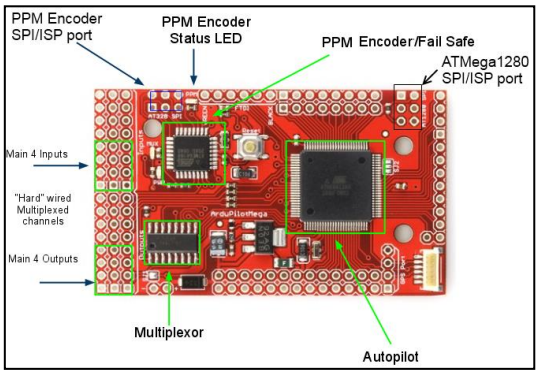

Figura 2.25: Componentes Ardupilot Mega 2560
Fuente: http://www.arduino.cc/es/Hardware

[14] http://www.sparkfun.com/products/10294

Características:

- Basado en un procesador de 16MHz Atmega2560
- Posee un integrado a prueba de fallos que utiliza un circuito separador (chip multiplexor y el procesador Atmega328P) para transferir el control de RC al piloto automático y viceversa. Incluye la capacidad de reiniciar el procesador principal en vuelo
- Un puerto de comunicación serial dedicado a la telemetría de dos vías utilizando los módulos adicionales XBee
- Voltaje de Funcionamiento 5V
- Voltaje de Entrada (Recomendado) 7-12VDC
- Voltaje de Entrada (Límite) 6-20VDC
- Pines E/S Digitales 54 (14 para señal PWM)
- Pines de Entrada Analógica 16
- Intensidad por Pin 40mA
- Intensidad en Pin de 3.3V DC 50mA
- Memoria Flash 256Kb (4KB Bootloader)
- SRAM 8KB
- EEPROM 4KB
- Oscilador 16MHz

2.4.7. Tarjeta Decodificadora PPM

La tarjeta decodificadora de señal PPM V2.0 está basada en la utilización del microcontrolador Atmega 328P, y es el encargado de tomar las señales PPM que emite el receptor y decodificarlas para convertirlas en una señal entendible para los servomotores o motores Brushless (señal PWM), esto permite controlar hasta un máximo de 8 dispositivos a la vez.

2.4.8. Protocolos de Comunicación

2.4.8.1. Comunicación SPI[15]

El Bus SPI (Serial Peripheral Interface) es un estándar de comunicaciones usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos. El bus de interfaz de periféricos serie o bus SPI es un estándar para controlar casi cualquier dispositivo electrónico digital que acepte un flujo de bits serie regulado por un pulso de reloj.

Incluye un puerto de línea de reloj (SCLK), un puerto de datos entrantes (MISO), un puerto para los datos de salida (MOSI) y un puerto selector de dispositivo (SS), que conecta o desconecta la operación del dispositivo con el que uno desea comunicarse. De esta forma, este estándar permite multiplexar las líneas de reloj. Se definen con parámetros CPOL (clock polarity) y CPHA (clock phase) con los cuales determina, con respecto al pulso de reloj el momento en el cual se considera válido un dato de entrada o se genera un dato de salida.

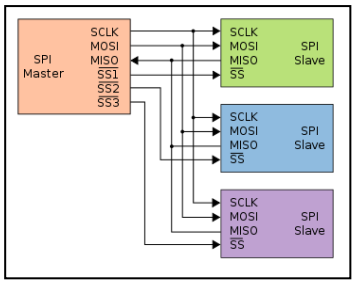

Figura 2.26: Estructura Lógica de Comunicación SPI
Fuente: Investigación de Campo

2.4.8.2. Comunicación I2C

Bajo estas siglas se encuentra uno de los estándares de comunicaciones serie más conocidos en la industria electrónica. I2C es el acrónimo en inglés de Inter-Integrated Circuit, es decir, circuito Inter-Integrado. El objetivo de este bus no será otro que el de interconectar distintos elementos para que pasen a formar parte de un todo.

[15] http://es.wikipedia.org/wiki/Serial_Peripheral_Interface

El bus I2C es un estándar que facilita la comunicación entre microcontroladores, memorias y otros dispositivos con cierto nivel de "inteligencia", sólo requiere de dos líneas de señal y una línea de alimentación común. Fue diseñado por Philips y permite el intercambio de información entre muchos dispositivos a una velocidad aceptable de unos 100 Kbits por segundo, aunque hay casos especiales en los que el reloj llega hasta los 3,4 MHz. La metodología de comunicación de datos del bus I2C es en serie y sincrónica. Una de las señales del bus marca el tiempo (pulsos de reloj) y la otra se utiliza para intercambiar datos. Dado que es un bus serie, el número de hilos necesarios suele ser ya pequeño de por sí, pero en este caso se ve reducido a sólo 3, de los cuales uno es el cable de masa. El cable de masa puede omitirse en todos aquellos casos en los que hay un único circuito impreso, ya que habitualmente existe una única zona de masa común a todos los componentes.

Los puertos utilizados son los siguientes:

- **SCL (SystemClock):** Es la línea de los pulsos de reloj que sincronizan el sistema.
- **SDA (System Data):** Es la línea por la que se mueven los datos entre los dispositivos.
- **GND (Masa)**

Las líneas SDA y SCL son del tipo drenador abierto, es decir, un estado similar al de colector abierto, pero asociadas a un transistor de efecto de campo (o FET). Se deben polarizar en estado alto (conectando a la alimentación por medio de resistores "pull-up") lo que define una estructura de bus que permite conectar en paralelo múltiples entradas y salidas.

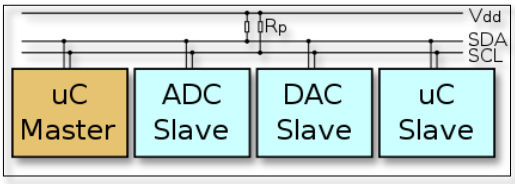

Figura 2.27 Conexión de varios dispositivos en bus I2C
Fuente: http://es.wikipedia.org/wiki/Archivo:I2C.svg

CAPÍTULO III

DESARROLLO DEL TEMA

3.1. PRELIMINARES

En el presente capítulo se detallan los conceptos básicos de programación, es necesario conocer esta información para posteriormente utilizar y aprovechar los beneficios del lenguaje de programación Arduino en su totalidad.

De igual manera se muestra como se realiza la adquisición de señal de cada uno de los elementos acoplados al sistema de ArduCopter, estos sensores varían entre sensores analógicos (ADXL330, IDG500, ISZ500) y sensores digitales (HMC5843, BMP085, GPS MT3329), el método de adquisición y procesamiento de información es distinto para cada uno de ellos, igualmente importante es conocer como se generan las señales PWM desde nuestro controlador para el manejo, en nuestro caso, de los motores DC Brushless.

También se analiza la programación del código de control de ArduCopter, y se detalla la función que realiza cada una de sus secciones (librerías).

De igual manera se muestra paso a paso como realizar la instalación del código de control ArduCopter en el dispositivo, para esto procedemos a utilizar una versión mas ligera del programa original Arduino denominada Arduino Relax, mediante esta versión del software compilador es posible realizar las calibraciones iniciales en nuestro dispositivo, también es posible utilizar el software denominado APM Planner.

El software APM Planner cumple las mismas funciones de configuración y calibración inicial, adicionalmente posee las aplicaciones que permiten configurar a nuestro dispositivo como un vehículo aéreo no tripulado (UAV) mientras visualizamos su localización por medio de mapas satelitales.

3.2. COMPILADOR ARDUINO

El entorno de desarrollo Arduino está constituido por un editor de texto para escribir el código, un área de mensajes, una consola de texto, una barra de herramientas con botones para las funciones comunes, y una serie de menús, adicionalmente permite la conexión con el hardware de Arduino para cargar los programas y comunicarse con ellos.

El programa desarrollado por Arduino es denominado "sketch". Estos programas son escritos en el editor de texto. Existe la posibilidad de cortar/pegar y buscar/remplazar texto. En el área de mensajes se muestra información mientras se cargan los programas y también muestra errores de compilación, la consola muestra el texto de salida para el entorno de Arduino incluyendo los mensajes de error completos y otras informaciones. La barra de herramientas permite verificar el proceso de carga, creación, apertura y guardado de programas, y la monitorización de comunicación serie.

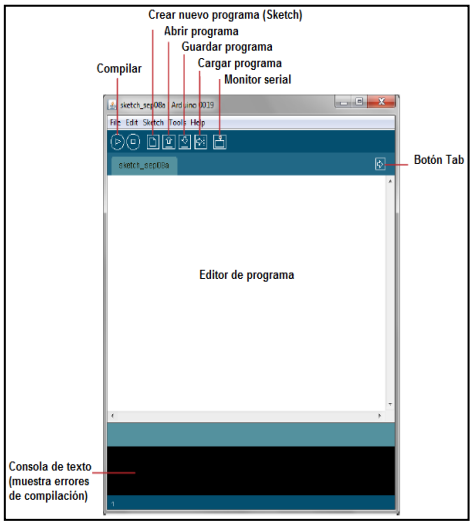

Figura 3.1: Arduino Software Versión 022
Fuente: www.arduino.com

Descripción

- **Verify/Compile (Compilar)**
  Verifica el código en busca de errores.

- **Stop (Parar)**
  Finaliza la monitorización serie y oculta otros botones.

- **New (Nuevo)** 

  Crea un nuevo sketch.

- **Open (Abrir)** 

  Presenta un menú de todos los programas (sketch) de su librería de programas (sketchbook). Un click sobre uno de ellos lo abrirá en la ventana actual.

  Nota: Debido a un error (bug) en Java, la barra de desplazamiento (scroll) no funciona en este menú, si se desea abrir un programa que no se vea en la lista utilice File/Sketchbook en el menú.

- **Save (Guardar)** 

  Salva el programa “sketch”.

- **Upload to I/O Board** 

  Compila el código y lo graba en la placa de Arduino.

- **Serial Monitor** 

  Inicia la monitorización serie.

Menú de Edición (Edit)

- **Copy for Forum**

  Copia el código de su sketch en el portapapeles con el formato original incluyendo la sintaxis coloreada para ser incluido en un foro.

- **Copy as HTML (Copiar con Formato HTML)**

  Copia el código de un programa (sketch) al portapapeles en formato HTML, adecuándolo para introducirlo en una página web.

Menú de Programación (Sketch)

- **Compilar (Verify/Compile)**

  Verifica los errores de su programa (sketch).

- **Import Library (Importar Librería)**
  Añade una librería a su programa (sketch) insertando la sentencia #include en el código.

- **Show Sketch Folder (Mostrar la Carpeta de Programas)**
  Abre la carpeta de programas (sketch) en el escritorio.

- **Add File...(Añadir Fichero)**
  Añade un fichero fuente al programa (se incluirá desde su ubicación actual). El fichero aparece en una nueva pestaña en la ventana del programa. Los ficheros pueden ser quitados del programa (sketch) utilizando el menú "tab".

Menú de Herramientas (Tools)

- **Auto Format**
  Da formato al código proporcionando estética: por ejemplo realiza tabulaciones entre la apertura y cierre de llaves, y las sentencias que tengan que ser tabuladas lo estarán.

- **Board (Tarjeta)**
  Selecciona el modelo de tarjeta que está usando.

- **Serial Monitor (Puerto de Comunicación)**
  Este menú contiene todos los dispositivos serie (real o virtual) de su equipo. Se refrescará automáticamente cada vez que abras el menú tools.

- **Burn Bootloader (Sector de Arranque)**
  Este elemento del menú le permite grabar un gestor de arranque (bootloader) dentro del microcontrolador de la placa Arduino. Aunque no es un requisito para el normal funcionamiento de la placa Arduino, le será útil si compra un nuevo Atmega (el cual viene normalmente sin gestor de arranque). Asegúrese que ha seleccionado la placa correcta en el menú

Boards antes de grabar el bootloader. Cuando use AVR ISP, tendrá que seleccionar en el menú Serial Port el puerto correspondiente.

- **Sketchbook (Carpeta de Programas)**

  El entorno de Arduino incluye el concepto de "sketchbook", que es el lugar estándar para el almacenamiento de sus programas. Los "sketchs" dentro de su "sketchbook" pueden abrirse desde el menú File/Sketchbook o desde el botón de la barra de herramientas Open.

  La primera vez que arranque el software Arduino, se creará un directorio para su "sketchbook". Puede visualizar o cambiar su localización dentro de "sketchbook location" siguiendo el menú File/Preferences**.**

- **Libraries (Librerías)**

  Las librerías proporcionan funcionalidad extra para la utilización en un programa, por ejemplo para trabajar con hardware o manipular datos. Para utilizar una librería en un sketch, seleccione el menú Sketch/Import Library. Esto insertará una o más sentencias #include al principio del sketch y compilará la librería con su sketch.

  Existe una lista de librerías en las referencias. Algunas librerías están incluidas en el software Arduino, otras pueden ser descargadas desde una gran variedad de fuentes. Para instalar estas librerías de terceros, se debe crear un directorio denominado “libraries” en el directorio de sktchbooks, después descomprimir la librería allí. Por ejemplo, para instalar la librería Data Time, sus ficheros deberían estar en una subcarpeta **/**libraries/DateTime en la carpeta de sketchbooks.

- **Serial Monitor (Monitor Serie)**
  Muestra los datos enviados desde la placa Arduino (placa USB o serie).

3.3. ESTRUCTURA DE UN PROGRAMA - SKETCH

El programa o “sketch” es un conjunto de instrucciones para que Arduino las realice, para realizar un “sketch” utilizando Arduino se debe conocer la estructura básica del lenguaje de programación, el lenguaje de programación es bastante simple y se compone de tres partes principales que son la declaración de variables, la función de configuración principal y la función cíclica principal.

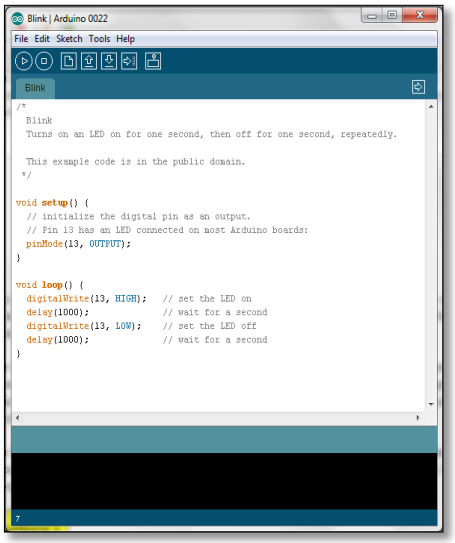

Figura 3.2: Estructura de un Programa Arduino
Fuente: Investigación de Campo

La declaración “Setup()” es la primera función que ejecuta y es la parte encargada de recoger la configuración principal del programa, debe contener la declaración de las variables y la configuración de otras funciones como la habilitación de los puertos analógicos/digitales como entradas o salidas, la habilitación de la comunicación serial entre otras.

La declaración “Loop()” es el núcleo de todos los programas de Arduino ya que contiene la parte de programación que se ejecutará cíclicamente, lo que posibilita que el programa este respondiendo continuamente ante los eventos que se produzcan.

3.4. DIRECTIVAS

Las directivas de pre-procesado comienzan con el símbolo numeral “#” y continúan con un comando específico.

3.4.1. Define (#define)

La directiva “#define” es un componente de programación C muy útil que permite al programador dar un nombre a un valor constante a una variable antes de que se compile el programa. Las constantes definidas en Arduino no aumentan el tamaño que el programa ocupa en la memoria.

Sintaxis

#define(nombre de la constante, valor).

El símbolo numeral “#” es necesario.

Ejemplo:

```
#define ledPin 3                    // el compilador define el valor 3 a la constante ledPin
```

Consejo

No poner punto y coma después de la sentencia #define o se mostrarán errores de compilación.

```
#define ledPin 3;                   // esto es un error
```

De la misma manera, incluir un signo de igualdad después de la sentencia #define también mostrará errores de compilación.

```
#define ledPin  = 3                 // esto también es un error
```

3.4.2. Include (#include)

La directiva “#include” es utilizada para incluir librerías externas en un programa, esto otorga al programador acceso a una gran cantidad de librerías estándar (grupos de funciones ya hechas), y también librerías escritas especialmente para Arduino.

3.5. TIPOS DE DATOS

La Tabla 3.1 muestra el tipo y tamaño de datos con los cuales podemos trabajar en el entorno Arduino.

Tabla 3.1: Tipos de Datos

TIPO DE DATOS	BYTES	RANGO	USO
int	2	-32768 hasta 32767	Representa valores enteros positivos y negativos
unsigned int	2	0 hasta 65535	Representa solo valores positivos
long	4	-2147483648 hasta 2147483647	Representa valores enteros positivos y negativos con un mayor rango numérico
unsigned long	4	0 hasta 4294967295	Representa solo valores positivos con un mayor rango numérico
float	4	3.4028235E+38 hasta -3.4028235E+38	Representa valores numéricos de coma flotante
double	4	3.4028235E+38 hasta -3.4028235E+38	Representa valores numéricos de coma flotante
boolean	1	Verdadero (1) o Falso (0)	Representa valores de verdadero y falso
byte	1	0 hasta 255	Representa solo valores positivos
char	1	0 hasta 255	Almacena caracteres como son A,B,C, etc, entre comillas simples 'A'
string	Es una matriz contínua (array) utilizando datos de tipo char		
array	Es una colección de variables		

Fuente: Investigación de Campo

3.5.1. Variables

Las variables son una forma de nombrar y almacenar datos que pueden variar continuamente para su uso posterior en el programa, tienen un nombre, un valor, y un tipo. Como por ejemplo:

```
int led = 13;
```

Crea una variable de nombre “led”, su dirección es el puerto o pin “13”, y su tipo de dato es “int”. Mas tarde en el programa, se puede hacer referencia a esta variable por su nombre, momento en el que se puede acceder a su valor y utilizarlo. Por ejemplo, en esta sentencia:

```
pinMode (led, OUTPUT);
```

En este caso, en realidad no se necesita utilizar una variable, esta declaración podría funcionar igual de bien que la anterior:

```
pinMode (13, OUTPUT);
```

La ventaja de una variable en este caso es que solo se necesita especificar el número del puerto una única vez, pero no se la puede usar varias veces, de esta manera si se decide cambiar el pin 13 por el 12, solo es necesario hacer el cambio en un punto del código. Además se puede usar un nombre descriptivo que tenga relación con el uso que se va a dar a la variable.

