

Universidad Autónoma de Baja California
Instituto de Ingeniería
Maestría y Doctorado en Ciencias e Ingeniería



**DISEÑO Y EVALUACIÓN DE UN DRON
AUTÓNOMO BASADO EN
NAVEGACIÓN GPS**

Tesis que para obtener el grado de

MAESTRO EN CIENCIAS

Presenta

Tomy Antonio Namikawa Nakashimada

Director de Tesis

Dr. Félix Fernando González Navarro

Dedicatorias

La presente Tesis está dedicada a Dios, ya que gracias a él, he logrado llegar hasta aquí.

A mi amada esposa Liliana Mejorada, por su sacrificio y esfuerzo, por darme todo su apoyo y creer en mi capacidad, porque siempre me ha brindado su comprensión, cariño y amor.

A mi amada hija Hitomi, por ser mi fuente de motivación e inspiración para superarme cada día. Y por sacrificar el tiempo que no le dediqué.

A mi amado hijo Kenji, por ser mi fuente de motivación e inspiración para superarme cada día. Y por sacrificar el tiempo que no le dediqué.

A mis padres y hermanas, Takashi y Sumiko, Clara Aiko y Marisela quienes con sus palabras de aliento me motivaron a seguir adelante, especialmente a ti papá, que siempre creíste que podía dar más de mí.

A mi querida Obachan† que siempre, siempre me tuvo en sus oraciones y en su corazón.



Agradecimientos

Quiero expresar mi más sincero agradecimiento a todas las personas que con su soporte científico y humano han colaborado en la realización de este trabajo de investigación.

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el otorgamiento de la beca y ayuda económica brindada.

A la Universidad Autónoma de Baja California (UABC) por el apoyo brindado y darme nuevamente la oportunidad de continuar con mis estudios; y por el apoyo para la realización del proyecto “Diseño y Evaluación de un Dron Autónomo basado en navegación GPS”.

Al Instituto de Ingeniería que brindó todas la facilidades para poder realizar el trabajo de investigación.

Estoy muy agradecido con mi director de tesis y amigo Dr. Fernando González, que me animó y brindó la oportunidad de seguir estudiando; por su orientación, dedicación y sobre todo por su paciencia.

Quiero agradecer profundamente a Alcohólicos Anónimos, porque cada día me salva de la vida que tuve. ☺

A mi padrino Ricardo Montemayor por su amistad incondicional, por su comprensión y por ayudarme cuando más lo necesité.

A mis amigos del café, por escucharme atentamente cuando necesitaba hablar y por sus palabras de aliento. Especialmente a mi amigo Dr. Jesús Martínez por motivarme en el inicio de la redacción de esta tesis.

A mis cuñadas Lilibeth y Lizeth por ayudarnos con nuestros hijos cuando necesitaba estudiar. Y por los años anteriores. A sus familias también gracias.

A mis queridos suegros Gerardo† y Lilia que confiaron en mí y me brindaron su ayuda en tiempos difíciles. Por sus palabras de aliento y motivación.

RESUMEN de la Tesis de Tomy Antonio Namikawa Nakashimada, presentado como requisito parcial para la obtención de grado de MAESTRO EN CIENCIAS, Mexicali, Baja California, Junio de 2018.

DISEÑO Y EVALUACIÓN DE UN DRON AUTÓNOMO BASADO EN NAVEGACIÓN GPS

Aprobado por:

Dr. Félix Fernando González Navarro
Director de Tesis

La fama del dron aumenta a medida que los sensores y sistemas de control son más avanzados y menos costosos. Las marcas comerciales en el mercado actualmente que no son fáciles de comprender o configurar. El tiempo necesario para tomar un sistema existente, se puede invertir en diseñar una mejor solución. Este proyecto tiene como finalidad el uso claro de los sistemas y sensores como base para configurar y construir un dron de propósitos generales para la investigación. El sistema consiste de varios componentes de entrada de datos, una interfase de usuario Android, un controlador de vuelo y un microcontrolador Arduino. Inicialmente, la interfase Android, envía datos GPS al microcontrolador, el cual los procesa, filtra señales, estima estados del sistema, calcula entradas de control y maneja la comunicación. Se obtuvo satisfactoriamente la interpretación de las señales que recibe el controlador de vuelo. Se definieron las características de los componentes y sus algoritmos de prueba. Se ensamblaron y realizaron pruebas de vuelo. Se propone obtener un sistema abierto flexible adaptable o modificable para el monitoreo de variables ambientales.

Palabras clave: dron, sensores, Android, Arduino, navegación autónoma, GPS, microcontrolador.

ABSTRACT of a Thesis by Tomy Antonio Namikawa Nakashimada, submitted as a partial requirement to obtain a MASTER IN SCIENCES degree, Mexicali, Baja California, June 2018.

DRONE DESIGN AND EVALUATION BASED ON GPS AUTONOMOUS NAVIGATION

Approved by:

Dr. Félix Fernando González Navarro
Thesis Supervisor

Drone popularity grows as sensors and control systems are more advanced and cheaper. Currently, there are many brands on the market that aren't easy to understand nor configure. Necessary time spent on those systems could be invested on designing a better and customizable solution. This project aims to understand systems and sensors in easy manner as a foundation to configure and build an adequate drone for research purposes. This system has several components to gather data, interface with Android smartphone, flight controller and the Arduino microcontroller. First thing is Android device sends GPS data to the microcontroller in order to process it, it filters signals, estimate system states, calculate control inputs, and manage communication. There were good results on decoding signals that the flight controller receives in order to tell it what to do. Other components were also reviewed and tested using individual algorithms. Build of all components and flight tests were performed. An open-source system is proposed that can be easily adapted for specific needs without closed-source limitations.

Keywords: drone, sensors, Android, Arduino, Autonomous navigation, GPS, microcontroller.

Índice

1	Introducción	1
1.1	Planteamiento del problema	3
1.2	Objetivo General	3
1.3	Metodología	4
1.4	Estructura General de la Tesis	4
2	Marco Teórico	6
2.1	Drones: Definición y Clasificación	6
2.2	Aplicaciones de los drones	9
2.3	Quadrirrotor	13
2.3.1	El Modelo Matemático del quadrirrotor	15
2.3.2	El modelo Newton-Euler	23
2.4	El filtro Kalman	34
2.4.1	Introducción	34
2.4.2	Estimación del Proceso	35
2.4.3	El origen Computacional del Filtro	36
2.4.4	El Algoritmo del Filtro de Kalman Discreto	38
2.4.5	Parámetros del filtro y ajustes finos	40
2.5	Sistema de Posicionamiento Global	41
2.5.1	Estándar NMEA	42
2.5.2	Formato del mensaje NMEA	43
2.6	Diseño del Hardware	45
2.6.1	Microcontrolador	45
2.6.2	Sistemas de Comunicación	52
2.6.3	SPI - Serial Peripheral Interface	62

2.6.4	Inter-Integrated Circuit - I2C	67
2.6.5	Aplicación	69
2.6.6	¿Cuales son las partes de un dron?	70
2.6.7	Chasis	70
2.6.8	Batería	71
2.6.9	Motores	75
2.6.10	Hélices	77
2.6.11	Controlador de Velocidad del Motor	78
2.6.12	Controlador de Vuelo	80
2.6.13	Transceptor de Radio	84
2.6.14	Bluetooth®	86
2.6.15	Módulo micro SD	88
2.6.16	Sensor Ultrasónico	90
3	Materiales y Métodos	92
3.1	Tamaño del Chasis	93
3.2	Sistema Eléctrico de Propulsión	96
3.2.1	Batería	96
3.2.2	Hélices	98
3.2.3	ESC	99
3.2.4	¿Qué es <i>eCalc</i> ?	100
3.2.5	Motores	108
3.3	Sensores y Microcontrolador	108
3.3.1	Bluetooth	108
3.3.2	micro SD	110
3.3.3	Sensor Ultrasónico	111
3.3.4	Naza M Lite	114
3.3.5	El Microcontrolador	116
3.4	Software	122
3.4.1	Introducción	122
3.5	Software de Arduino	124
3.6	Software en el Celular	125
3.6.1	¿Qué es MIT App Inventor?	126