Una variable puede ser declarada en el inicio del programa antes de setup(), localmente a una determinada función e incluso dentro de un bloque como puede ser un bucle. El sitio en el que la variable es declarada determina el ámbito de la misma. Una variable global es aquella que puede ser empleada en cualquier función del programa. Estas variables deben ser declaradas al inicio del programa antes de la función setup(),como por ejemplo:

```
int pin = 13;

void setup()
{
pinMode(pin, OUTPUT);
}
void loop()
{
digitalWrite(pin, HIGH);
}
```

Como se puede observar, la variable “led” se usa tanto en la función setup() como en loop(). Ambas funciones hacen referencia a la misma variable, por lo que un cambio en una afectará al valor que tendrá en la otra, como por ejemplo:

```
int led = 13;                    // asigna la variable led al puerto 13

void setup()
{
led = 12;                        // cambia la asignación del puerto 13 al puerto 12
pinMode(led, OUTPUT);
}
```

Si se tiene que utilizar una variable en una sola función, se la puede declarar de la siguiente forma:

```
void setup()
{
int led = 13;                    // se declara la función como variable local en setup()
pinMode(led, OUTPUT);
digitalWrite(led, HIGH);
}
```

En este caso, la variable “led” sólo podrá ser usada dentro de la función setup(), entonces no se podrá utilizar la variable en otra ocasión si se trata de hacer algo como esto:

```
void loop()
{
digitalWrite(led, LOW);          // incorrecto la variable pin no tiene función aquí
}
```

3.5.2. Constantes

Las constantes que vienen predefinidas en el lenguaje de Arduino se usan para facilitar la lectura de los programas siendo clasificadas en los siguientes grupos mostrados en la Tabla 3.2.

Tabla 3.2: Constantes

Constantes Booleanas	
TRUE	Se define la mayoría de las veces como 1. Cualquier entero que es “no-cero” es TRUE, en un sentido Booleano. Así, en un sentido Booleano, -1, 2 y -200 son todos true.
FALSE	Se define como 0 (cero).
Constantes Digitales	
HIGH	El significado de HIGH (en referencia a un pin) depende de si el pin está configurado como entrada (INPUT) o como salida (OUTPUT). Cuando un pin se configura como entrada (INPUT) usando pinMode, y se lee con digitalRead, el microcontrolador nos retornará HIGH si en el pin hay 3 voltios o más. Un pin puede ser configurado como entrada (INPUT) usando pinMode, y después establecerlo a HIGH con digitalWrite, esto conectará el pin a 5 Voltios a través de una resistencia interna de 20K, resistencia pull-up , la cual establecerá el pin al estado de lectura HIGH a menos que la conectemos a una señal LOW a través de un circuito externo. Cuando un pin se configura como salida (OUTPUT) con pinMode, y se establece a HIGH con digitalWrite, el pin tiene 5V.
	El significado de LOW difiere también según esté configurado como entrada (INPUT) o como salida (OUTPUT). Cuando un pin

LOW	está configurado como entrada (INPUT) con pinMode, y se lee con digitalRead, el microcontrolador retornará LOW si el voltaje presente en el pin es de 2V o menor. Cuando un pin es configurado como salida (OUTPUT) con pinMode,y establecido LOW con digitalWrite, el pin tiene 0 voltios.
Constantes de Definición	
INPUT	Los pines de Arduino (Atmega) configurados como INPUT con pinMode() se encuentran en un estado de alta impedancia, esto lo hace muy útil para leer un sensor, pero no para alimentar un LED.
OUTPUT	Los pins configurados como salida (OUTPUT) con pinMode() se dice que están en estado de baja impedancia. Esto implica que pueden proporcionar una sustancial cantidad de corriente a otros circuitos. Los pines de Atmega pueden proveer de una corriente positiva de hasta 40mA (miliamperios) a otros dispositivos o circuitos. Esto lo hace muy útil para alimentar LED's pero inservible para leer sensores.

Fuente: Investigación de Campo

3.5.3. Operadores

Tabla 3.3: Clasificación de Operadores

Operadores Comparativos	
==	Igual a
¡=	Diferente de
<	Menor que
>	Mayor que
<=	Menor o igual que
>=	Mayor o igual que
Operadores Booleanos	
&&	"Y" (AND)
‖	"O" (OR)
¡	Negación (NOT)
-=	Composición Resta
Operadores Aritméticos	
=	Asignación
+	Suma
-	Resta
*****	Multiplicación
/	División
%	Residuo
Operador de Composición	
++	Incremento
--	Decremento
+=	Composición Suma
-=	Composición Resta
***=**	Composición Multiplicación
/=	Composición División
Sintaxis	
;	Punto y coma
{}	Llaves
//	Comentarios en una línea
//**	Comentarios en múltiples líneas

Fuente: Investigación de Campo

3.6. FUNCIONES

El caso típico para crear una función es cuando el usuario necesita realizar la misma acción múltiples veces dentro de un mismo programa.

La estandarización de fragmentos de código en funciones tiene diversas ventajas:

- Las funciones ayudan al programador a ser organizado, además ayudan a conceptualizar el programa
- Las funciones codifican una acción en un lugar, así que sólo deben ser depuradas de errores una vez
- Reducen las posibilidades de error
- Las funciones hacen el programa más pequeño y más compacto por que las secciones de código se reutilizan varias veces
- Hace más fácil la reutilización de código en otros programas por hacerlo más modular y, como efecto paralelo, usando funciones se obtiene un código mas comprensible

Hay dos funciones necesarias en un sketch de Arduino, setup() y loop(), el resto de funciones debe ser definido fuera de las llaves de estas dos funciones.

3.6.1. Función E/S Digital

3.6.1.1. pinMode()

Descripción

Configura el puerto o pin especificado para comportarse como una entrada o una salida digital 0 o 1(0 - 5VDC, 0 - 3.3VDC en los nuevos dispositivos).

Sintaxis

pinMode(“pin”, “modo”)

Parámetros

Pin: Es el número de puerto que se desea configurar.

Modo: INPUT (Entrada) o OUTPUT (Salida).

3.6.1.2. digitalWrite()

Descripción

Escribe un valor HIGH o LOW hacia un pin digital. Si el pin ha sido configurado como OUTPUT con pinMode(), su voltaje será establecido al correspondiente 5V o 3.3VDC para HIGH, 0VDC para LOW.

Si el pin es configurado como INPUT, escribir un valor de HIGH con digitalWrite() habilitará una resistencia interna de 20Kohm de pull-up. Escribir LOW invalidará la resistencia.

Sintaxis

- digitalWrite("pin","valor")

Parámetros

Pin: Es el número del pin digital que se desea configurar.

Valor: HIGH o LOW.

Ejemplo:

```
int led = 13;                    // se crea una variable "led", el pin digital 13
                                 es asignado a la variable

void setup()
{
pinMode(led, OUTPUT);            // configura el pin como salida
}
void loop()
{
digitalWrite(led, HIGH);         // se emite una señal HIGH, si se conecta un
                                 LED al puerto 13 este se iluminara
delay(1000);                     // espera un segundo
digitalWrite(led, LOW);          // apaga el LED
delay(1000);                     // espera un segundo
}
```

3.6.1.3. digitalRead()

Descripción

Lee los valores digitales de un pin especificado, HIGH o LOW.

Sintaxis

- digitalRead(“pin”)

Parámetros

Pin: Es el número de pin digital que se desea leer.

Ejemplo:

```
int led = 8;                    // se crea una variable “led”, se conecta un
                                LED al pin número 8
int push = 7;                   // se conecta un botón al pin número 7
int val = 0;                    // variable donde se almacena el valor leído

void setup()
{
pinMode(led, OUTPUT);           // establece el pin número 13 como salida
pinMode(push, INPUT);           // establece el pin número 7 como entrada
}

void loop()
{
val = digitalRead(push);        // leer el pin de entrada
digitalWrite(led, val);         // establece el LED al valor del botón
}
```

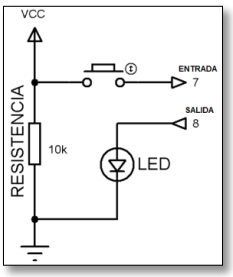

Figura 3.3: Configuración Física de Entrada Digital
Fuente: Investigación de Campo

3.6.2. Función E/S Analógica

3.6.2.1 analogRead()

Descripción

Lee el valor de tensión en el pin analógico especificado. La placa Arduino Mega posee 16 canales conectados a un conversor analógico digital con resolución de 10 bits. Esto significa que convertirá tensiones entre 0 y 5 voltios a un número entero entre 0 y 1023. Esto proporciona una resolución en la lectura de 5 voltios/1023 unidades, es decir, 0.0049 voltios (4.9mV) por unidad.

El conversor tarda aproximadamente 100 microsegundos (0.0001 segundos) en leer una entrada analógica por lo que se puede llevar una tasa de lectura máxima aproximada de 10.000 lecturas por segundo.

Sintaxis

- analogRead(“pin”)

Parámetros

Pin: Indica el número del pin de la entrada analógica que deseamos leer (de 0 a 15 en la tarjeta Arduino Mega).

3.6.2.2. analogWrite()

Descripción

Escribe un valor analógico (PWM) en un pin o puerto, puede ser usado para controlar la luminosidad de un LED o la velocidad de un motor DC. Después de llamar a la función analogWrite()**,** el pin generará una onda cuadrada estable con el ciclo de trabajo especificado hasta que se vuelva a llamar a la función analogWrite() (o una llamada a las funciones digitalRead() o digitalWrite() en el mismo pin). La frecuencia de la señal PWM será de aproximadamente 490Hz, en la placa Arduino Mega, se puede utilizar los pines desde el 2 hasta el pin 13.

Sintaxis

- analogWrite(“pin”, “valor”)

Parámetros

Pin: Es el pin en el cual se quiere generar la señal PWM.

Valor: El ciclo de trabajo deseado comprendido entre 0 (siempre apagado) y 255 (siempre encendido).

Ejemplo:

Crear un programa que genere una señal PWM para regular la luminosidad de un LED, el ciclo de trabajo es proporcional al voltaje leído desde un potenciómetro.

```
int led = 10;                    // se crea una variable “led”, se conecta un
```

```
                                        LED al pin número 10

int analog = 0;                         // se crea una variable “analog”, se conecta un
                                        potenciómetro al pin 0
int val = 0;                            // se crea una variable “val” para almacenar la
                                        información obtenida de voltaje


void setup()
{
pinMode(led, OUTPUT);                   // establece el pin número 10 como salida
}

void loop()
{
val = analogRead(analog);               // lee el voltaje que ingresa en el pin 10
analogWrite(led, val/4);                // los valores de analogRead van desde 0 a
                                        1023 y los valores de analogWrite van desde 0
                                        a 255, por eso ajustamos el ciclo de trabajo a
                                        el valor leído dividido para 4


}
```

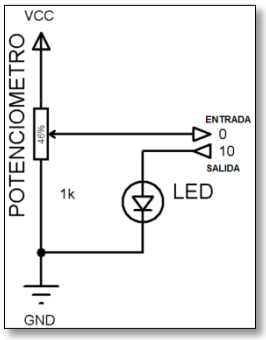

Figura 3.4: Configuración Física de Lectura Analógica
Fuente: Investigación de Campo

3.6.2.3. analogReference()

Descripción

Configura el voltaje de referencia usado por la entrada analógica. La función analogRead() devolverá un valor de 1023 para aquella tensión de entrada que sea igual a la tensión de referencia. Las opciones son:

- **DEFAULT:** Es el valor de referencia analógico que viene por defecto 5 voltios en placas Arduino antiguas y de 3.3 voltios en las nuevas placas Arduino.
- **INTERNAL:** Es una referencia de tensión interna de 1.1 voltios en el Atmega168 o Atmega328 y de 2.5 voltios en el Atmega8.
- **EXTERNAL:** Se usará una tensión de referencia externa que tendrá que ser conectada al pin AREF.

Parámetros

Tipo: El tipo de referencia que se desea usar (DEFAULT, INTERNAL, o EXTERNAL)

Precaución

Es recomendable que cuando se use la referencia de tensión externa se conecte al pin AREF usando una resistencia de 5K. Esto evitará posibles daños internos en el Atmega si la configuración de la referencia analógica es incompatible con el montaje físico que se ha llevado a cabo. Tenga en cuenta que esa resistencia afectará la tensión que se use como tensión de referencia ya que el pin AREF posee una resistencia interna de 32K. Ambas resistencias forman un divisor de tensión, por lo tanto, si por ejemplo se conecta una tensión de 2.5V con la resistencia de 5K la tensión que se utilice como tensión de referencia será de 2.2V ya que esa será la tensión que caiga en el pin AREF debido al divisor de tensión al que se ha hecho referencia.

El conectar la tensión de referencia externa a través de una resistencia permite configurar mediante software la utilización de la tensión de referencia interna a una tensión de referencia externa sin que la configuración física del hardware afecte la configuración actual del conversor A/D.

Uso del pin AREF

La tensión aplicada en el pin AREF será la encargada de que el conversor A/D de su máxima lectura (1023) cuando lea una tensión igual a la aplicada en ese pin. Tensión por debajo de esa tensión conectada a AREF será escalada proporcionalmente, así cuando se usa la tensión de referencia por defecto (DEFAULT) el valor que nos devuelve una tensión de 2.5V en una entrada analógica será 512.

La configuración por defecto del Arduino es la de no tener nada conectado de forma externa al pin AREF. En este caso la configuración de la tensión de referencia será DEFAULT lo cual conecta AVCC (alimentación positiva +5V) de forma interna al pin AREF. Este pin es un pin de baja impedancia (mucha corriente) por lo que si usando la configuración DEFAULT de la tensión de

referencia se conecta otra tensión que no sea la que posee AVCC, podría dañar el Atmega.

El pin AREF también puede conectarse de forma interna a una fuente de referencia de 1.1 voltios (o 2.5 en el Atmega8) usando el comando analogReference(INTERNAL). Con esta configuración las tensiones aplicada a los pines analógicos que sean iguales a la tensión de referencia interna o superior nos devolverán 1023 cuando se lleva a cabo su lectura con la función analogRead(). Tensiones más bajas devolverán valores proporcionales, por ejemplo, si tenemos una tensión de 0.55 voltios en el pin analógico, nos devolverá 512.

La conexión entre la fuente de 1.1 voltios y el pin AREF es de muy alta impedancia (baja corriente), por lo que la lectura de la tensión de 1.1 voltios solo se podrá hacer con un multímetro que tenga alta impedancia de entrada. Si se aplicase por error una tensión de referencia externa en el pin AREF mientras se está usando la configuración INTERNAL para la tensión de referencia, no dañará el microcontrolador pero hará que la tensión de referencia sea la externa y no la de 1.1 voltios de la fuente interna. Aún así es recomendable que cualquier tensión externa que se conecte al pin AREF se haga a través de una resistencia de 5K para evitar el problema mencionado arriba.

La correcta configuración del software cuando se utiliza una tensión de referencia externa se hace mediante la función analogReference(EXTERNAL). Eso desconectará ambas tensión de referencia internas del pin AREF y por tanto será la externa la cual gobierne la tensión máxima leída por el ADC.

3.6.3. Funciones de Tiempo

3.6.3.1. delay()

Descripción

Pausa el programa por un tiempo determinado (en milisegundos) especificado por un parámetro. Hay 1000 milisegundos en un segundo.

Sintaxis

- delay(“tiempo en ms”)

Parámetros

ms: el número de milisegundos que se desea pausar el programa.

Ejemplo:

```
int led = 13;                    // se crea una variable “led”, se conecta un LED al pin número 13

void setup()
{
pinMode(led, OUTPUT);            //declara que el pin digital se va a usar como salida
}
void loop()
{
digitalWrite(led, HIGH);         //enciende el LED
delay(1000);                     //espera durante un segundo (1000 milisegundos)
digitalWrite(led, LOW);          //apaga el LED
delay(1000);                     //espera durante un segundo (1000 milisegundos)
}
```

3.6.3.2. delayMicroseconds()

Descripción

Detiene brevemente el programa por la cantidad en tiempo (en microsegundos) especificada como parámetro. Existen mil microsegundos en un milisegundo, y un millon de microsegundos en un segundo.

Sintaxis

- delayMicroseconds(“tiempo es us”)

Parametros

us: el número de microsegundos a detener.

Ejemplo:

```
int out = 8;                     // se crea la variable “out” y se selecciona el pin digital 8

void setup()
```

```
{
pinMode(out, OUTPUT);                // establece el pin digital como salida
}

void loop()
{
digitalWrite(out, HIGH);             // establece el encendido del pin
delayMicroseconds(50);               // espera por 50 microsegundos
digitalWrite(out, LOW);              // establece el encendido del pin
delayMicroseconds(50);               // espera por 50 microsegundos
}
```

3.6.4. Funciones Matemáticas

3.6.4.1. min() (mínimo)

Descripción

Calcula el mínimo de dos números (x, y).

Parámetros

x: El primer número guardará cualquier tipo de dato.

y: El segundo número guardará cualquier tipo de dato.

Ejemplo:

```
valmin = min(valmin, 512);           // se crea una variable denominada “valmin”, el valor de esta variable será sensado siempre que sea menor a 512
```

3.6.4.2. max() (máximo)

Descripción

Calcula el máximo de dos números (x, y).

Parámetros

x: El primer número guardará cualquier tipo de dato.

y: El primer número guardará cualquier tipo de dato.

Ejemplo:

```
valmax = max(val, 512);              // se crea una variable denominada “valmax”, el valor de esta variable será sensado siempre que sea mayor a 512
```

El valor max() suele ser usado para restringir el valor más bajo del rango de una variable, mientras que min() es utilizado para restringir el valor máximo del rango.

3.6.5. Función Comunicación Serial

Se utiliza para la comunicación entre la placa Arduino y un ordenador, todas las placas Arduino tienen al menos un puerto serie (también conocido como UART o USART). Se comunica con el ordenador a través de los pines digitales 0 (RX) y 1 (TX), por lo tanto no se puede usar los pines 0 y 1 como entrada o salida digital, se puede utilizar el monitor serie incorporado en el compilador de Arduino para comunicarse con la placa Arduino.