3.7	La App	128
3.8	Módulo de Motores	129
3.8.1	Calibración de ESCs	129
3.8.2	Armar los motores	130
3.8.3	Despegue Vertical	130
3.9	Módulo Bluetooth	131
3.10	Módulo micro SD	132
3.11	Módulo Ultrasónico	133
3.12	GPS + Brújula	133
3.13	Control de Vuelo	136
3.14	Control de Navegación	138
3.14.1	Generación de puntos de navegación	140
3.14.2	Dirección-relativa y Distancia	142
4	Resultados y Discusión	144
4.1	Prueba de Vuelo	144
4.2	Ajuste de Parámetros	146
5	Conclusiones	152
5.1	Conclusiones	152
5.2	Trabajo Futuro	154
5.3	Limitaciones	155
A	Ensamble del Chasis	157
A.1	Ensamble del Chasis	157
A.2	Acoplamiento Motor-Base	159
A.3	Montaje Motor-Brazo y Alambrado	160
A.4	Ensamble Brazos-Base Inferior	160
A.5	Tipos de tubos	161
A.6	Ensamble Tren de aterrizaje y Gimbal	162
A.7	Ensamble Base Superior y Base de Batería	163

B Código	166
B.1 Calibración de ESC	166
B.2 Armar los motores	167
B.3 Despegue Vertical	171
B.4 Calibración de Puertos Naza	177
B.5 Dirección-Relativa y Distancia	181
Abreviaciones	183
Notaciones	186
Referencias	191

Lista de Figuras

2.1	Parrot de ala fija (Parrot.SA., 2017)	8
2.2	USAF Predator (Turbosquid2018, 2013)	8
2.3	Pequeños drones en la meteorología de la capa límite (Smith, 2017)	13
2.4	Pequeños drones en la meteorología de la capa límite (Smith, 2017)	14
2.5	Octacóptero Skyhawk (Shenzhen SkyhawkRC Technology Co., 2018)	14
2.6	Steady Drone Q4DX (akrcguy, 2014)	14
2.7	USAF RQ-4 Global Hawk (US-Air-Force, 2014)	14
2.8	USAF RQ-4 Global Hawk (US-Air-Force, 2014)	14
2.9	Dron profesional para vídeo filmación (DIEDRO, 2015)	15
2.10	Drones en eventos sociales (Ward, 2017)	15
2.11	Las variables de Control del quadrirrotor (en la izquierda, el sistema o marco inercial, en la derecha, el marco del chasis (Luukkonen, 2011).	16
2.12	<i>Modelo simplificado del quadrirrotor en vuelo estacionario (Bresciani, 2008)</i>	19
2.13	<i>Modelo simplificado del quadrirrotor en vuelo estacionario (Bresciani, 2008)</i>	20
2.14	<i>Movimiento lateral en vuelo estacionario (Bresciani, 2008)</i>	21
2.15	<i>Movimiento frontal y trasero en vuelo estacionario (Bresciani, 2008)</i>	22
2.16	<i>Giro entorno a z en vuelo estacionario (Bresciani, 2008)</i>	23

2.17 Ciclo del filtro Kalman discreto. La <i>predicción</i> proyecta la estimación del estado actual en el tiempo. La <i>corrección</i> ajusta la estimación proyectada por una medición actual en ese tiempo. (Bishop <i>et al.</i> , 2001)	38
2.18 Imagen completa de la operación del filtro Kalman, combinando el diagrama de alto-nivel de la figura 2.17 con las ecuaciones de las tablas 2.1 y 2.2 (Bishop <i>et al.</i> , 2001).	40
2.19 Constelación de satélites GPS (GPS.gov, 2018)	42
2.20 Diagrama de bloques del microcontrolador. Los microcontroladores están equipados con los sub-sistemas mostrados.	46
2.21 Conjunto de Registros de un microcontrolador (Register Set)	47
2.22 Conceptos de Temporización. El voltaje DC efectivo que se entrega a una carga se puede variar a través de ajustar el ciclo de trabajo de una señal PWM digital.	50
2.23 Bus de datos de 8-bits, controlado por una señal de reloj, transmitiendo un byte cada pulso de reloj, se requieren 9 conductores (Jimbo, 2018)	53
2.24 Ejemplo de una interfase serial, transmitiendo un bit cada pulso de reloj con dos conductores (Jimbo, 2018).	53
2.25 Algunos símbolos en el paquete de datos tienen tamaños de bits configurables (Jimbo, 2018).	55
2.26 Conexión serial UART (Jimbo, 2018).	57
2.27 TTL serial (Jimbo, 2018).	58
2.28 RS-232 serial (Jimbo, 2018).	59
2.29 Interfase UART simplificada (Jimbo, 2018).	60
2.30 Diagrama de bloques interno del UART (Jimbo, 2018).	61
2.31 Trama de bits serial UART (MikeGrusin, 2018).	62
2.32 Trama de bits de datos serial sincrónica y pulsos de reloj (MikeGrusin, 2018).	63
2.33 Trama de bits datos de entrada y salida y los pulsos de reloj en Bus SPI (MikeGrusin, 2018).	65
2.34 Trama de bits datos de entrada y salida y los pulsos de reloj en Bus SPI utilizando SS (MikeGrusin, 2018).	66

2.35 Conexión multi-esclavos en bus SPI (MikeGrusin, 2018).	67
2.36 Conexión Maestro y esclavos usando mismo bus de datos, en protocolo I2C (SFUPTOWNMAKER, 2018).	68
2.37 Batería LiPo 7.4V (2S) 1500mAh 25C (Modeltronic, 2018)	73
2.38 Motor DC 2212 1000KV para multirrotor	75
2.39 Componentes de un motor Outrunnersin escobillas: Rotor y Es- tátor. En este tipo de motor la carcasa está unida al rotor (Sam, 2014)	76
2.40 ESC de 30A (banggood.com, 2018)	79
2.41 Diagrama simbólico de la conexión ESC-Motor	80
2.42 NRF24L01 tarjeta básica (Tutoriales, 2016)	85
2.43 NRF24L01 tarjeta de potencia y filtros (Tutoriales, 2016)	85
2.44 módulo HC-05 Bluetooth 2.0	86
2.45 Pinout del Módulo Bluetooth	88
2.46 Conexión de un módulo Bluetooth	88
2.47 tarjeta microSD (Lady-ada, 2017)	89
2.48 Módulo micro SD (Lady-ada, 2017)	89
2.49 Emisión-Recepción de ondas ultrasónicas (Kelemen et al., 2015)	90
2.50 Diagrama de Tiempo de un sensor ultrasónico (Electrical Engi- neering, 2016)	90
3.1 El tamaño del chasis se mide del centro de un motor al centro del motor opuesto (LearningRC, 2017)	94
3.2 Chasis de 650mm para Quadrirrotor (unmannedtech, 2012)	94
3.3 Aeroquad - descontinuado (aeroquad, 2018)	95
3.4 Tarot 650: Chasis fibra de carbono (QWinOut, 2017)	95
3.5 Batería LiPo	97
3.6 Hélices Tarot Carbon Fiber 1455 y 1455R (eCalc, 2018)	98
3.7 Punta de ala mezclada (Blended winglet) (eCalc, 2018)	98
3.8 Electronic Speed Controller - ESC (banggood.com, 2018)	99
3.9 eCalc - Multicópteros (Mueller, 2017)	101
3.10 Sección General de eCalc (Mueller, 2017)	102
3.11 Sección Celdas de Batería en eCalc (Mueller, 2017)	102