La placa Arduino Mega tiene tres puertos adicionales de serie: Serial1 en los pines 19 (RX) y 18 (TX), Serial2 en los pines 17 (RX) y 16 (TX), Serial3 en los pines 15 (RX) y 14 (TX) para comunicarse con el ordenador personal, pero sólo el pin 0 y 1 de comunicación serie posee una adaptador USB. Para utilizar estos puertos es necesario un dispositivo serie externo TTL, se debe conectar el pin TX al pin RX del dispositivo, el pin RX al pin TX del dispositivo y el pin GND del Arduino Mega al pin GND del dispositivo. (No se debe conectar estos pines directamente a un puerto serie RS232, que operan a +/- 12V ya que esto puede dañar la placa Arduino.)

3.6.5.1. begin()

Descripción

Establece la velocidad de datos en bits por segundo (baudios) para la transmisión de datos en serie. Para comunicarse con el computador, utilice una de estas velocidades: 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 o 115200 bps.

Sintaxis

- Serial.begin(“velocidad”)

Solamente en Arduino Mega 2560 ya que posee 4 interfaces de comunicación serie:

- Serial.begin(“velocidad”)
- Serial1.begin(“velocidad”)
  Serial2.begin(“velocidad”)
  Serial3.begin(“velocidad”)

Parámetros

velocidad: Velocidad en bits por segundo (baudios).

Ejemplo:

```
// Arduino Mega usando sus 4 puertos serie (Serial, Serial1, Serial2, Serial3),
con diferentes velocidades de transmisión de datos

void setup()
{
Serial.begin(4800);
Serial1.begin(9600);
Serial2.begin(19200);
Serial3.begin(38400);
}
void loop()
{
}
```

3.6.5.2. print()

Descripción

Imprime los datos al puerto serie como texto ASCII. Este comando puede tomar muchas formas. Los números son impresos mediante un juego de caracteres ASCII para cada dígito. Los valores de tipo "float" son impresos en forma de dígitos ASCII con dos decimales por defecto. Los valores tipo "byte" se envían como un único carácter. Los caracteres y las cadenas se envían tal cual. Por ejemplo:

Serial.print(78) imprime "78"

Serial.print(1.23456) imprime "1.23"

Serial.print(byte(78)) imprime "N" (cuyo código ASCII es 78)

Serial.print('N') imprime "N"

Serial.print("Hello world.") imprime "Hello world."

Un segundo parámetro opcional especifica la base (formato) a usar, los valores permitidos son BYTE, BIN (binarios), OCT (octales), DEC (decimales), HEX (hexadecimales). Para números de coma flotante, este parámetro especifica el numero de posiciones decimales a usar. Por ejemplo:
Serial.print(78, BYTE) imprime "N"
Serial.print(78, BIN) imprime "1001110"
Serial.print(78, OCT) imprime "116"
Serial.print(78, DEC) imprime "78"
Serial.print(78, HEX) imprime "4E"
Serial.println(1.23456, 0) imprime "1"
Serial.println(1.23456, 2) imprime "1.23"
Serial.println(1.23456, 4) imprime "1.2346"

Sintaxis

- Serial.print(“valor”)
- Serial.print(“valor”, “formato”)

Parámetros

valor: El valor a imprimir de cualquier tipo.
formato: Especifica el número de la base (para números enteros) o el número de posiciones decimales (para números de coma flotante o tipo "float")

3.6.5.3. println()

Descripción

Imprime los datos al puerto serie como texto, este comando tiene la misma forma que Serial.print().

Sintaxis

- Serial.println(“valor”)
- Serial.println(“valor”, “formato”)

Parámetros

valor: el valor a imprimir (de cualquier tipo)

formato: Especifica el número de la base (para números enteros) o el número de posiciones decimales (para números de coma flotante o tipo "float")

Ejemplo:

Con los conocimientos adquiridos desarrollar un programa que simule un voltímetro digital utilizando el puerto AREF para obtener el voltaje de referencia externo de 0 a 5V, sensando sus valores máximos y mínimos, los valores resultantes de la conversión A/D deben ser transmitidos mediante la utilización de los puertos seriales, un puerto para cada valor a diferente velocidad de transmisión (9600, 19200, 38400, 57600 baudios), los valores a obtener son: el valor de referencia de la conversión A/D, el valor calibrado en voltaje, el valor máximo y el valor mínimo.

```
// Voltímetro Digital 0 - 5V

int analoginput = 0;                          // puerto analógico utilizado pin 0
float voltaje;                                // se crea una variable denominada “voltaje”
float valor;                                  // se crea una variable denominada “valor”
int valmin;                                   // se crea una variable denominada “valmin”
int valmax;                                   // se crea una variable denominada “valmax”

void setup()
{
analogReference(EXTERNAL);                    // se activa la función AREF para obtener el
                                              voltaje de referencia externo +5V
Serial.begin(9600);                           // habilita los puerto seriales a diferentes
                                              velocidades de transmisión

Serial1.begin(19200);
Serial2.begin(38400);
Serial3.begin(57600);
}
void loop()
{
valor = analogRead(analoginput);              // la variable valor lee los datos del puerto
                                              configurado como entrada analógica
voltaje = (valor/205.10);                     // la variable voltaje acondiciona la variable
                                              "valor” para muestrear el voltaje en forma
                                              digital

valmin=valor;
valmin = min (valmin,512);                    // se configura el valor mínimo
valmax=valor;
valmax = max (valmax,512);                    // se configura el valor máximo
Serial.print("voltage (V): ");                      // se transmiten los datos de voltaje por
                                                    el puerto de comunicación serial 0

Serial.println(voltaje,2);

Serial1.print("valor en entrada analogica: ");       // se transmiten los datos resultantes
                                                    del acondicionamiento A/D por el
                                                    puerto de comunicación serial 1

Serial1.println(valor);
```

```
Serial2.print("valor mínimo: ");          // se transmiten los valores mínimos del acondicionamiento A/D por el puerto de comunicación serial 2

Serial2.println(valmin);

Serial3.print("valor máximo: ");          // se transmiten los valores máximos del acondicionamiento A/D por el puerto de comunicación serial 3

Serial3.println(valmax);
delay(800);
}
```

3.6.5.4. read()

Descripción

Lee los datos entrantes del puerto serie.

Sintaxis

- Serial.read ()

Solamente en Arduino Mega:

- Serial.read ()
- Serial1.read ()
- Serial2.read ()
- Serial3.read ()

Ejemplo:

```
int letra = 0;                          // se crea una variable denominada “letra”

void setup()
{
Serial.begin(9600);                     // habilita el puerto serie a 9600 bps
}

void loop()
{                                       // envía datos solamente cuando recibe datos
if (Serial.available() > 0)
{
                                        // lee el byte entrante del puerto serial 0
letra = Serial.read();

                                        // transmite la información recibida del puerto serial 0 hacia el puerto serial 1
Serial.print("He recibido: ");
Serial.println(letra, BYTE);            // la información se muestra en formato BYTE
}
}
```

3.6.5.5. write()

Descripción

Escribe y transmite diferentes tipo de datos por el puerto serie. Estos datos se envían como un byte o una serie de bytes, para enviar los caracteres que representan los dígitos de un número se utiliza la función print().

Sintaxis

- Serial.write(val)
- Serial.write(str)
- Serial.write(buf, len)

Parámetros

val: Un valor enviado como un solo byte.
str: Una cadena “string” para enviar una serie de bytes.
buf: Un arreglo “array” para enviar una serie de bytes.
len: Tamaño del buffer.

3.6.5.6. available()

Descripción

Devuelve el número de bytes (caracteres) disponibles para ser leídos por el puerto serie. Se refiere a datos ya recibidos y disponibles en el buffer de recepción del puerto.

Sintaxis

- Serial.available()

Sólo para Arduino Mega:

- Serial.available()
- Serial1.available()
- Serial2.available()
- Serial3.available()

Ejemplo:

Crear un programa el cual transmita los datos disponibles del puerto serie 0 hacia el puerto serie 1.

```
int inByte = 0;

void setup()
{
Serial.begin(9600);
Serial1.begin(9600);
}

void loop()
{
if (Serial.available())
{
int inByte = Serial.read();
Serial1.print(inByte, BYTE);

}
                                        // lee la información disponible desde el puerto
                                        serial 0 y envía los datos al puerto serial 1

if (Serial1.available())
{
int inByte = Serial1.read();
Serial.print(inByte, BYTE);
}
}
```

3.6.5.7. flush()

Descripción

Vacía el búfer de entrada de datos del puerto serie.

Sintaxis

- Serial.flush()

Solamente en Arduino Mega:

- Serial1.flush()
- Serial2.flush()
- Serial3.flush()

3.6.5.8. end()

Descripción

Desactiva la comunicación serie, permitiendo a los pines RX and TX ser usados como entradas o salidas digitales. Para reactivar la comunicación serie se debe habilitar la instrucción Serial.begin().

Sintaxis

- Serial.end()

Solamente en Arduino Mega:

- Serial.end()
- Serial1.end()
- Serial2.end()
- Serial3.end()

3.7. ESTRUCTURAS DE CONTROL

3.7.1. Condicional IF

La estructura condicional “if”, puede ser usada en conjunto con uno o más operadores de comparación y su función lógica determina que si la declaración escrita dentro de los paréntesis es verdadera el código dentro de las llaves se ejecutará. Si no, el programa ignora dicho código.

Las llaves pueden ser omitidas luego de una declaración “if”. De hacer esto, la siguiente línea (definida por el punto y coma) será la única afectada por la condición.

Sintaxis

Todas las formas de escritura mostradas son correctas.

- if (x > 120) digitalWrite(LED, HIGH);
- if (x > 120)
  digitalWrite(LED, HIGH);
- if (x > 120){ digitalWrite(LED, HIGH); }
- if (x > 120)
  {
  digitalWrite(LED1, HIGH);

```
digitalWrite(LED2, HIGH);
}
```

3.7.2. Condicional IF/ELSE

El condicional “if/else”, permite mayor control sobre el flujo del código que la declaración “if” básica, por permitir agrupar múltiples comprobaciones, por medio del condicional “else” se evalúa la expresión declarada y si es verdadera se ejecuta la sentencia primaria, de caso contrario se ejecuta la sentencia secundaria.

Sintaxis

-
```
if (variable < 500)
{
        // ejecutar A
}
else if (variable >= 1000)
{
        // ejecutar B
}
else
{
        // ejecutar C
}
```

Ejemplo:

```
int led = 13;
int push = 14;

void setup()
{
Serial.begin(9600);
pinMode(led, OUTPUT);
pinMode(push, INPUT);
digitalWrite(led, LOW);
}
void loop()
{
```

```
if (digitalRead(push)== HIGH)          // si la lectura digital de “push” es alta (1 lógico) se enciende el LED y se transmite la señal “led on”

{
digitalWrite(led, HIGH);
Serial.print("LED ON");
delay(100);
}
else                                   // de lo contrario el LED permanece apagado y se transmite la señal “led off”

{
digitalWrite(led, LOW);
Serial.println("LED OFF");
delay(100);
}
}
```

3.7.3. Declaración FOR

Descripción

La inicialización se produce sólo la primera vez. Cada vez que se va a repetir el bucle, se revisa la condición. Si es verdadera, el bloque de funciones y el incremento del contador se ejecutan, y la condición vuelve a ser comprobada de nuevo. Si la condición es falsa, el bucle termina.

El bucle “for” tiene tres partes o argumentos en su inicialización:

for (inicialización; condición de finalización; incremento)

Ejemplo:

Crear un programa que varié la intensidad de un LED mediante una señal PWM, esta señal debe variar automáticamente utilizando la declaración FOR.

```
int PWM = 10;                          // se crea la variable “PWM” para el puerto digital 10

void setup()
{
                                       // no es necesario nada aquí
}

void loop()
{
for (int i=0; i <= 255; i++)           // se crea una variable local “i”, la variable “i” inicia en 0, se ejecuta el condicional mientras “i” sea menor o igual a 255

{
analogWrite(PWM, i);                   // se emite la señal PWM
delay(50);
```

```
}
}
```

3.7.4. Sentencia SWITCH/CASE

Como la declaración IF, “switch...case” controla el flujo del programa permitiendo al usuario crear diferentes códigos los cuales son ejecutados en función de varias condiciones. En particular, una sentencia “switch” compara el valor de una variable con el valor especificado en la sentencias “case”. Cuando se encuentra una sentencia “case” cuyo valor coincide con dicha variable, el código de esa sentencia se ejecuta.

La función “break” tiene como función salir de la sentencia “switch” y es usada típicamente al final de cada “case”. Sin la sentencia “break”, la sentencia “switch” continuaría ejecutándose.

Sintaxis

- switch (valor)

```
{
case etiqueta 1:
        // sentencias
break;
case etiqueta 2:
        // sentencias
break;
default:
        // sentencias
}
```

Parámetros

valor: La variable cuyo valor lo compara con los diferentes “case”.

etiqueta: Es el valor que se compara con la variable.

Ejemplo:

```
int led1 = 12;
int led2 = 13;
int key  = 0;
```

```
void setup()
{
Serial.begin(9600);
Serial1.begin(9600);
pinMode(led1, OUTPUT);
pinMode(led2, OUTPUT);
Serial.print("INTRODUZCA LA CLAVE:  ");
}

void loop()
{
if(Serial.available() > 0)
{
key = Serial.read();

switch(key)
{
case '0':
Serial1.println("Clave Correcta");
digitalWrite(led1, HIGH);
break;

case '1':
Serial1.println("Bienvenido");
digitalWrite(led2, HIGH);
delay(500);
break;

default:
Serial1.println("Clave Incorrecta");
digitalWrite(led1, LOW);
digitalWrite(led2, LOW);
delay(500);
}
}
}
```

3.7.5. Condicional WHILE

Descripción

Los bucles “while” se ejecutan continuamente hasta que la expresión dentro del paréntesis (), pasa a ser falsa. Algo debe modificarse en la variable comprobada o de lo contrario el bucle “while” nunca terminará de ejecutarse. Lo que modifique la variable puede estar en el código, como una variable que se incrementa, o ser una condición externa, como el valor que da un sensor.

Sintaxis

- while(expresión)

  {

```
        // sentencia(s)
}
```

Ejemplo:

```
int i=0;                          // se crea una variable “i”
void setup()
{
Serial.begin(9600);               // habilita el puerto serial 0 a 9600bps
}
void loop()
{
while(i<10)                       // se ejecuta la condición de conteo mientras “i”
                                  sea menor a 10
{
Serial.println(i, DEC);           // la variable “i” es transmitida en formato
                                  decimal
i++;                              // lo mismo que escribir (i=i+1;)
delay(500);
}
}
```

3.7.6. Condicional DO/WHILE

El bucle "'do"' trabaja de la misma manera que el bucle "'while"', con la excepción de que la condición se comprueba al final del bucle, por lo que este bucle se ejecuta siempre al menos una vez.

Sintaxis

- do

```
{
        // sentencia
} while (condición);
```

Ejemplo:

```
void setup()
{
Serial.begin(9600);               // habilita el puerto serial 0 a 9600bps
}
void loop()
{
int i = 0;
do
{
 i++;
Serial.println(i, DEC);
delay(500);
```

```
}
while(i<20);
i=0;                                  // se ejecuta la condición de conteo mientras “i” sea menor a 20

}
```

3.7.7. Break

Es comúnmente usado para salir de los bucles (do, for, o while), pasando por alto la condición normal del bucle, es usado también para salir de la estructura de control switch.

Ejemplo:

```
int PWM=13;                           // declaración de variables
int sens=0;
int pot=0;
int umbral=512;

void setup()
{
pinMode(PWM, OUTPUT);                 // declaración de modos
pinMode(pot, INPUT);
Serial.begin(9600);
}
void loop()
{
for (int x = 0; x < 255; x ++)        // declaración del bucle FOR
{
analogWrite(PWM, x);
sens = analogRead(pot);
Serial.println(sens, DEC);
if (sens > umbral)                    // declaración del condicional IF
{
x = 0;
delay(50);
break;                                // si la condición se cumple sale del bucle FOR
}
delay(50);
}
}
```

3.7.8. Continue

La sentencia “continue” omite el resto de interacciones de un bucle (do, for, o while). Continúa saltando a la condición de bucle y procediendo con la siguiente interacción.

Ejemplo:

```
int PWM = 13;
int x = 0;
void setup()
{
Serial.begin(9600);
pinMode(PWM, OUTPUT);
}
void loop()
{
for (x = 0; x < 255; x ++)
{
if (x > 40 && x < 120){                    // crea un salto en estos valores
continue;
}

analogWrite(PWM, x);
Serial.println(x, DEC);
delay(150);
}
}
```

3.8. LIBRERÍAS

Una librería es una colección de instrucciones diseñadas para ejecutar una o varias tareas específicas, una librería debe describir de una manera eficaz todas las preguntas con respecto al qué, cuándo y dónde, pero debe ocultar la forma.

Características de las librerías:

- Simplifica el uso y organización del código
- Mejora la legibilidad del código
- Descentralizar la lógica

3.8.1. Elaboración de Librerías Arduino

Esta información nos ayudará a convertir las principales funciones de un programa en una librería. Esto permite que diferentes usuarios usen el código que se ha elaborado y puedan utilizarlo fácilmente conjuntamente con otros programas.

Programa para realizar una lectura de temperatura utilizando el sensor LM35:

```
int sensor = A0;

void setup()
{
```

```
pinMode(sensor, INPUT);
Serial.begin(9600);
Serial.println("LIBRERIA LM35");
Serial.println("TEMPERATURA: ");
}
void loop()
{
centigrados();
farenheit();
delay(400);
}
void centigrados()
{
sensor=analogRead(A0);
sensor=map (sensor,0,1023,0,500);
Serial.print(sensor);Serial.print(" oC  ");
}
void farenheit()
{
sensor=analogRead(A0);
sensor=map (sensor,0,1023,32,930);
Serial.print(sensor);Serial.print(" oF");
Serial.println();
}
```

Si se compila este programa se realizará la adquisición de señal del sensor analógico de temperatura LM35 mediante el puerto analógico 0 (pin 97 del Atmega 2560). En primer lugar, tenemos las funciones de centígrados () y farenheit () que realizan la función de adquirir la señal y calibrarla para transmitir su valor en función de sus rangos de temperatura (centígrados y farenheit). En segundo lugar, tenemos la variable “sensor” que indica el pin a utilizar y la utilización de la instrucción pinMode () que inicializa el pin en modo de entrada.

Para elaborar una librería se necesitan dos archivos principales: el primero es el archivo de cabecera (extensión.h) y el código fuente (extensión.cpp). El archivo de cabecera contiene las definiciones para la librería, es básicamente un listado de todo lo que hay dentro de la misma, mientras que el archivo del código fuente tiene el código real. Vamos a llamar a nuestra librería "LM", por lo que nuestro archivo de cabecera será “LM.h”.