3.12 Sección Controlador de Velocidad del Rotor en eCalc (Mueller, 2017)	103
3.13 Configuración de la Hélice	103
3.14 Asistente KV hélice (Mueller, 2017)	104
3.15 Lista de modelos compatibles según las especificaciones (Kelly, 2014)	105
3.16 Estimaciones de los parámetros principales en eCalc (Kelly, 2014)	106
3.17 Resultados del cálculo según especificaciones primera página de eCalc (Kelly, 2014)	107
3.18 Resultados del cálculo según especificaciones segunda página de eCalc (Kelly, 2014)	107
3.19 Motor DCBL Outrunner modelo Turnigy MultiSTAR 4822-690 (hobbyking.com, 2018)	108
3.20 Diagrama esquemático de conexión Arduino-Bluetooth a través de UART	109
3.21 Diagrama simbólico de conexión Arduino-Bluetooth a través de UART	110
3.22 Conexión Micro SD al puerto SPI (Lady-ada, 2017)	111
3.23 Conexión Arduino a micro SD	111
3.24 Conexión Arduino-Sonar - Diagrama esquemático	113
3.25 Conexión Arduino-Sonar - Diagrama simbólico	114
3.26 Controlador o estabilizador de vuelo modelo Naza M Lite	116
3.27 Dron en configuración "X"	116
3.28 Arduino Due (arduino.cc, 2018)	118
3.29 Raspberry Pi 3 (Raspberry.org, 2018)	119
3.30 Diagrama esquemático del dron prototipo	121
3.31 Diagrama simbólico del dron prototipo	121
3.32 Arduino - Entorno Integrado de Programación ??	124
3.33 Participación en el Mercado de Sistemas Operativos a nivel mundial de Abril 2017 - Abril 2018 (Pi, Charly, 2018)	125
3.34 Vista de Diseño (Interfaz con el usuario)	127
3.35 Vista de Bloques (código)	127
3.36 Bloques para Enviar Cadena	128

LISTA DE FIGURAS

LISTA DE FIGURAS

3.37 Trama de datos de GPS (Hagiopol, 2016)	135
3.38 Ancho de Pulsos del receptor de RC (Parallax Inc., 2018)	136
3.39 Receptor RC - Salidas al controlador de vuelo (Parallax Inc., 2018)	136
3.40 Diagrama de Bloques del Sistema	139
3.41 Diagrama de flujo de control de Navegación	140
3.42 Matriz de puntos de navegación (N=10)	141
3.43 Rumbo, Dirección-relativa, Dirección Magnética y Norte Magnético (Administrador, 2017)	143
4.1 Ensamble final del dron prototipo	144
4.2 4 sujetadores por lado	146
4.3 Ajustando parámetros de Despegue	147
4.4 Pruebas de Velocidad utilizando Ecuación 4.1	147
4.5 Dron prototipo en Vuelo	149
4.6 Curvas de Velocidad	150
4.7 Estimación de Tiempo de Vuelo y Alcance [Diseño propio]	151
A.1 Diagrama de ensamble general del chasis (unmannedtech, 2012)	158
A.2 Diagrama de ensamble del núcleo (unmannedtech, 2012)	158
A.3 Armado de Bases y Motores	159
A.4 Montaje de motor en Base	159
A.5 Montaje del Motor orientado	160
A.6 Diagrama de ensamble de Base inferior (unmannedtech, 2012) .	161
A.7 Ensamble de la base inferior (unmannedtech, 2012)	161
A.8 Tubos de gimbal y del tren de aterrizaje (unmannedtech, 2012) .	162
A.9 Colocacion de la Base de Gimbal (unmannedtech, 2012)	162
A.10 Colocacion del Tren de Aterrizaje (unmannedtech, 2012)	163
A.11 Montaje de la base de la bateria (unmannedtech, 2012)	164
A.12 Ensamble Final del núcleo del Prototipo	164
A.13 Ensamble Final del Prototipo	165