El archivo de cabecera consiste básicamente en la definición de la librería por medio de un “class” y adicionalmente una línea de programación para cada función de la librería, conjuntamente con las variables que se van a usar, un “class” es un contenedor de uno o más datos (variables o propiedades) junto a las

operaciones de manipulación de dichos datos, fundamentalmente contiene el estado y el comportamiento del concepto que representa, de este modo obtenemos lo siguiente:

```
class LM                                    // se define la “class” de nombre LM
{
public:                                     // se definen las variables públicas
LM(float pin);                              // la variable de la librería LM será de tipo
                                            flotante “float”
void centigrados();                         // se definen las instancias
void farenheit();

private:                                    // se definen las variables privadas
int _pin;
};
#endif
```

Un “class” es simplemente una colección de funciones y variables agrupadas en un mismo lugar. Estas funciones y variables pueden ser públicas, lo que significa que pueden ser modificadas por el usuario que está utilizando la librería, o privada, lo que significa que sólo se puede acceder a ellas desde la propia definición “class”. Cada “class” tiene una función especial conocida como constructor, que se utiliza para crear una instancia de la “class” (o sea, un objeto), el constructor tiene el mismo nombre que la “class”.

Se necesitan adicionar otros elementos más en el archivo de encabezado, una de ellas es una instrucción “# include” que da acceso a los tipos de librerías estándar y las constantes del lenguaje Arduino (esto se agrega automáticamente a los programas normales, pero no a las librerías), y se coloca antes de la definición de la “class” mostrada anteriormente, finalmente obtenemos el archivo de cabecera completo de la siguiente forma:

```
/*
  LM.h - Librería LM35
/
#ifndef  LM_h
#define LM_h
#include "WProgram.h"

class LM
{
public:
LM(float pin);
void centigrados();
void farenheit();

private:
int _pin;
```

```
};
#endif
```

Para construir el código fuente (.cpp) lo primero es adicionar lo siguiente y con esto el resto del código tendrá acceso a las funciones estándar de Arduino, y a las definiciones escritas en la librería “LM.h”.

```
#include "WProgram.h"
#include "LM.h"
```

A continuación viene el constructor. Una vez más, en el constructor establece lo que debe ocurrir cuando alguien crea una instancia o solicitud en la “class”. En este caso, el usuario especifica el pin que le gustaría utilizar. Configuramos el pin como entrada en una variable privada para su uso en las otras funciones:

```
LM::LM (float pin);
(pinMode (pin, INPUT);
_pin = pin;
```

La primera función es “LM::” antes del nombre de la función. Esto indica que la función es parte de la “class de LM”, lo segundo es el subrayado en el nombre de nuestra variable privada “_pin”. Esta variable puede tener cualquier nombre, siempre y cuando coincida con la definición que figura en el archivo de encabezado. Agregar un subrayado al inicio del nombre es un arreglo común para dejar claro que las variables son privadas.

Después viene el código del programa que estamos convirtiendo en una librería:

```
void LM::centigrados()                  // definición de la instancia centígrados
{
float tempc = 0;                        // se crea una variable local “tempc” en la
                                        instancia centígrados
tempc = analogRead(_pin);               // la variable adquiere la señal entrante del
                                        puerto “_pin”
tempc = (tempc/205.18);                 // la información obtenida es modificada al
                                        rango de temperatura en grados centígrados
tempc = tempc * 100;
Serial.print("TEMPERATURA GRADOS CENTIGRADOS: ");
Serial.println(tempc,2);                // se transmite la temperatura
delay(500);
}
void LM::farenheit()                    // definición de la instancia farenheit
{
```

```
float tempfar = 0;                          // se crea una variable local “tempfar” en la
                                            instancia farenheit
tempfar = analogRead(_pin);                 // la variable adquiere la señal entrante del
                                            puerto “_pin”
tempfar = ((tempfar/205.18)*100);           // la información obtenida es modificada al
                                            rango de temperatura en grados farenheit
   tempfar = ((tempfar*1.8)+32);
Serial.print("TEMPERATURA GRADOS FARENHEIT: ");
Serial.println(tempfar,2);                  // se transmite la temperatura
delay(500);
}
```

Incluyendo todas las secciones podemos obtener el siguiente código fuente (.cpp):

```
#include "WProgram.h"
#include "LM.h"
LM::LM(float pin)                           // constructor
{
pinMode(pin,INPUT);                         // define el pin como entrada
_pin = pin;
}
void LM::centigrados()
{
float tempc = 0;
tempc = analogRead(_pin);
tempc = (tempc/205.18);
tempc = tempc * 100;
Serial.print("TEMPERATURA GRADOS CENTIGRADOS: ");
Serial.println(tempc,2);
delay(500);
}
void LM::farenheit()
{
float tempfar = 0;
tempfar = analogRead(_pin);
tempfar = ((tempfar/205.18)*100);
tempfar = ((tempfar*1.8)+32);
Serial.print("TEMPERATURA GRADOS FARENHEIT: ");
Serial.println(tempfar,2);
delay(500);
}
```

Para la utilización de la librería en primer lugar se debe crear el directorio o carpeta LM dentro del subdirectorio de librerías de la carpeta Arduino en Mis Documentos. Se debe copiar o mover los archivos “LM.h” y “LM.cpp” a ese directorio y reiniciar el entorno de Arduino. Si nos dirigimos al menú Sketch/Import Library se puede observar la opción “LM”. La librería será compilada con los programas que la utilicen.

Ejemplo:

Este es un sketch que incluirá la librería que acabamos de realizar:

```
#include <LM.h>                    // se incluye la librería LM.h
float temp = A0;                   // se puede utilizar cualquier entrada analógica
                                   desde A0 hasta A15
LM lm(temp);                       // la librería realiza un reconocimiento del
                                   pin que se va a utilizar
void setup()
{
Serial.begin(9600);                // habilita el puerto serial 0 a 9600bps
}
void loop()
{
lm.centigrados();                  // se ejecutan las instancias con los
                                   valores obtenidos de la variable temp
lm.farenheit();
}
```

Hay algunas diferencias con el programa original (aparte del hecho de que parte del código se ha trasladado a la librería).

En primer lugar, hemos añadido una declaración “#include” al principio del programa. Esto permite que la librería LM esté disponible para el programa y que se envíe a la placa de Arduino. Cuando ya no sea necesaria una librería en un programa se debe eliminar la sentencia “#include” para ahorrar espacio de memoria.
En segundo lugar, creamos una instancia de la “class” LM llamada lm:

```
LM lm(temp);
```

Cuando esta línea se ejecute (esto sucede incluso antes de que se ejecute la función setup (), se llamará al constructor de la “class de LM”, y se le pasará un parámetro, en este caso, el valor del puerto analógico 0 (A0).
La función setup() solo posee la habilitación para la comunicación del puerto serial, esto se debe a que la llamada a pinMode() se produce dentro de la librería (cuando se construye la instancia).

Por último, para llamar a las funciones centigrados() y farenheit(), tenemos que precederlas del prefijo lm (que es el nombre de la instancia que hemos creado). Podríamos tener varias instancias de la “class de LM”, cada una con su propio PIN almacenado en la variable privada _pin de esa instancia. Al llamar a una función se indica que la instancia de la “class” se debe utilizar, es decir, si tuviéramos:

```
LM lm(A0); LM lm(A1);                    // el programa sólo reconocerá el puerto
                                         entrante del pin A1
```

Si probamos el nuevo programa, el IDE de Arduino no reconocerá las funciones de la librería. Por último se debe crear un archivo llamado keywords.txt en el directorio de LM.

El fichero “keywords.txt” debe contener lo siguiente:

```
LM              KEYWORD1
centigrados     KEYWORD2
farenheit       KEYWORD2
```

Cada línea tiene el nombre de la palabra clave, seguido de un tabulador (no espacios), seguido por el tipo de palabra clave. La “class” debe ser del tipo KEYWORD1 y se muestran de color naranja; las funciones deben ser del tipoKEYWORD2 y serán de color marrón al momento de escribirlas en el IDE de Arduino. Se tendrá que reiniciar el entorno Arduino para conseguir que reconozca las nuevas palabras clave.

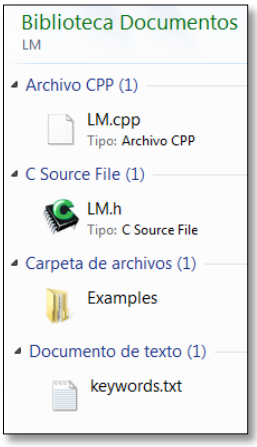

Figura 3.5: Archivos del Directorio LM. Mis Documentos/Arduino/libraries/LM
Fuente: Investigación de Campo

Es conveniente acompañar las librerías con algún programa de ejemplo que haga uso las mismas. Para ello, cree un directorio “Examples” dentro del directorio LM. A continuación, copie el directorio que contiene el programa de ejemplo que escribimos arriba (lo llamaremos LM), en el directorio de ejemplos se podrá encontrar el programa de ejemplo con la opción Sketch/Show Sketch Folder, y al reiniciar el entorno Arduino se podrá observar la opción Library-LM dentro del menú File/Sketchbook/Examples que contiene su ejemplo.

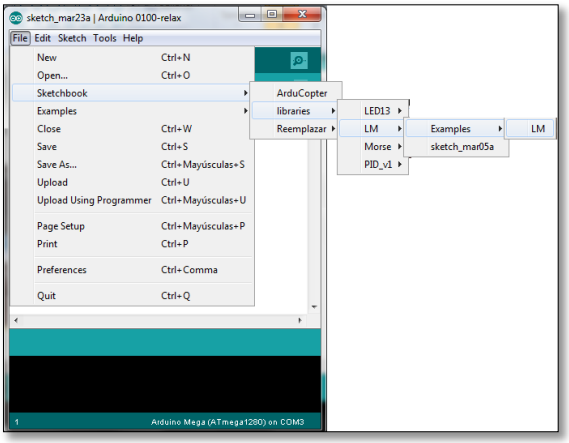

Figura 3.6: Ubicación de la Librería LM
Fuente: Investigación de Campo

3.9. LIBRERÍAS ESTÁNDAR AVR

3.9.1. AVR/IO (#include <avr/io.h>)

Descripción

Este archivo de cabecera incluye la información de reconocimiento de registro de puertos, define sus funciones como puertos de entrada o salida de datos, previamente adecuado para el dispositivo especificado por el compilador. Esta librería nunca debe ser incluida directamente. Algunos nombres de los registros comunes de todos los dispositivos AVR se definen directamente en “avr/common.h”, que se incluye en la librería “avr/io.h”.

3.9.2. AVR/ EEPROM (#include <avr/eeprom.h>)

Descripción

Este archivo de cabecera declara la interfaz para algunas rutinas de las bibliotecas simples adecuadas para el manejo de datos de la memoria EEPROM contenida en los microcontroladores AVR.

Además de las funciones de escritura, hay un conjunto de comandos de actualización. Esta función reconoce primero cada dato de entrada y omite el proceso de escritura si el valor es el mismo comparando los datos antiguos con los nuevos datos ingresados.

Para realizar las funciones de lectura/escritura primero se debe asegurar que la EEPROM está lista para ser accesada. Dado que esto puede causar largos

retrasos si una operación de escritura aún esta pendiente, por esta razón primero se debe sondear la EEPROM utilizando eeprom_is_ready() antes de intentar cualquier acción de E/S.

3.9.3. AVR/PGMSPACE (#include <avr/pgmspace.h)

Este archivo de cabecera proporciona la compatibilidad necesaria para trabajar con otros archivos de cabecera en compiladores diferentes.

3.9.4. MATH (#include <math.h>)

Este archivo de cabecera declara las constantes matemáticas y las funciones básicas.

3.10. LIBRERÍAS DEL CÓDIGO DE ARDUCOPTER

3.10.1. Fast Serial (#include <FastSerial.h>)

Esta librería fue desarrollada y enfocada a la comunicación serial entre el dispositivo ArduCopter y la PC a una velocidad de 115200bps, la comunicación serial se basa en la transmisión de datos “uno tras otro” lo que significa que se transfiere la información un bit a la vez. Arduino Mega basado en el microcontrolador Atmega 2560 posee cuatro interfaces para la comunicación serial, definiendo su utilización con otros dispositivos de la siguiente manera:

Tabla 3.4: Puertos de Comunicación Serial ArduCopter

Definición en Librería	Puerto Atmega 2560	Puerto Arduino Mega	Función
Puerto0	PE0/RXD0	Pin 1	FTDI Comunicación con el usuario
Puerto0	PE1/TXD0	Pin 0	FTDI Comunicación con el usuario
Puerto1	PD2/RXD1	Pin 19	GPS
Puerto1	PD3/TXD1	Pin 18	GPS
Puerto2	PH0/RXD2	Pin 17	ADC
Puerto2	PH1/TXD2	Pin 16	ADC
Puerto3	PJ0/RXD3	Pin 15	Telemetría
Puerto3	PJ1/TXD3	Pin 14	Telemetría

Fuente: Investigación de Campo

Sintaxis

- FastSerialPort0 (Serial);
- FastSerialPort1 (Serial1);
- FastSerialPort2 (Serial2);
- FastSerialPort3 (Serial3);

Parámetros

Velocidad de transmisión: 115200bps.

Ejemplo:

```
#include <FastSerial.h>

FastSerialPort0(Serial);
FastSerialPort1(Serial1);
FastSerialPort2(Serial2);
FastSerialPort3(Serial3);

void setup()
{
Serial.begin(115200);
Serial1.begin(115200);
Serial2.begin(115200);
Serial3.begin(115200);
}

void loop()
{
Serial.println("I.T.S.A.");
Serial1.println("PROGRAMACION ARDUINO");
Serial2.println("ELECTRONICA");
Serial3.println("2012");
delay(1000);
}
```

Método de Modificación

Siendo esta una librería modificable, se puede intercambiar la distribución de pines de comunicación serial intercambiando la definición preestablecida, la información para ello está localizada en el archivo de cabecera de la librería (FastSerial.h**)**, para modificar los puertos establecidos es necesario intercambiar la numeración de los diferentes puertos de la siguiente manera:

```
#define FastSerialPort0(_portName)   FastSerialPort(_portName, 0)   //1, 2, 3
#define FastSerialPort1(_portName)   FastSerialPort(_portName, 1)   //2, 3, 0
#define FastSerialPort2(_portName)   FastSerialPort(_portName, 2)   //3, 0, 1
#define FastSerialPort3(_portName)   FastSerialPort(_portName, 3)   //0, 1, 2
```

3.10.2. APM Radio Control (<APM_RC.h>)

Esta librería se encarga de adecuar las señales PPM emitidas por el receptor mediante el Atmega 328P y transformarlas en señales PWM, la cual posteriormente es emitida a los motores Brushless DC. ArduCopter posee 8 pines de entrada destinados a la recepción de señal PPM (modulación por posición de pulso) proveniente del receptor del radio control, y 8 pines de salida destinados a emitir la señal PWM.

Tabla 3.5: Puertos de Emisión de Señal PWM ArduCopter

Definición en Librería	Puerto Atmega 2560	Puerto Arduino Mega	Función
CH_0	PL4/OC5B	Pin 45	Salida 0 de Señal PWM
CH_1	PL5/OC5C	Pin 44	Salida 1 de Señal PWM
CH_2	PB6/OC1B	Pin 12	Salida 2 de Señal PWM
CH_3	PB7/OC1C	Pin 13	Salida 3 de Señal PWM
CH_4	PH5/OC4C	Pin 8	Salida 4 de Señal PWM
CH_5	PH4/OC4B	Pin 7	Salida 5 de Señal PWM
CH_6	PE5/OC3C	Pin 3	Salida 6 de Señal PWM
CH_7	PE4/OC3B	Pin 2	Salida 7 de Señal PWM
CH_8	PL3/OC5A	Pin 46	Salida 8 de Señal PWM
CH_9	PB5/OC1A	Pin 11	Salida 9 de Señal PWM
CH_10	PE3/OC3A	Pin 5	Salida 10 de Señal PWM

Fuente: Investigación de Campo

Tabla 3.6: Puertos Alternativos de Emisión de Señal PWM ArduCopter

Puerto Atmega 2560	Puerto Arduino Mega
PB4/OC2A	Pin 10
PH6/OC2B	Pin 9
PG5/OC0B	Pin 4

Fuente: Investigación de Campo

Tabla 3.7: Puerto de Entrada de Señal PPM ArduCopter

Puerto Atmega 2560	Puerto Arduino Mega	Función
PL0/ICP4	49	Entrada de Señal PPM
PL1/ICP5	48	Entrada de Señal PPM

Fuente: Investigación de Campo

Método de Modificación

Es recomendable no realizar cambios en la programación de los puertos destinados a la emisión de señal PWM para el control de motores ya que sólo se dispone de tres puertos alternativos para realizar el cambio. Se debe considerar

también que los puertos establecidos ya cuentan con estándares de programación que no se deben cambiar.

- Salida de Radio Control: Señal de salida para motores o servomotores (estándar 20ms)
- Rango de Señal por Canal (ch, pwm): 900-2100us (ch = 0 - 10)

3.10.3. GPS (<AP_GPS.h>)

Configura el sistema de posicionamiento global (GPS) para trabajar conjuntamente con la programación del ArduCopter, muestra datos de latitud, longitud, altitud, velocidad en tierra, tiempo, ubicación, curso y el número de satélites sincronizados con el dispositivo.

Ejemplo:

```
#include <FastSerial.h>
#include <AP_Common.h>
#include <AP_GPS.h>                    // se incluyen las librerías GPS
GPSFastSerialPort0(Serial);
FastSerialPort1(Serial1);

AP_GPS_MTK gps(&Serial1);              // habilita la velocidad de transmisión serie
#define T6 1000000
#define T7 10000000

void setup()
{
Serial.begin(38400);
Serial1.begin(38400);
stderr = stdout;
gps.print_errors = true;
Serial.println("GPS MTK");
gps.init();                            // habilita las funciones de la librería
delay(1000);
}
void loop()
{
delay(20);
gps.update();                          // se actualizan los datos emitidos por el dispositivo
                                       GPS
if (gps.new_data) {
Serial.print("gps:");
Serial.print(" Lat:");                 // se presenta la información de latitud
Serial.print((float)gps.latitude / T7, DEC);
Serial.print(" Lon:");                 // se presenta la información de longitud
Serial.print((float)gps.longitude / T7, DEC);
Serial.print(" Alt:");                 // se presenta la información de altitud
Serial.print((float)gps.altitude / 100.0, DEC);
```

```
Serial.print(" SAT:");                  // se presenta la información de los satélites
                                        sincronizados al dispositivo GPS
Serial.print(gps.num_sats, DEC);
Serial.print(" FIX:");                  // se presenta la información de localización
Serial.print(gps.fix, DEC);
Serial.print(" TIM:");                  // se presenta la información de tiempo
Serial.print(gps.time, DEC);
Serial.println();
gps.new_data = 0;                       // la información es actualizada
}
}
```

3.10.4. Comunicación I2C (<I2C.h>)

La librería I2C de ArduCopter es una versión paralela a la librería "Wire.h" que posee Arduino, ya que cumple las mismas funciones de comunicación con los dispositivos I2C, en la tarjeta electrónica de Arduino Mega el puerto definido para la entrada y salida de datos es el pin 20(SDA), y para la señal de pulso de reloj es el pin 21(SCL). Cabe mencionar que se puede acoplar hasta 127 dispositivos con protocolo de comunicación I2C conectados a los pines de reloj y de datos, y la dirección determina qué dispositivo va a responder sin causar una interferencia entre ellos.

Tabla 3.8: Dispositivos Periféricos I2C

Dispositivo I2C	Dirección (address)	Registros
Magnetómetro HMC5843	0x1E	7 bytes
Sensor Barométrico BMP085	0x77	5 bytes

Fuente: Investigación de Campo

La recepción de datos desde los dispositivos I2C hacia Arduino requiere de dos cosas:

- La dirección del dispositivo único (address) en hexadecimal.
- El número de bytes a recibir.

Sintaxis

- begin(): Inicializa la librería
- begin(address): Define la dirección del dispositivo
- requestFrom(address, count): Solicita los bytes del dispositivo esclavo
- beginTransmission(address): Habilita la comunicación entre los dispositivos

- endTransmission(): Termina la comunicación entre los dispositivos

Ejemplo:

Adquirir la señal de temperatura del sensor digital TC74 y mostrar los datos obtenidos de temperatura en grados centígrados y farenheit.

```
#include <I2C.h>

const byte temp_address = 0x4D;        // dirección(address) del sensor TCP =0x4D
                                       // la información de dirección se encuentra en el
                                       datasheet de cada elemento
const byte field = 1;                  // define los bytes de recepción

void setup()
{
Serial.begin(9600);                    // habilita el puerto de comunicación serial
Serial.println("TEMPERATURA");
Wire.begin();                          // se inicia la comunicación I2C
}
void loop()
{
Wire.beginTransmission(temp_address);      // se inicia la transmisión y habilitación del
                                           sensor I2C
Wire.send(0x00);                           // se emite un byte en cero para encerar el
                                           dispositivo
Wire.endTransmission();                    // termina la transmisión, el dispositivo se
                                           habilita y comienza a transmitir su
                                           información
Wire.requestFrom(temp_address, field);     // se solicita la información
temperatura();                             // se ejecuta la instancia temperatura()
delay(750);
}

void temperatura()
{
int temp;                              // se declaran dos variables locales para
                                       almacenar la información del sensor
int tempf;
temp= Wire.receive();                  // lee la información emitida del sensor
tempf=((temp*1.8)+32);                 // calibración para mostrar la temperatura en
                                       grados farenheit
if(temp>=191)
{
temp=temp-256;                         // calibración para emitir la temperatura en grados
                                       bajo cero
tempf=tempf-460;
Serial.print(temp );Serial.print("oC  ");Serial.print(tempf );Serial.println("oF");
}
else                                   // se transmiten los valores en grados centígrados
                                       y farenheit
{
Serial.print(temp );Serial.print("oC  ");Serial.print(tempf );Serial.println("oF");
}
}
```

3.10.5. Comunicación SPI (<SPI.h>)

La librería “SPI.h” de Arduino permite la conexión entre uno o varios dispositivos que utilicen el protocolo de comunicación SPI con Arduino, de una manera rápida a cortas distancias. Para una conexión SPI al igual que la conexión I2C siempre hay un dispositivo maestro y los dispositivos periféricos funcionaran como esclavos, en la tarjeta electrónica Arduino Mega el puerto definido para la entrada de datos es el pin 50(MISO), para la salida de datos es el pin 51(MOSI), para el pulso de reloj el pin 52(SCK), y el puerto de selección de dispositivo esclavo el pin 53(SS).

3.10.6. Memoria Externa (<DataFlash.h>)

Esta es una biblioteca destinada esencialmente para reconocimiento y uso de una memoria externa, Arduino utiliza la memoria externa para almacenar valores o datos, la memoria utilizada es la AT45DB161 la cual posee el protocolo de comunicación SPI.

Ejemplo:

Crear un programa el cual almacene diferentes datos en una memoria externa 25LC256 con protocolo de comunicación SPI y al mismo tiempo muestre los datos almacenados transmitiéndolos por el puerto serial 0.

```
#include <SPI.h>
int variable = 'a';                    // se declara una variable tipo texto
int datos, registro=0;

void setup()
{
pinMode(53, OUTPUT);                   // se declara el puerto 53 como salida
digitalWrite(53, HIGH);                // se declara el puerto 53 en estado alto
Serial.begin(9600);                    // se habilita la comunicación serial
SPI.begin();                           // se habilita la comunicación SPI

digitalWrite(53, LOW);                 // se declara el puerto 53 en estado bajo
SPI.transfer(6);                       // al declarar el estado bajo del puerto 53 se
                                       habilita la escritura de información
SPI.transfer(0);
digitalWrite(53, HIGH);                // se declara el puerto 53 en estado alto
}
void loop()
{
digitalWrite(53, LOW);                 // se declara el puerto 53 en estado bajo
SPI.transfer(6);                       // escritura habilitada
digitalWrite(53, HIGH);                // se declara el puerto 53 en estado ato
                                       // escritura deshabilitada
```

```
digitalWrite(53, LOW);                  //escritura habilitada
SPI.transfer(2);                        // escribe la variable en la sección 0 desde el
                                        registro 2

SPI.transfer(0);
SPI.transfer(registro);
SPI.transfer(variable);
digitalWrite(53, HIGH);
delay(500);

digitalWrite(53, LOW);
SPI.transfer(6);                        // escritura habilitada y guarda la variable
digitalWrite(53, HIGH);

digitalWrite(53, LOW);
SPI.transfer(3);                        // lee la variable de la sección 0 desde el registro 3
SPI.transfer(0);
SPI.transfer(registro);                 // se solicita leer el registro
datos = SPI.transfer(0x00);             // lee la variable
digitalWrite(53, HIGH);

Serial.print("direccion");              // se transmite y muestra el registro y la variable
                                        guardada

Serial.print(registro);
Serial.print(" : ");
Serial.println(datos, BYTE);            // muestra la información en formato texto
registro++;                             // la variable y el registro se incrementan, de
                                        esta manera se guardan los datos en un registro
                                        diferente

variable++;
digitalWrite(53, LOW);                  // lectura deshabilitada
SPI.transfer(4);
digitalWrite(53, HIGH);
}
```

3.10.7. Conversor Analógico Digital (<AP_ADC.h>)

Un conversor analógico/digital es un dispositivo electrónico capaz de convertir una señal analógica en un valor binario, en decir, se encarga de transformar una o varias señales analógicas a digitales (0 y 1). El dispositivo establece una relación entre su señal de entrada (señal analógica) y su salida (resultante digital) dependiendo de su resolución, la resolución determina la precisión con la que se reproduce la señal original. El objetivo de utilizar un conversor analógico/digital en el sistema ArduCopter se debe a la implementación del acelerómetro y un giróscopo, ambos analógicos, la señal analógica de estos sensores es adquirida y transformada a formato digital, el dispositivo para realizar esta función es el conversor analógico/digital ADS 7844 con protocolo de comunicación SPI.

Tabla 3.9: Puertos de Conexión ArduCopter - ADS 7844

Puerto Atmega 2560	Puerto Arduino Mega	Función
PH0	Pin 12/RXD2	MOSI/Entrada de datos
PH1	Pin 13/TXD2	MISO/Salida de datos
PH2	Pin 14	SCK/Pulso de reloj
PC4	Pin 33	SS/CS/Selector

Fuente: Investigación de Campo

3.10.8. Sensor Barométrico BMP085 (<AP_Baro.h>)

El sensor barométrico digital tiene como principales funciones adquirir datos de presión atmosférica, temperatura, y altitud. Este sensor posee el protocolo de comunicación I2C de esta manera en la tarjeta electrónica de Arduino Mega el puerto definido para la entrada y salida de datos es el pin 20(SDA), y para la señal de pulso de reloj se utiliza el pin 21(SCL).

Ejemplo:

Adquirir la señal de temperatura, presión y altitud del sensor barométrico digital BMP085.

```
#include <Wire.h>                       // cumple las mismas funciones que la librería I2C de ArduCopter
#include <Adafruit_BMP085.h>            // la librería “Adafruit_BMP085.h” ya incluye la dirección del dispositivo para comunicarse con Arduino

Adafruit_BMP085 bmp;                    // se declara la variable “bmp”
void setup()
{
Serial.begin(9600);                     // se habilita la transmisión serial
bmp.begin();                            // se habilita la librería del sensor
}

void loop()
{
Serial.print("TEMPERATURA = ");          // se transmite TEMPERATURA
Serial.print(bmp.readTemperature());    // se transmiten los datos de temperatura
Serial.println(" oC");

Serial.print("PRESION = ");             // se transmite PRESION
Serial.print(bmp.readPressure());       // se transmiten los datos de presión
Serial.println(" Pa");

Serial.print("ALTITUD = ");             // se transmite ALTITUD
Serial.print(bmp.readAltitude());       // se transmiten los datos de altitud sin calibración
Serial.println(" mtrs");

Serial.print("ALTITUD REAL = ");        // se transmite ALTITUD REAL
Serial.print(bmp.readAltitude(101500)); // se transmiten los datos de altitud calibrados
Serial.println(" mtrs");
```

```
Serial.println();
delay(500);
}
```

3.10.9. Magnetómetro HMC5843 (<AP_Compass.h>)

El sensor HMC5843 tiene como principal función adquirir la señal variable del campo magnético terrestre, esta información es procesada y mostrada simulando el movimiento sobre los tres ejes principales (X, Y, Z), también se puede mostrar la información recolectada simulando una brújula normal que muestra los cuatro puntos cardinales (Norte, Sur, Este y Oeste), este sensor posee el protocolo de comunicación I2C de esta manera en la tarjeta electrónica de Arduino Mega el puerto definido para la entrada y salida de datos es el pin 20(SDA), y para la señal de pulso de reloj se utiliza el pin 21(SCL).

Ejemplo:

Adquirir la señal variable del campo magnético terrestre y mostrar sus valores en base a los ejes principales de movimiento (X, Y, Z) utilizando el magnetómetro HMC5843.

```
#include <HMC.h>                    // la librería “HMC.h” ya incluye la dirección del
                                    dispositivo para comunicarse con Arduino
void setup()
{
Serial.begin(9600);
HMC.init();                         // inicia la lectura de información
}
void loop()
{
int x,y,z;                          // se declara las 3 variables que contienen los
valores de los tres ejes de movimiento con respecto al norte geográfico
delay(100);
HMC.getValues(&x,&y,&z);            // adquisición de los registros(bytes) que emite el
                                    dispositivo
Serial.print("x: ");                // muestra el valor del eje x
Serial.print(x);
Serial.print(" y: ");               // muestra el valor del eje y
Serial.print(y);
Serial.print(" z: ");               // muestra el valor del eje z
Serial.println(z);
}
```

3.10.10. Control Integral Derivativo (<AC_PID.h>)

Un controlador PID es un mecanismo de control por realimentación que calcula la desviación o error entre un valor medido de entrada (input) y el valor que se quiere obtener (setpoint), para aplicar una acción correctora sobre el valor de salida (output), que ajuste el proceso.

Sintaxis

PID(Input, Output, Setpoint, Kp, Ki, Kd): Habilita las funciones de la librería PID

- Input: Es la variable que se desea controlar
- Output: Es la variable resultante ajustada por el control PID
- Setpoint: Es el valor deseado de entrada que se desea mantener estable
- Kp, Ki, Kd: Parámetros de ajuste, determinan el cambio que se produce al valor de salida

Compute(): Contiene el algoritmo de control PID

SetMode(): Configura el control PID en modo manual (MANUAL) o automático (AUTOMATIC)

SetOutputLimits(x, y): Configura el rango o ventana de trabajo de un control PID

SetTunings(Kp, Ki, Kd): Define las características del control PID

- Kp: Determina la reacción en el control PID al valor de error actual
- Ki: Determina la reacción en el control PID al valor de error en un determinado tiempo
- Kd: Determina la reacción en el control PID a la variación del valor de error

SetSampleTime(): Determina la frecuencia de trabajo del control PID, el valor estándar es 200ms

SetControllerDirection(): Determina si el control PID funciona directamente proporcional (DIRECT) o inversamente proporcional (REVERSE)

Ejemplo:

Realizar un sketch el cual realizara la adquisición de señal del puerto analógico 0 y de esta manera controlar la señal PWM de salida por el puerto digital 6.

```
#include <PID_v1.h>                    // librería PID de arduino
#define pwm 6                          // se define el puerto 6 como salida
```

```
double Setpoint, Input, Output;                    // se definen las variables de control PID

                                                   // se especifica los parámetros iniciales de
                                                   ajuste en forma directamente proporcional
PID myPID(&Input, &Output, &Setpoint,0,5,1, DIRECT);

void setup()
{
Serial.begin(9600);                                // se habilita la comunicación serial
Serial.print("INPUT");Serial.print("OUTPUT");Serial.println("SETPOINT");
Input = analogRead(0);                             // la variable “Input” adquiere los valores
                                                   obtenidos del puerto analógico 0
Setpoint = 125;                                    // se establece el valor deseado
myPID.SetOutputLimits(1, 250);                     // se establece el rango de trabajo o
                                                   ventana del control PID
myPID.SetMode(AUTOMATIC);                          // se habilitan las funciones del control PID
                                                   en modo de cálculo automático
}
void loop()
{                                                  // se transmiten los valores de entrada,
                                                   salida y valor deseado
Serial.print(Input,0);Serial.print(Output,0);Serial.println(Setpoint,0);
Input = analogRead(0);
myPID.Compute();                                   // se realiza el cálculo PID
analogWrite(pwm,Output);                           // se emiten los valores calculados a través
                                                   del puerto 6
delay(250);
}
```

3.10.11. Sensores Inerciales (<AP_InertialSensor.h>)

Esta librería se encarga de acondicionar las señales que entregan los sensores inerciales, estos sensores son los giróscopos y el acelerómetro, cada uno provee tres señales las cuales deben ser calibradas teniendo en cuenta las características de cada dispositivo.

Tabla 3.10: Sensores Inerciales

Características de Resolución	
Dispositivo	**Resolución**
Acelerómetro ADXL335	330mV/º
Giróscopo IDG500	2 mV/º
Giróscopo ISZ500	2 mV/º

Fuente: Investigación de Campo

Ejemplo:

Realizar un sketch el cual adquiera las señales analógicas variables emitidas desde los sensores (acelerómetro y giróscopo), y mostrar sus valores en base a los ejes principales de movimiento (X, Y, Z).

```
int gyrox;                                  // variable x giróscopo
int gyroy;                                  // variable y giróscopo
int gyroz;                                  // variable z giróscopo
int accelx;                                 // variable x acelerómetro
int accely;                                 // variable y acelerómetro
int accelz;                                 // variable z acelerómetro
float gx,gy,gz;                             // subvariables x, y, z del giróscopo
float ax,ay,az;                             // subvariables x, y, z del acelerómetro

void setup()
{
Serial.begin(9600);                         // se habilita la comunicación serial
Serial.print("ACELEROMETRO");Serial.print(" ");Serial.println("GIROSCOPO");
}
void loop()
{
gyrox=analogRead(0);                        // la variable “gyrox” adquiere los valores
                                            obtenidos del puerto analógico 0

gyroy=analogRead(1);                        // la variable “gyroy” adquiere los valores
                                            obtenidos del puerto analógico 1

gyroz=analogRead(2);                        // la variable “gyroz” adquiere los valores
                                            obtenidos del puerto analógico 2

accelx=analogRead(3);                       // la variable “accelx” adquiere los valores
                                            obtenidos del puerto analógico 3

accely=analogRead(4);                       // la variable “accely” adquiere los valores
                                            obtenidos del puerto analógico 4

accelz=analogRead(5);                       // la variable “accelz” adquiere los valores
                                            obtenidos del puerto analógico 5


gx=((gyrox*0.01745329252)*0.4);             // la subvariable “gx” acondiciona los datos
                                            obtenidos de la variable “gyrox”, de esta
                                            manera representar sus valores finales
                                            en radianes (pi/180)
gy=((gyroy*0.01745329252)*0.41);
gz=((gyroz*0.01745329252)*0.41);
ax=((9.80665/423.8)*accelx);                // la subvariable “ax” acondiciona los datos
                                            obtenidos de la variable “accelx”, d esta
                                            manera representa la escala del
                                            acelerómetro
ay=((9.80665/423.8)*accely);
az=((9.80665/423.8)*accelz);

Serial.print("X=");Serial.print(gx,2); Serial.print("X=");Serial.println(ax,2);
Serial.print("Y=");Serial.print(gy,2); Serial.print("Y=");Serial.println(ay,2);
Serial.print("Z=");Serial.print(gz,2); Serial.print("Z=");Serial.println(az,2);
Serial.println();                           // se transmiten los valores finales
delay(500);
```

}

3.10.12. Configuraciones ("defines.h",”config.h","config_channels.h")

Este conjunto de librerías definen y configuran los puertos utilizados por el sistema de navegación del ArduCopter, además de otras funciones.