Lista de Tablas

2.1	Ecuaciones de predicción de filtro Kalman discreto	39
2.2	Ecuaciones de corrección de filtro Kalman discreto	39
2.3	Voltajes de Batería	72
3.1	Tarot 650 Kit (QWinOut, 2017)	95
3.2	Modelos con chasis de 650mm	96
3.3	Características de la Batería (HobbyStar, 2018)	97
3.4	Características del ESC (banggood.com, 2018)	99
3.5	Variables de Entrada a eCalc	100
3.6	Naza M Lite - Tabla del Pinout	115
3.7	Elementos Periféricos Requeridos	117
3.8	Arduino Due vs. Raspberry Pi	119
3.9	Tabla de Puertos del Controlador de Vuelo	137
4.1	Pruebas de Despegue	147
5.1	Comparación de precios de Drones	154

Capítulo 1

Introducción

El proyecto que aquí se presenta es de carácter teórico-práctico y fue desarrollado en el Laboratorio de Inteligencia Artificial del Instituto de Ingeniería de la Universidad Autónoma de Baja California. El presente trabajo se refiere al diseño e implementación de un dron. También llamado Vehículo Aéreo No Tripulado (VANT) o en inglés UAV (Unmanned Aerial Vehicle). Los drones son robots voladores que pueden volar miles de kilómetros, o pequeños drones que vuelan en espacios reducidos (Krijnen & Dekker, 2014). Los drones que no transportan operadores humanos, de vuelo remoto o autónomo, que llevan carga letal o no-letal, todos estos son considerados drones (Gupta *et al.*, 2013). Las características apreciables en un dron son: larga duración de vuelo, mejoramiento en la seguridad de misión, repetibilidad de vuelo debido al mejoramiento de vuelo autónomo y la reducción de costos de operación comparado con los sistemas tripulados (Watts *et al.*, 2012). En los 1990s, la NASA (National Aero Space Agency) a través de su programa de Aeronaves para la investigación ambiental y la tecnología de sensores ERAST (Environmental Research Aircraft and Sensor Technology), marcó los primeros grandes avances hacia el desarrollo de protocolos y capacidades en el uso de drones para la investigación científica. Sin embargo, había una restricción, la mayoría de los drones capaces de llevar potentes sensores de precisión eran grandes y de alto costo, mientras que las pequeñas plataformas carecían de precisión en los datos.

A finales de esa misma década, inspirados por los esfuerzos y mejoras de

1. INTRODUCCIÓN

la NASA en la miniaturización, se vieron iniciativas de pequeñas organizaciones en el desarrollo o modificación de drones confeccionados a sus propias necesidades. La era del *Hágalo usted mismo* evolucionó rápidamente ya que los investigadores encontraron soluciones de bajo costo basados en modelos de radio control modificados. Además, el reconocimiento de drones que ofrecen grandes potenciales para muchas áreas de investigación científica y monitoreo, ocasionó una demanda significativa en el sector público para sistemas que combinan sensores o muestreos con una calidad típicamente encontrada en las grandes aeronaves, con las ventajas de portabilidad y costo de aeronaves más pequeñas (Watts *et al.*, 2012).

Los drones ejecutan misiones en interiores y exteriores en condiciones difíciles. Los drones están equipados con sensores y cámaras para misiones de inteligencia, vigilancia y reconocimiento. Las aplicaciones se basan según la misión (militar o civil) y tipos de ambientes de vuelo (exteriores o interiores) y (acuáticos, terrestres, aéreos o espaciales). Los drones tienen una variedad de aplicaciones en nuestra vida cotidiana. El uso del dron se ha aplicado en diversas áreas, como sistemas de seguridad privados, en la agricultura, control de incendios forestales, búsqueda y rescate de personas, vigilancia fronteriza, eventos sociales y deportivos, evaluación de desastres, inspección de estructuras altas como chimeneas o torres de alta tensión, muestreo atmosférico (Nyquist, 1997) y monitoreo de la vegetación en los pastizales (Quilter & Anderson, 2001).