Configuración ("defines.h"):

Tabla 3.11: Puertos de Expansión

Puerto Arduino Mega	Función Alternativa
AN0 54	Divisor de voltaje
AN1 55	Divisor de voltaje
AN2 56	Divisor de voltaje
AN3 57	Divisor de voltaje
AN4 58	Entrada analógica
AN5 59	Entrada analógica
AN6 60	Entrada analógica, Puerto de expansión
AN7 61	Entrada analógica, Puerto de expansión
AN8 62, AN9 63, AN10 64, AN11 65	No conectado/NC
AN12 66, N13 67, AN14 68, AN15 69	No conectado/NC

Fuente: Investigación de Campo

Configuración ("config.h"):

Tabla 3.12: Configuración de Puertos - APM1

Definición en Librería	Puerto Atmega 2560	Puerto Arduino Mega	Función
A_LED_PIN	PC0	Pin 37	Led A
B_LED_PIN	PC1	Pin 36	Led B
C_LED_PIN	PC2	Pin 35	Led C
SLIDE_SWITCH_PIN	PG1	Pin 40	CLI Switch
PUSHBUTTON_PIN	PG0	Pin 41	Reset
OPTFLOW_CS_PIN	PC3	Pin 34	Sensor óptico
BATTERY_PIN_1	PF0	Pin 0	Medidor de voltaje de batería
CURRENT_PIN_1	PF1	Pin 1	Medidor de corriente de batería

Fuente: Investigación de Campo

Configuración ("config_channels.h"):

Tabla 3.13: Asignación de Canales para los Motores - APM1

Motor	Definición de Canal
MOT_1	CH_1
MOT_2	CH_2
MOT_3	CH_3
MOT_4	CH_4
MOT_5	CH_7
MOT_6	CH_8

MOT_7	CH_10
MOT_8	CH_11

Fuente: Investigación de Campo

3.11. COMPILADOR ARDUINO RELAX

El software compilador Arduino Relax es una versión paralela del compilador Arduino normal destinada a trabajar únicamente con los dispositivos de ArduCopter o el Atmega 2560, esta versión tiene como principal característica ser una versión más ligera que la original, esto significa que omite ciertas líneas de código estándar las cuales generan errores de compilación en el software de ArduCopter. Este software se lo puede adquirir entrando en la página del proyecto www.diydrones.com posteriormente siguiendo la sección ArduCopter.

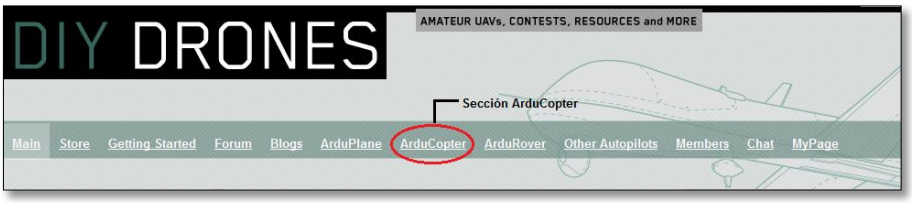

Figura 3.7: Página Principal De Diydrones
Fuente: www.diydrones.com

En la sección de ArduCopter se procede a seguir la sección “Downloads” (descargas), aquí podremos encontrar el software compilador y la actualización más reciente del código de ArduCopter.

Figura 3.8: Página Principal de ArduCopter
Fuente: http://code.google.com/p/arducopter/wiki/ArduCopter

Al encontrar el ítem correspondiente al software de Arduino Relax procedemos a seleccionarlo, esto nos conducirá finalmente a iniciar nuestra descarga.

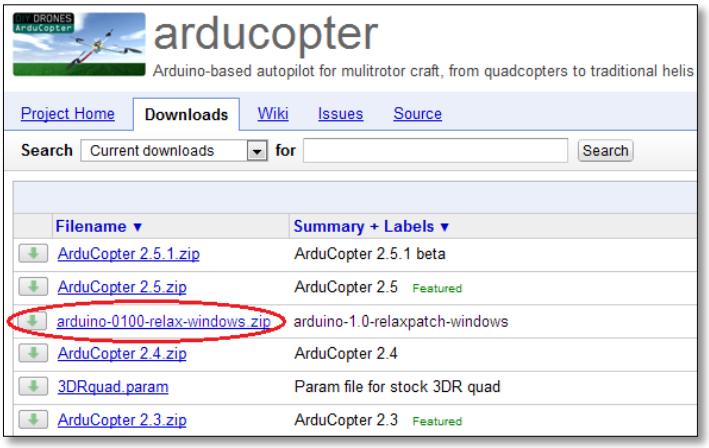

Figura 3.9: Página Principal de ArduCopter/Descargas
Fuente: http://code.google.com/p/arducopter/downloads/list

Finalmente se procede a seleccionar el ítem “Arduino-0100-relax-windows.zip” para iniciar la descarga correspondiente, las características de guardado al finalizar la descarga dependerán del navegador y del sistema operativo utilizado, por defecto el Internet Explorer incluido en el OS de Windows 7 guardará los archivos descargados en la carpeta de “Descargas”.

Figura 3.10: Página Principal de ArduCopter/Descargas/Arduino Relax
Fuente: http://code.google.com/p/ardupilot-mega/downloads/detail?name=arduino-0100-relax-windows.zip&can=2&q=

Para adquirir la última versión del software de control de ArduCopter se deben seguir los mimos pasos. Una de las principales ventajas que posee el software de compilación Arduino y Arduino Relax es el hecho de que no es necesario instalar sus funciones en la raíz del OS de nuestro ordenador, simplemente es necesario, crear una carpeta denominada Arduino ubicada específicamente dentro de la carpeta de “Mis Documentos”.

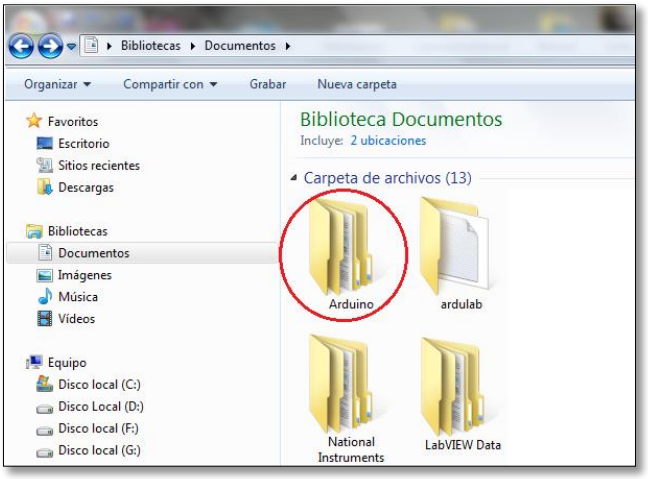

Figura 3.11: Creación de Carpeta Arduino
Fuente: Investigación de Campo

Posteriormente se procede a mover los archivos incluidos en la carpeta de ArduCopter (ArduCopter y libraries) en la carpeta denominada “Arduino” creada anteriormente.

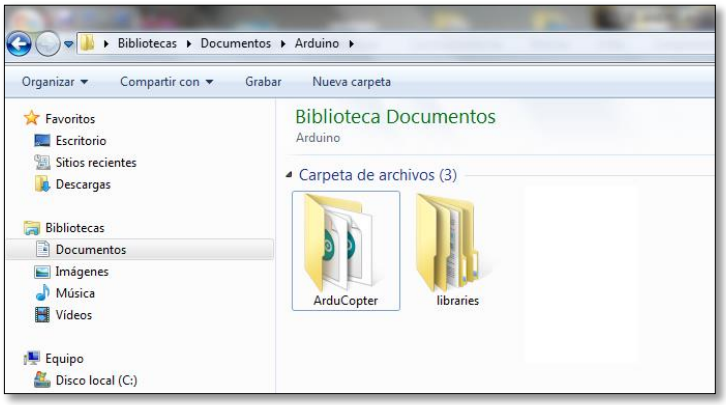

Figura 3.12: Carpetas ArduCopter - libraries
Fuente: Investigación de Campo

El software compilador Arduino Relax puede ser ejecutado desde cualquier localización en nuestro equipo, de esta manera al inicializar nuestro software compilador dentro de la sección File/Sketchbook podremos visualizar que el software reconocerá automáticamente el código de ArduCopter.

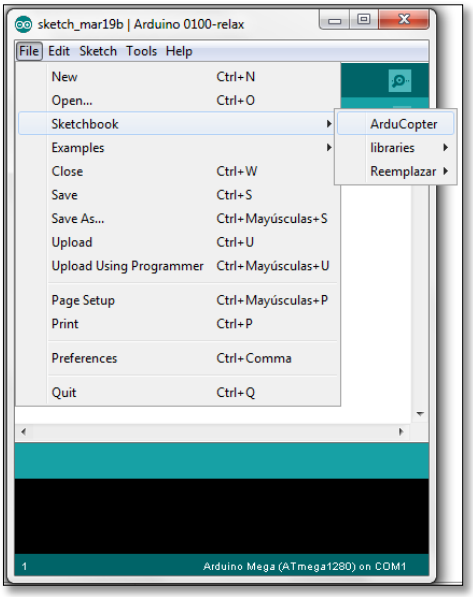

Figura 3.13: Reconocimiento de Código ArduCopter
Fuente: Investigación de Campo

Al acceder al código de ArduCopter podremos observar el código principal, para determinar que el código mostrado es correcto y no presenta errores es necesario realizar una verificación o compilación del código, previamente se realiza la selección de la tarjeta electrónica en la cual estamos trabajando en el menú “Tools”(herramientas) opción “Boards”, en nuestro caso se debe seleccionar la opción Arduino Mega 2560.

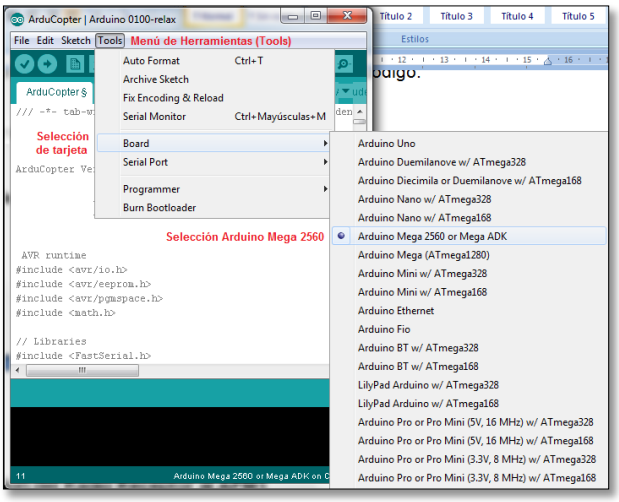

Figura 3.14: Procedimiento de Selección de Tarjeta Arduino Mega 2560
Fuente: Investigación de Campo

Al finalizar nuestra selección se procede a ejecutar la opción “Verify” (verificación o compilación) para verificar el buen funcionamiento del código sin errores.

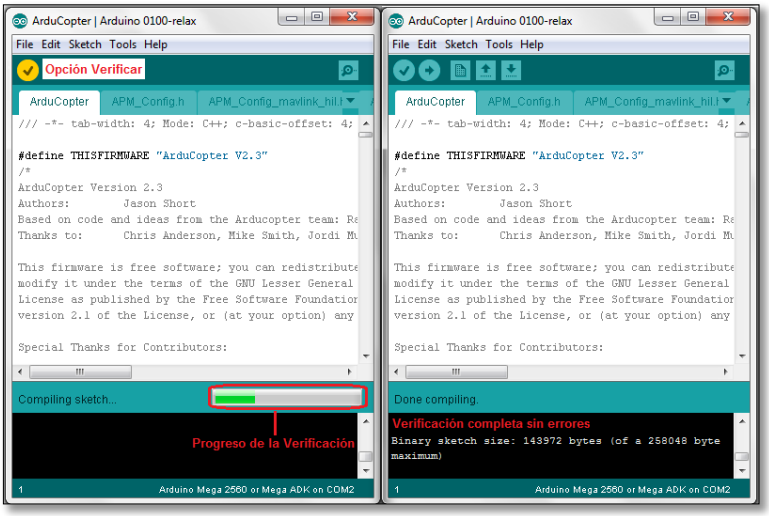

Figura 3.15: Verificación de Código ArduCopter
Fuente: Investigación de Campo

3.12. SINCRONIZACIÓN DE COMUNICACIÓN APM1 - PC

El procedimiento descrito a continuación debe ser realizado cada vez que conectemos nuestro dispositivo APM1 o un dispositivo de la familia Arduino a un nuevo ordenador (PC), esto es necesario ya que el dispositivo APM1 posee un elemento electrónico FTDI para convertir la señal de comunicación serial a comunicación USB, los controladores de este elemento deben ser instalados para obtener un correcto reconocimiento de nuestro dispositivo por parte del ordenador y no presentar fallas con la transferencia del código de ArduCopter al dispositivo APM1.

Figura 3.16: Conexión USB del Dispositivo APM1
Fuente: Investigación de Campo

Al realizar la conexión nuestro ordenador inmediatamente mostrara una ventana informativa indicando que no se pudo instalar el software controlador de nuestro dispositivo, por este motivo se procede a realizar la instalación en forma manual realizando el siguiente procedimiento:

1. Acceder al menú Inicio/Panel de Control/Administrador de Dispositivos, aquí podremos observar que nuestro dispositivo no ha sido reconocido por nuestro ordenador siendo catalogado como Dispositivo desconocido.
2. Seleccionar el dispositivo ejecutando click derecho sobre el dispositivo, aparecerá inmediatamente un cuadro de opciones, seleccionar la opción “Actualizar software de controlador”.

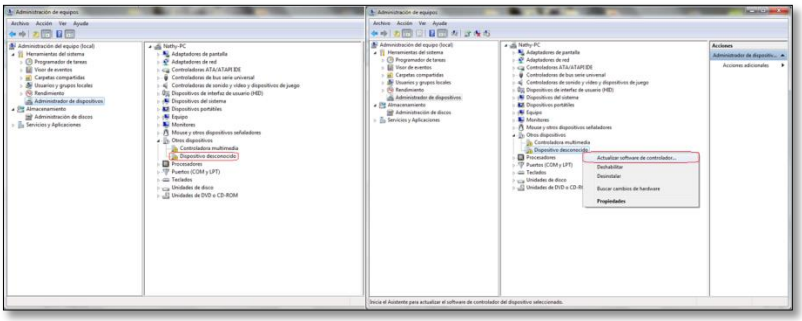

Figura 3.17: Comunicación APM1 - Ordenador Parte 1
Fuente: Investigación de Campo

3. Al seleccionar esta opción aparecerá una ventana informativa preguntando al usuario como desea actualizar el software del controlador, seleccionar la opción de búsqueda del software de controlador en el equipo y posteriormente seleccionar la opción “Examinar”.

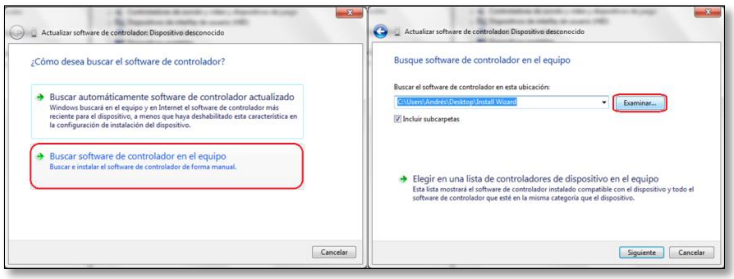

Figura 3.18: Comunicación APM1 - Ordenador Parte 2
Fuente: Investigación de Campo

4. Los controladores FTDI pueden ser localizados directamente al abrir el directorio que contiene el compilador de Arduino o Arduino Relax, al abrir este directorio localizaremos el subdirectorio denominado drivers/FTDI USB DRIVERS, se debe seleccionar la opción “drivers” y posteriormente seleccionar la opción “Siguiente”.

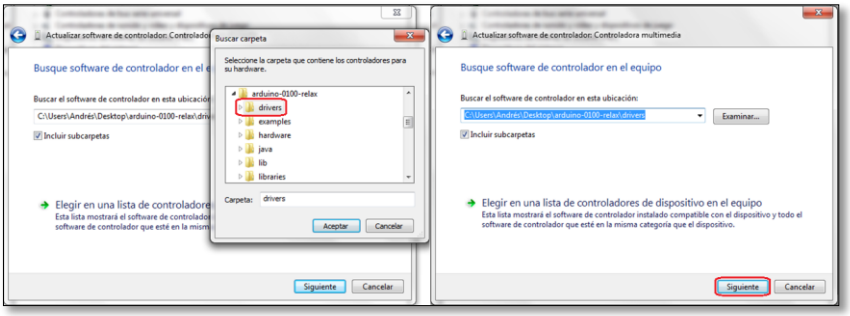

Figura 3.19: Comunicación APM1 - Ordenador Parte 3
Fuente: Investigación de Campo

5. Los controladores procederán a ser instalados, es posible que el software de seguridad del ordenador bloquee parcialmente la instalación, para habilitar la instalación se debe indicar que es una instalación segura para el ordenador, posteriormente los controladores serán instalados correctamente.

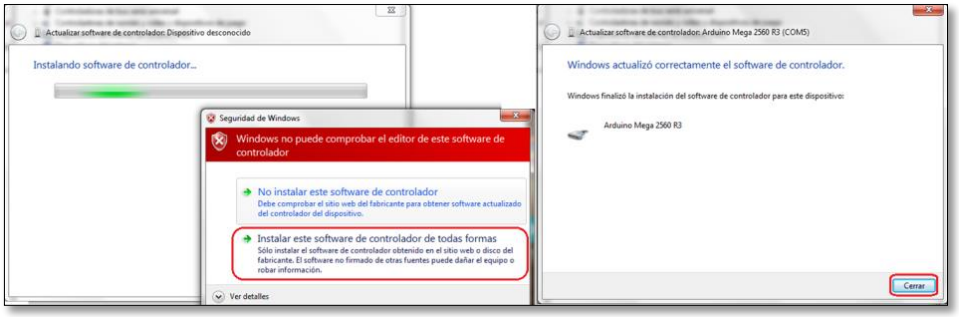

Figura 3.20: Comunicación APM1 - Ordenador Parte 4
Fuente: Investigación de Campo

3.13. TRANSFERENCIA DEL CÓDIGO DE ARDUCOPTER AL APM1 MEGA 2560

Al finalizar la verificación del código de ArduCopter sin errores y realizando la sincronización de comunicación del APM1 - PC se procede a trasferir el código de ArduCopter en la tarjeta electrónica APM1 seleccionando la opción de subir (Upload), al finalizar la carga sin errores podremos observar el mensaje de carga completa (Done Uploading).