El crecimiento de la mancha urbana lleva incluido el incremento de materiales en la construcción, en carpetado asfáltico, industria, viviendas y la deforestación. Provoca mayor acumulación de calor debido a la absorción de los materiales. Mientras que en los alrededores suburbanos la temperatura es menor. A esta diferencia de temperaturas se le conoce como Isla de Calor Urbano (ICU). Este fenómeno está presente en todas las ciudades del mundo. Sin embargo, en las ciudades con climas extremos es más evidente. Esto se traduce en riesgos para la salud (golpes de calor, deshidratación, muerte), mayor consumo de energía por los equipos de refrigeración, mayor consumo de agua. En la ciudad de Mexicali, se han realizado estudios para la detección de la ICU mediante la creación de modelos matemáticos (Casillas-Higuera *et al.*, 2014) y

por medio del uso del suelo industrial (Casillas, 2009), se han analizado casos de mortalidad asociadas a este fenómeno y análisis de la atmósfera (Díaz Caravantes *et al.*, 2014).

1.1 Planteamiento del problema

Según los estudios realizados en la ciudad de Mexicali, hay dos fuentes de datos para detectar una ICU: imágenes satelitales y datos de estaciones meteorológicas. Las imágenes satelitales tienen una resolución de 1.1 km (Cueto), no ofrecen mayor definición y hay solamente 4 estaciones meteorológicas. Estos datos se pueden complementar con los datos que se obtengan de sensores instalados en drones a medida que ejecutan un plan de vuelo (posición geográfica vs temperatura, humedad, presión atmosférica, radiación solar) obteniendo una mejor resolución y tener la capacidad de detección de puntos más calientes dentro de la ciudad. Esto beneficiaría a la comunidad a través de prevenir riesgos de salud, promover la forestación urbana y el uso de materiales más eficientes.

1.2 Objetivo General

El objetivo general de esta investigación es diseñar y construir un dron funcional tipo quadrirrotor, de vuelo autónomo, pequeño y de bajo costo. La obtención de la base del conocimiento para su diseño y construcción. Y se utilice para expandir el entendimiento de las condiciones atmosféricas en zonas urbanas.

Objetivos Específicos

1. Diseñar un prototipo de mecanismo para monitoreo de variables ambientales con capacidad de navegación autónoma.
2. Instrumentar electrónicamente el dron con un estabilizador de vuelo.
3. Instrumentar electrónicamente el dron con un microcontrolador.

4. Justificar la selección del microcontrolador, como elemento inteligente de control y navegación.
5. Realizar escenarios de pruebas controladas y de navegación.

1.3 Metodología

En esta tesis se hará un intensivo inicial sobre los drones. Se ampliará el estudio del arte y conocer las tecnologías existentes, sus características, tipos, aplicaciones y modelos matemáticos.

En el diseño, se determinarán las características de los componentes electrónicos necesarios y la plataforma del microcontrolador. Se probará su funcionamiento y calibración. Para la motorización del dron, se identificarán los tamaños, características eléctricas y la relación entre sus componentes, selección, ensamble y pruebas de funcionamiento. Todas las pruebas de funcionamiento se realizarán con el microcontrolador y códigos en C++.

A través de este documento, obtendremos un criterio para seleccionar las mejores combinaciones de componentes y se construirá un modelo funcional. El alcance y complejidad hacen de este proyecto, una gran plataforma de aprendizaje.

1.4 Estructura General de la Tesis

Esta tesis está organizada de la siguiente manera:

El Capítulo 2 define lo que es un vehículo aéreo no tripulado, una breve historia de su origen, sus clasificaciones y aplicaciones. Luego hace un enfoque en los drones del tipo multi-rotor y las características de vuelo así como algunas ventajas. Se definen los modelos matemáticos del dron, el filtro Kalman y su algoritmo. Se repasan conceptos relacionados con el sistema de posicionamiento global (GPS). Se describe con detalle la electrónica y motorización de la plataforma del dron.

El Capítulo 3 describe el tipo de estructura utilizado y las ventajas en los materiales utilizados frente a otros que hay en el mercado para esta aplicación. Establece los elementos básicos del dron: chasis, hélices, motores,

controladores de velocidad (ESC) y batería. Describe los elementos de programación involucrados en el dron: lenguaje de programación, calibraciones, comunicación, módulo de vuelo, sensores y el microcontrolador. Se describe la aplicación en Android (App) y el lenguaje utilizado. Se describen los códigos utilizados en módulo de motorización.