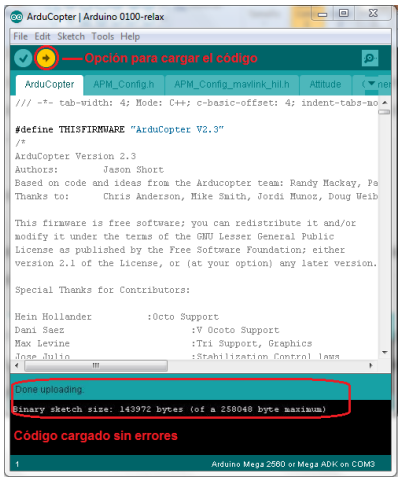

Figura 3.21: Transferencia del Código de ArduCopter al APM1
Fuente: Investigación de Campo

3.14. CONEXIÓN DEL RADIO RECEPTOR AL APM1 MEGA 2560

Para iniciar la conexión del radio receptor por motivos de seguridad primero es necesario desconectar los cuatro cables de transmisión de señal PWM.

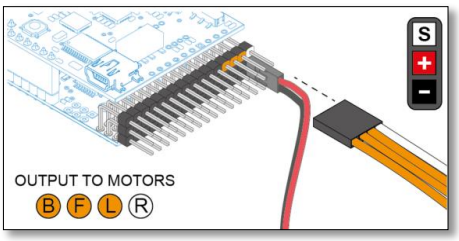

Figura 3.22: Cables de Transmisión de señal PWM
Fuente: http://code.google.com/p/arducopter/wiki/AC2_Radio

Una vez desconectados estos cables se procede a la conexión de los siete cables de RC del receptor a los puertos de conexión del APM1 Mega 2560, de igual manera se conectan los cables de alimentación del receptor en las líneas de distribución de energía del APM1 Mega 2560.

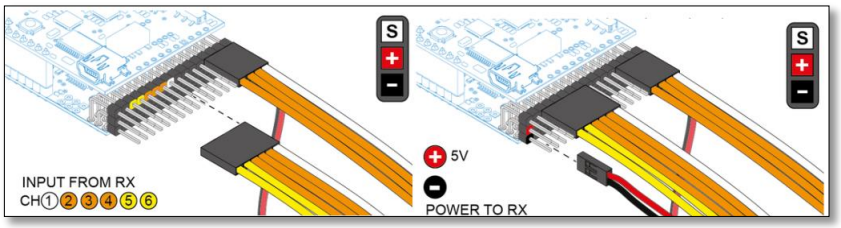

Figura 3.23: Conexión de Cables del Receptor
Fuente: http://code.google.com/p/arducopter/wiki/AC2_Radio

Los cables de alimentación y los cables de RC del receptor deben ser conectados en el respectivo canal siguiendo las guías numéricas que muestra el mismo dispositivo.

Foto 2: Receptor Futaba R168DF
Fuente: Investigación de Campo

De esta manera el dispositivo RC receptor está conectado correctamente a los puertos de recepción del AMP1.

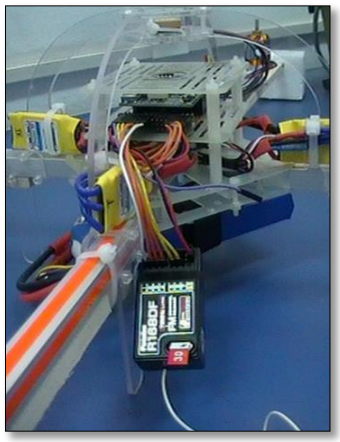

Foto 3: APM1 - Dispositivo Receptor
Fuente: Investigación de Campo

3.15. PRIMERA CONFIGURACIÓN Y CALIBRACIÓN DE LOS SISTEMAS DE NAVEGACIÓN

3.15.1. Configuración y Calibración por Arduino Relax

Al mantener la comunicación entre el APM1 y la PC podemos acceder y visualizar el código instalado utilizando la aplicación de Monitor Serial que incluye el software compilador Arduino Relax.

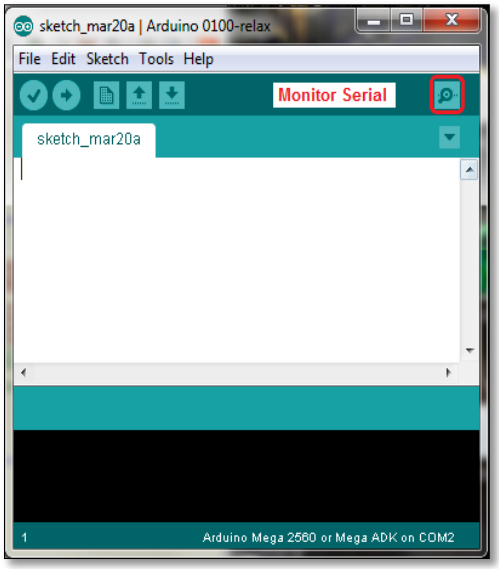

Figura 3.24: Monitor Serial/Arduino Relax
Fuente: Investigación de Campo

Al acceder al Monitor Serial podremos observar la versión del código de ArduCopter instalada anteriormente, de igual manera ya es posible acceder al menú de configuración y calibración presionando tres veces seguidas la tecla "Enter" de nuestro teclado como indica el mismo código.

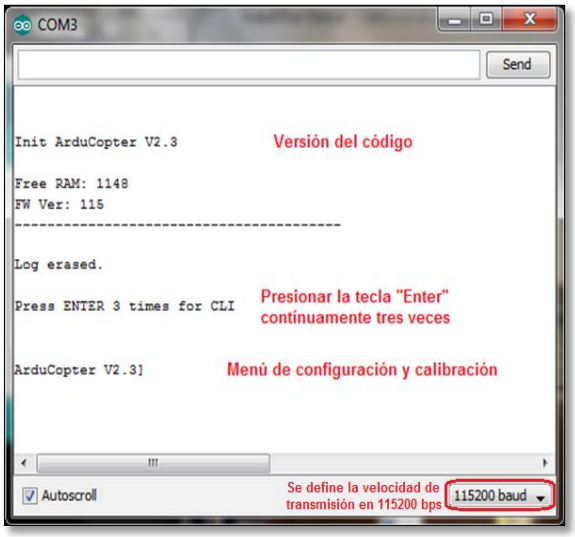

Figura 3.25: Monitor Serial
Fuente: Investigación de Campo

Para acceder a las opciones de configuración y calibración del código de ArduCopter es necesario introducir la palabra "setup" (configuración), posteriormente para poder visualizar las opciones de configuración es necesario introducir la palabra "help", de esta manera podremos observar las distintas opciones de configuración.

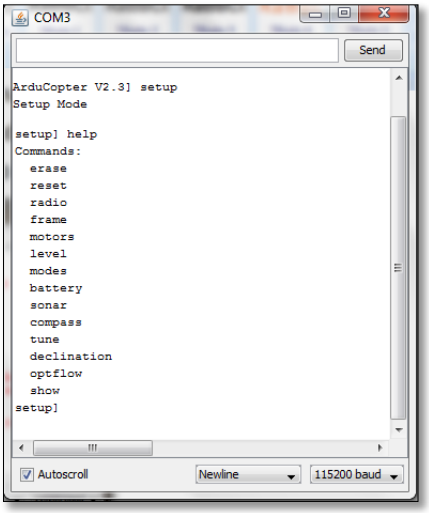

Figura 3.26: Menú de Configuraciones ArduCopter
Fuente: Investigación de Campo

Cada opción representa una configuración del código de ArduCopter, de igual manera se puede realizar una calibración de los dispositivos periféricos acoplados a la APM1.

Descripción de Comandos:

erase: Esta opción borra toda información de configuración y calibración preestablecida anteriormente dejando la memoria libre para guardar nuevos datos.

reset: Reinicia todas las funciones y las reestablece.

radio: Guarda la información de los límites máximos y mínimos de todos los canales de radio, el límite debe ser establecido entre 1100 y 1900.

frame: Establece la posición de vuelo del ArduCopter (x o +), en nuestro caso debe ser establecido en la configuración de cruz (+).

motors: Configura el correcto funcionamiento de todos los motores.

level: Configura los valores del acelerómetro, el ArduCopter debe estar sobre una superficie plana para realizar esta calibración.

modes: Establece los modos de vuelo.

battery: Almacena la información de las características de la batería.

sonar: Configura los valores del sensor periférico sonar.

compass: Esta opción habilita o deshabilita la brújula digital.

declination: Guarda la información de declinación con respecto al meridiano de Greenwich en grados.

show: Muestra todos los valores de configuración y calibración.

La opción “level” y la opción “radio” son fundamentales de configurar y calibrar antes de continuar con la utilización de nuestro dispositivo ArduCopter. La opción “level” se encargará de almacenar los datos referentes a la nivelación física de nuestro dispositivo, para realizar esta calibración es necesario que nuestro dispositivo se encuentre en un lugar plano sin ningún tipo de obstrucciones.

Para realizar la calibración de nivel (level) de nuestro dispositivo es necesario ingresar al menú de configuraciones (setup) siguiendo los pasos anteriormente vistos, al encontrarnos en el menú de configuraciones se debe ingresar la palabra “level” y presionando la tecla “Enter” automáticamente se guardarán los valores de nivel obtenidos.

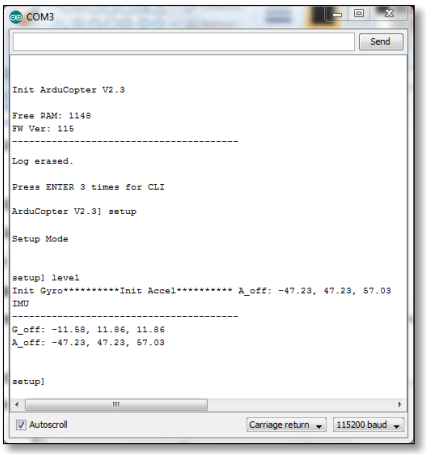

Figura 3.27: Calibración de Nivel
Fuente: Investigación de Campo

Para realizar la calibración de la radio es necesario ingresar al menú de configuraciones (setup) siguiendo los pasos anteriormente vistos, al encontrarnos en el menú de configuraciones se debe ingresar la palabra “radio”, en este momento el software se dispone a guardar los valores obtenidos desde el radio control, para esto es necesario mover todos los controles en sus rangos máximos y mínimos, después de realizar los respectivos movimientos se debe presionar la tecla “Enter” y automáticamente se guardarán los valores de nivel obtenidos del radio control.

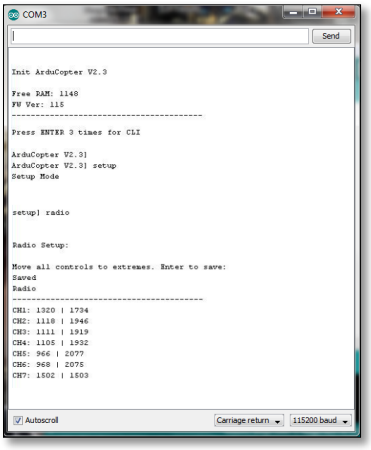

Figura 3.28: Calibración de Radio
Fuente: Investigación de Campo

Los valores obtenidos no deben ser mayores a 2100 ni menores a 1100 para cada canal.

También se puede utilizar la opción “test” al finalizar las configuraciones y calibraciones iniciales, esta opción ayuda al usuario a comprobar el correcto funcionamiento de los dispositivos y otras funciones que conforman el sistema de navegación del ArduCopter, para poder conocer las opciones de comprobación es necesario introducir la palabra “help”, de esta manera podremos observar las distintas opciones que posee.

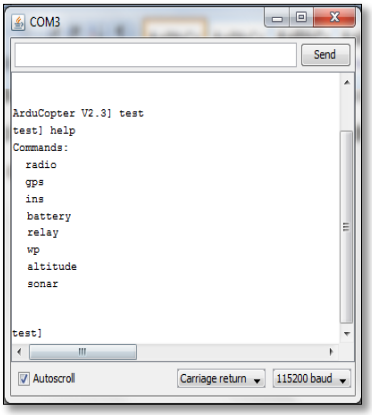

Figura 3.29: Menú Test ArduCopter
Fuente: Investigación de Campo

Cada opción nos mostrará el correcto funcionamiento de los dispositivos acoplados al sistema de ArduCopter.

Descripción de Comprobación:

radio: Muestra los valores de cada canal emitidos por el radio transmisor.

gps: Muestra la información de posicionamiento global (GPS), el sistema tarda un tiempo estimado de dos minutos en obtener los datos necesarios.

ins: Muestra la información del sensor inercial, en nuestro caso muestra la información del acelerómetro ADXL330.

battery: Muestra la información del nivel de voltaje actual que posee la batería.

wp: Muestra los datos almacenado de los puntos de referencia (waypoint).

altitude: Muestra la información del sensor barométrico, en nuestro caso muestra la información del sensor BMP085.

compass: Muestra la información de brújula digital en sus respectivos ejes (N,S,E,O).

3.15.2. Configuración y Calibración por APM Planner

Para realizar la configuración mediante el APM Planner es necesario mover el selector CLI en modo de vuelo, es decir, lejos de los puertos de RC, esto permitirá que nuestro dispositivo ArduCopter guarde las configuraciones y calibraciones realizadas, el APM Planner también nos permite realizar la programación para efectuar un vuelo programado.

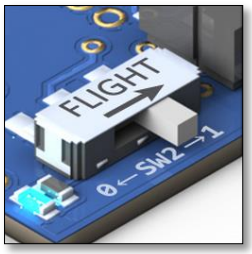

Figura 3.30: Selector CLI
Fuente: Investigación de Campo

Posteriormente procedemos a seleccionar la opción “Firmware” para iniciar con las configuraciones y calibraciones de radio, además de configurar y habilitar otras opciones de nuestro dispositivo ArduCopter.

Figura 3.31: APM Panner
Fuente: Investigación de Campo

En la opción de “Firmware” es necesario indicarle al software que se realizarán las configuraciones en un dispositivo ArduCopter y también es necesario sincronizar el software APM Planner con el dispositivo ArduCopter para esto modificamos las opciones mostradas en la parte superior derecha, también se debe configurar el puerto de conexión (el puerto de conexión variará al utilizar un ordenador diferente) y la velocidad de transmisión a 115200bps, al terminar estas configuraciones se procede a ejecutar la opción de conectar (Connect).

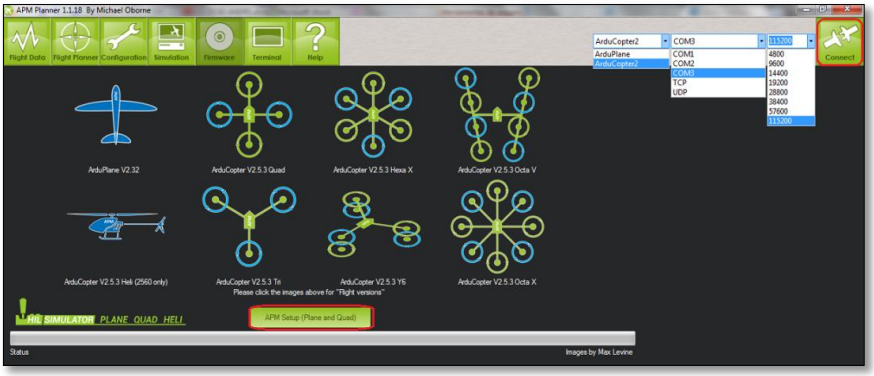

Figura 3.32: APM Planner - Configuración de Comunicación
Fuente: Investigación de Campo

Al establecerse la conexión seleccionamos la opción “APM Setup”, al realizar esta acción inmediatamente podremos observar la ventana de configuraciones iniciando con la configuración de la radio.

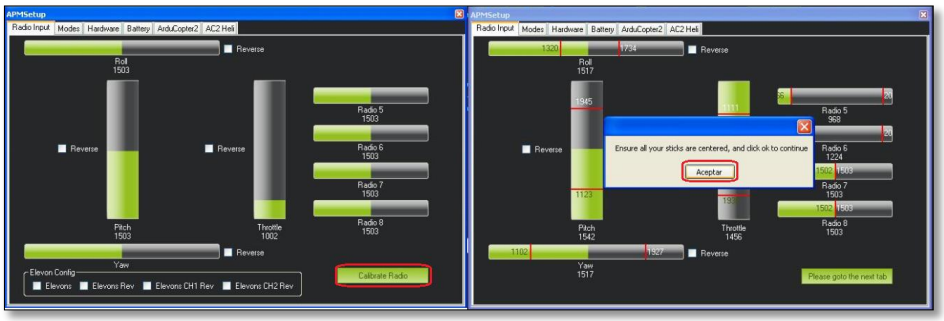

Figura 3.33: APM Planner - APM Setup
Fuente: Investigación de Campo

Seleccionando la opción “Calibrate Radio” podremos calibrar y guardar los valores de los canales de radio, para esto es necesario mover todos los controles en sus rangos máximos y mínimos, para guardar estos valores debemos seleccionar la opción “Click when Done”, se debe recordar que los valores obtenidos no deben ser mayores a 2100 ni menores a 1100 para cada canal.

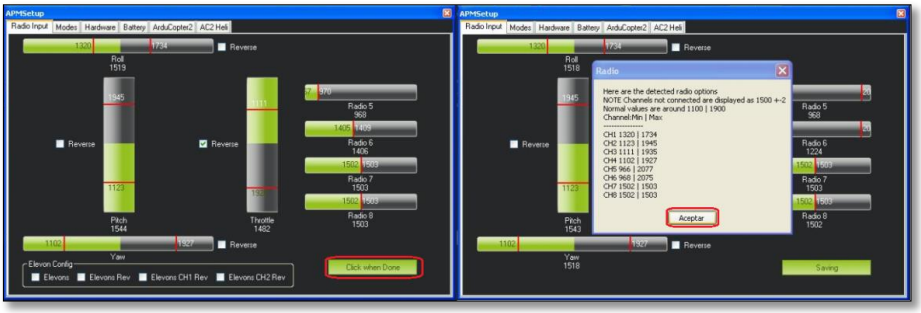

Figura 3.34: APM Planner – Configuración de Radio Control
Fuente: Investigación de Campo

La siguiente configuración muestra los modos de vuelo que posee nuestro dispositivo, por defecto esta seleccionado el modo estable ya que es el modo mas simple de volar en el dispositivo ArduCopter, posteriormente se detallará las características de cada modo de vuelo.

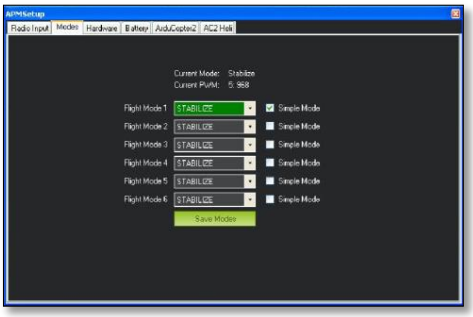

Figura 3.35: APM Planner – Configuración de Modo de Vuelo
Fuente: Investigación de Campo

La siguiente configuración muestra los dispositivos periféricos acoplados a nuestro ArduCopter, estos dispositivos nos permitirán obtener un mayor control de vuelo con el dispositivo sonar y el sensor de velocidad de aire, además de la adquisición de otro tipo de información como por ejemplo la brújula digital y el sensor de flujo óptico, mediante esta ventana de configuración podremos habilitar o deshabilitar los dispositivos.

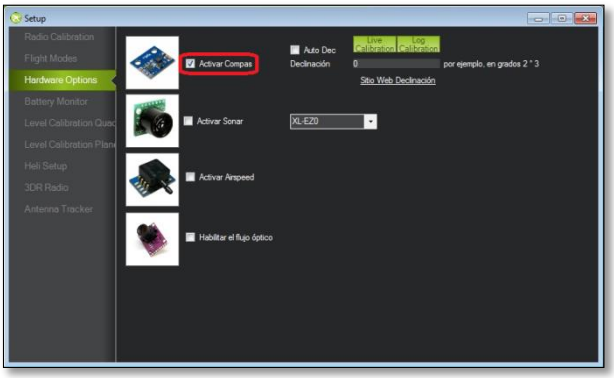

Figura 3.36: APM Planner – Configuración de Dispositivos Periféricos
Fuente: Investigación de Campo

Para finalizar en la opción “ArduCopter2” debemos seleccionar la configuración de marco de vuelo del dispositivo ArduCopter, en nuestro caso debemos seleccionar la opción en cruz “+”, para realizar una calibración de nivel seleccionamos la opción “Level”, se debe recordar que para realizar la calibración de nivel el dispositivo ArduCopter debe estar recto y nivelado.

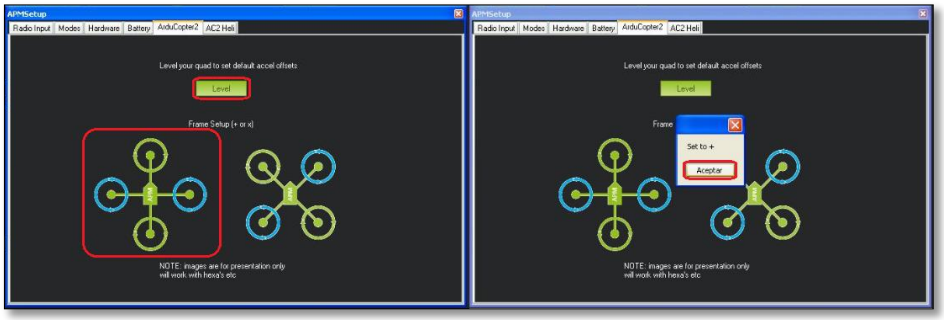

Figura 3.37: APM Planner – Configuración de Nivel y Marco de Vuelo
Fuente: Investigación de Campo

3.16. CALIBRACIÓN DE LOS CONTROLADORES DE VELOCIDAD - ESC

El objetivo de calibrar los controladores de velocidad de nuestro ArduCopter se debe a que los motores deben funcionar sincronizadamente al realizar un vuelo y de esta manera obtener un vuelo nivelado, así evitamos tener problemas de revoluciones bajas o una función errónea de los mismos como el caso de no responder a los comandos emitidos de nuestro radio control.

Se debe realizar este procedimiento al momento de haber realizado una actualización del código de ArduCopter, de igual manera si se ha reemplazado un motor o un variador de velocidad. Se debe tener en cuenta que los variadores no se calibrarán si no reciben una señal PWM, de esta manera sólo se escuchara una serie de sonidos lentos (pitidos) provenientes de los variadores, existen dos maneras de calibras los variadores, el método automático y método manual.

3.16.1. Calibración Automática

Es la más simple de realizar pero es necesario que todos los variadores estén conectados a los puertos de distribución de poder de la tarjeta electrónica lógica principal, por seguridad es necesario remover las hélices de los motores y de esta manera procedemos con las siguientes indicaciones:

1. La batería no debe estar conectada y se debe desconectar el cable USB.
2. Coloque el mando del acelerador al máximo (Throttle) y conecte la batería.

Figura 3.38: Posición Máxima del Mando del Acelerador
Fuente: Investigación de Campo

3. Al iniciar las funciones del APM1 se debe esperar hasta que las luces (A, B, C) se mantengan en un ciclo continuo de encendido y apagado.

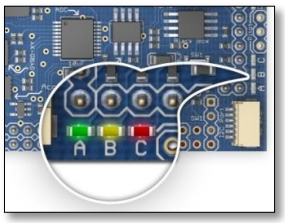

Figura 3.39: Luces Indicadoras A - B - C
Fuente: Investigación de Campo

4. Al mantenerse las luces en ciclo continuo se debe desconectar la batería y volver a conectarla inmediatamente, en este momento la señal máxima emitida del radio control es transmitida a los variadores y se produce un disparo de calibración de alta, al realizar esta acción el variador emitirá un sonido de confirmación (dos pitidos) al realizar correctamente el procedimiento.

Figura 3.40: Posición Mínima del Mando del Acelerador
Fuente: Investigación de Campo

5. A continuación sin desconectar la batería el mando del acelerador debe ser colocado en la posición mínima, se produce el disparo de calibración de baja, al realizar esta acción el variador emitirá un sonido de confirmación (dos pitidos) al realizar correctamente el procedimiento.

6. Ahora se debe mover el mando del acelerador lentamente para confirmar que los variadores están calibrados y sincronizados.
7. Desconectar la batería.

3.16.2. Calibración Manual

A diferencia del método automático la calibración manual se la debe realizar un variador a la vez, es necesario que se conecte el variador que va a ser calibrado al puerto de distribución de poder de la tarjeta electrónica lógica principal, por seguridad es necesario remover la hélice del motor y de esta manera procedemos con las siguientes indicaciones:

1. La batería no debe estar conectada.
2. Coloque el mando del acelerador al máximo (Throttle) y conecte la batería.
3. Al realizar el procedimiento anterior se debe escuchar un sonido de confirmación (dos pitidos), después de escuchar el sonido de confirmación inmediatamente se debe colocar al mando del acelerador en la posición mínima y se vuelve a emitir el sonido de confirmación, esto significa que los disparos de calibración de alta y baja se han emitido correctamente.
4. Desconectar la batería.
5. Repetir el procedimiento para cada motor.

3.17. ARMADO DE MOTORES

Es necesario realizar el procedimiento de armado de motores al terminar con la calibración de los variadores de velocidad, ya que cada vez que iniciemos nuestro dispositivo ArduCopter podremos observar que los motores no iniciaran sus funciones inmediatamente, por motivos de seguridad los motores se encuentran desarmados, por esta razón es necesario ejecutar las siguientes indicaciones:

1. Conectar la batería.
2. Se debe esperar unos momentos hasta obtener la confirmación de que el sistema esta listo para ser iniciado, el parpadeo continuo de las luces verde y roja (A y C) nos indica que el sistema está listo.

3. Ahora se debe mantener el mando del acelerador (Throttle) en la posición mínima y el mando de eje de cabeceo (Rudder) hacia la derecha.
4. Cuando la luz verde (A) se mantenga fija los motores y los variadores de velocidad estarán listos y armados para su funcionamiento, la luz roja (C) se mantendrá fija cuando llegue información al GPS.
5. Para desarmar los motores es necesario mantener el mando del eje de cabeceo (Rudder) hacia la izquierda hasta que la luz verde indicadora (A) mantenga un parpadeo contínuo.

Figura 3.41: Posición de Mandos de Control para Armado de Motores
Fuente: Investigación de Campo

Para el modelo de RC Futaba T6EXP el mando del acelerador (Throttle) debe estar en la posición máxima y el mando del eje de cabeceo (Rudder) hacia la derecha.

3.18. MODOS DE VUELO

Existen diferentes modos de vuelo, estos pueden ser establecidos utilizando el software APM Planner, también mediante la variación del canal 5 de nuestro radio control, los modos de vuelo y sus características serán mostrados a continuación:

Figura 3.42: Configurador de Modos de Vuelo APM Planner
Fuente: Investigación de Campo

Modo ACRO: Es un modo únicamente para realizar vuelos de velocidad, no recomendable para principiantes.

Modo ESTABILIZE: Es recomendable para realizar vuelos en primera persona (FPV).

Modo ALT HOLD: Al entrar en este modo de vuelo el sistema realizará las configuraciones necesarias para mantener el nivel de altitud actual.

Modo SIMPLE: Permite un vuelo libre sin restricciones, es decir, el usuario tiene el control de vuelo, este modo de vuelo es recomendable en caso de perder un motor.

Modo AUTO: El modo automático sigue la ruta aérea configurada por el planificador de vuelo, es necesario configurar una ruta de vuelo que contenga un retorno al punto de partida (RTL) ya que de no hacerlo el dispositivo ArduCopter volara libremente.

Modo RTL: Este modo le permite al dispositivo realizar un vuelo programado teniendo en cuenta que al finalizar la ruta debe retornar al punto de partida, también se puede configurar la altitud.

Modo GUIDED: Es similar al modo RTL con la diferencia de que se puede modificar el punto final de la ruta de vuelo.

Modo LOITER: Al activarse este modo el dispositivo ArduCopter mantiene su posición, velocidad y altitud, en este modo no es permitido aterrizar ni apagar motores.

Modo POSITION: Es similar al modo LOITER, pero el mando del acelerador está habilitado para el controlador.

Modo CIRCLE: Es similar al modo LOITER, pero se mantiene en una posición realizando un vuelo orbital (en círculos) sobre un objetivo, en este modo no es permitido aterrizar ni apagar motores.

3.19. PRIMER VUELO

Al finalizar la correcta sincronización de los controladores de velocidad (ESC), y después de haber configurado el modo de vuelo se procede a realizar un armado de motores para iniciar con el primer vuelo de prueba, el modo de vuelo adecuado para el usuario principiante es el modo “ESTABILIZE” ya que el dispositivo ArduCopter mantiene constantemente los motores sincronizados a una velocidad constante y el usuario puede realizar un reconocimiento de los mandos de control del RC inicial para observar como estos afectan el movimiento del dispositivo, al

realizar el armado de motores se procede a realizar una aceleración constante mediante el mando de aceleración (Throttle) sin mover los otros mandos de control, esto le permitirá al dispositivo elevarse de forma coordinada.

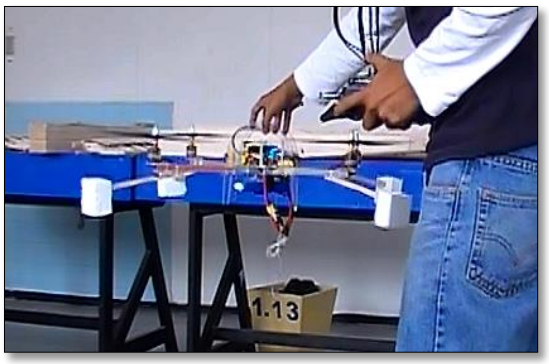

Foto 4: Primer Vuelo ArduCopter
Fuente: Cristian Zambrano

Mediante la adaptación del kit de telemetría podremos observar la ubicación geográfica exacta de nuestro dispositivo ArduCopter mediante el funcionamiento del dispositivo GPS incorporado en el mismo.

Figura 3.43: APM Planner - Ubicación Geográfica en Quito
Fuente: Cristian Zambrano

3.20. GASTOS REALIZADOS

3.20.1. Costos Primarios

A continuación en la Tabla 3.14 se detallan todos los dispositivos electrónicos y elementos principales utilizados para la realización del proyecto.

Tabla 3.14: Costos Primarios

Elemento	Cantidad	Valor Unitario	Valor Total
ArduCopter Quad V1.0	1	430.00	430.00
Hélices Modelo EPP 12x4	4	9.00	36.00
Batería Li-Po	1	50.00	50.00
		Total:	516.00

Elaborado por: Cristian Zambrano

3.20.2. Costos Secundarios

A continuación en la Tabla 3.15 se detalla el valor de los elementos que están indirectamente relacionados con la elaboración del proyecto.

Tabla 3.15: Costos Secundarios

Elemento	Valor Unitario	Valor Total
Derechos de Asesor	120	120
Internet	0.60	79.60
Arduino Mega 2560	80.00	80.00
Transporte	6.00	78.00
Materiales de Papelería	30.00	30.00
	Total:	387.60

Elaborado por: Cristian Zambrano

3.20.3. Costo Total

Los gastos totales se presentan en la Tabla 3.16.

Tabla 3.16: Costo Total

Gastos	Valor
Gastos Primarios	516.00
Gastos Secundarios	387.60
Total:	903.60

Elaborado por: Cristian Zambrano

CAPÍTULO IV

CONCLUSIONES Y RECOMENDACIONES

4.1. CONCLUSIONES

- Se ha logrado una correcta identificación de todos los elementos electrónicos que componen el sistema ArduCopter, estos dispositivos simulan el funcionamiento de principales instrumentos de vuelo obteniendo como resultado un sistema real de navegación aérea.

- Al analizar los parámetros de programación que ofrece Arduino podemos concluir que es aplicable para el desarrollo de una programación de control sencilla o una programación de control compleja, esto se debe a que Arduino fue desarrollado bajo el concepto de programación abierta, brindando al usuario un entorno de programación modificable y aplicable a sus necesidades.

- Las características del dispositivo Quadcopter ofrecen como resultado un aeromodelo fácil de maniobrar, esto es aplicable en entornos en los cuales es necesario un vuelo preciso como por ejemplo un vuelo de inspección.

- Las guías de prueba, calibración y configuración de los diferentes dispositivos instalados en el sistema ArduCopter permiten al usuario familiarizarse completamente con el funcionamiento del dispositivo en su totalidad.

4.2. RECOMENDACIONES

- Es necesario identificar correctamente los dispositivos periféricos adaptados al dispositivo ArduCopter, estos se dividen entre dispositivos analógicos y digitales, su funcionamiento, adquisición y procesamiento de información es diferente para cada dispositivo.

- Cada versión del código ArduCopter posee diferentes arreglos y actualizaciones realizadas por el autor, el presente trabajo fue realizado siguiendo los parámetros de la versión 2.5.3 ArduCopter y la versión 1.1.71 del APM Planner, si el usuario actualiza el código es necesario realizar las calibraciones necesarias antes de su utilización.

- Es necesario instalar los controladores FTDI para la sincronización APM1 - PC identificando correctamente el OS de nuestro ordenador, la mala instalación de estos controladores puede ocasionar una falla de comunicación entre nuestro dispositivo lo que ocasionará una mala compilación del código y un mal funcionamiento del mismo.

- Se debe recordar que el dispositivo GPS debe estar enganchado antes de realizar un vuelo con el dispositivo ArduCopter.

- Es necesario buscar una forma de optimizar el consumo de energía por parte de nuestro dispositivo ArduCopter, el tiempo de vuelo se ve limitado ya que los motores Brushless utilizados consumen un alto nivel de corriente.

- Ya que la estructura del dispositivo ArduCopter no es rígida es posible realizar múltiples cambios los cuales permitan incorporar paneles solares para la alimentación de nuestra tarjeta electrónica APM1 dejando libre la batería Li - Po para ser utilizada únicamente para alimentar los motores.

GLOSARIO DE TÉRMINOS

APM1: Ardupilot Mega 1.

BUS I²C: I²C es un bus de comunicaciones en serie, su nombre viene de Inter-Integrated Circuit (Circuitos Inter-Integrados).

BUS SPI: Es un estándar de comunicaciones, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos.

DRONES: Vehículos aéreos pilotados remotamente.

GPS: es un sistema satelital de posicionamiento.

IMU: Unidad de medición inercial.

MEMORIA FLASH: Es una tecnología de almacenamiento derivada de la memoria EEPROM que permite la lectura - escritura de múltiples posiciones de memoria en la misma operación.

OpenSource: El software OpenSource se define por la licencia que lo acompaña, que garantiza a cualquier persona el derecho de usar, modificar y redistribuir el código libremente.

OS: Sistema Operativo.

PPM: Modulación por posición de pulsos.

PWM: Modulación por ancho de pulsos.

RPV: Vehículo aéreo pilotado por control remoto (Remotely Piloted Vehicle).

RESOLUCIÓN: Es el número de píxeles que puede ser mostrada en la pantalla, se refiere a la agudeza y claridad de una imagen o una señal procesada.

SERVO: es un dispositivo actuador que tiene la capacidad de ubicarse en cualquier posición dentro de su rango de operación, y de mantenerse estable en dicha posición.

SENSOR: es un dispositivo capaz de detectar magnitudes físicas o químicas, llamadas variables de instrumentación, y transformarlas en variables eléctricas.

UAV: Vehículo aéreo no tripulado.

BIBLIOGRAFÍA

http://www.arduino.cc/es/

http://www.u-nav.com/3500fw.html

http://www.5hzelectronica.com/ardupilotarduinocompatiblecontroladoruavatmega328.aspx

http://www.rcjuampa.com.ar/product_info.php?products_id=1200&osCsid=c7e34712b6cdc9b9fecc210cf0328188

http://www.micropilot.com/products-mp2128g.html

http://www.didacticaselectronicas.com/index.php?option=com_virtuemart&page=shop.product_details&flypage=flypage.tpl&product_id=449&Itemid=6&vmcchk=1&Itemid=6

https://sites.google.com/site/mikuadricoptero/

http://translate.google.com.ec/translate?hl=es&sl=en&u=http://www.ssl.umd.edu/projects/RangerNBV/thesis/241.htm&ei=cbUMT_KyEIK4twfHgrWdBQ&sa=X&oi=translate&ct=result&resnum=1&ved=0CDUQ7gEwAA&prev=/search%3Fq%3Dsensores%2BIMU%26hl%3Des%26biw%3D1088%26bih%3D538%26prmd%3Dimvns

http://es.wikipedia.org/wiki/Unidad_de_medici%C3%B3n_inercial

http://diydrones.com/forum/topics/705844:Topic:460939?id=705844%3ATopic%3A460939&page=2#comments

http://es.wikipedia.org/wiki/Veh%C3%ADculo_a%C3%A9reo_no_tripulado

http://code.google.com/p/ardupilot-mega/