El Capitulo 4 describe las pruebas de vuelo, las respuestas observadas.

El Capitulo 5 se describen las conclusiones y trabajos futuros.

Capítulo 2

Marco Teórico

En este capítulo, se expondrán algunos conceptos básicos de aerodinámica, los drones, su clasificación y aplicaciones así como iniciativas de empresas en aplicaciones particulares. Se ha revisado información de distintos trabajos con drones, analizando las distintas plataformas que se desarrollan en la actualidad; diseños de compañías y/o proyectos universitarios. De acuerdo a los datos obtenidos se ha seleccionado la plataforma que más y mejor se ajusta a los requisitos del proyecto.

Se analizaron los distintos componentes del sistema como el que se quiere construir, con enfoque al tipo quadricóptero, por ser uno de los más económicos y nobles en la operación de vuelo. Se realizó un estudio sobre el actual estado de la técnica para la construcción de quadricóptero. Se analizaron los sistemas de mecanismo aéreo, motores, hélices, controladores, etc. por sus características y se realizó una simulación de su capacidad, con el fin de elegir los elementos adecuados para obtener el impulso deseado y otras características para las tareas requeridas.

2.1 Drones: Definición y Clasificación

El dron se entiende como un vehículo inhabitado, re-utilizable y motorizado, controlado de manera remota, semi-autónoma, autónoma o que tenga una combinación de estas capacidades; que lleva varios tipos de carga, siendo capaz de ejecutar tareas específicas dentro o fuera de la atmósfera terrestre por

un tiempo determinado y con una misión (Van Blyenburgh, 1999). El dron es un sistema de vuelo que no requiere de un piloto a bordo ya que a través de una serie de unidades de procesamiento, sensores incorporados y un enlace de comunicación, es guiado autónomamente basado en misiones pre-programadas, o controlado de forma remota (Peng *et al.*, 2009).

Los drones han evolucionado rápidamente en la última década principalmente en el uso militar, y se ha comenzado la exploración de aplicaciones en la comunidad civil (Watts *et al.*, 2012) para el reconocimiento por medio de sensores remotos y el propósito científico para recolección de datos. Los avances en la fabricación, navegación, las capacidades del control remoto y los sistemas de almacenamiento de energía, han hecho posible el desarrollo de una amplia gama de drones los cuales son utilizados en diversas situaciones donde la presencia humana es difícil, imposible o peligrosa (Bachmann *et al.*, 2009; Hassanalian *et al.*, 2015).

Los drones tienen una gran variedad de configuraciones dependiendo de la plataforma, la misión y ciertos parámetros. En (Watts *et al.*, 2012) se clasifican, para el uso civil y militar basadas en el tamaño, duración de vuelo, y sus capacidades específicas. (Arjomandi *et al.*, 2006) clasifica los drones basados en el peso, rango y duración, carga en las alas, máxima altitud y tipo de motor. (Gupta *et al.*, 2013) clasifica en tres principales tipos: micro y mini, tácticos y estratégicos. Es más, los drones pueden ser clasificados basados en el tipo de motor (Arjomandi *et al.*, 2006), los hay de combustión interna y eléctricos. Los drones más populares son los de ala fija y de rotación:

Drones de ala fija: Usualmente consiste de alas rígidas, fuselaje y cola, que usa motor y hélice como su sistema de propulsión y puede cubrir un rango amplio de posibles ambientes de operación, como la selva, el desierto, la ciudad, el mar, la montañas y el ártico (Hassanalian & Abdelkefi, 2017)

Este tipo de drones, tienen una estructura sencilla y alta eficiencia aerodinámica, con tiempos de vuelo más largos a mayor velocidad. Tienen una autonomía igual a 1 hora de vuelo a velocidades de hasta 100 km/h con una cobertura amplia en terreno. Por lo que son utilizados para reconocimiento aéreo, con la captura de imágenes geo-referenciadas. Como se muestra en las Figuras 2.1 y 2.2.



Figura 2.1: Parrot de ala fija
(Parrot.SA., 2017)

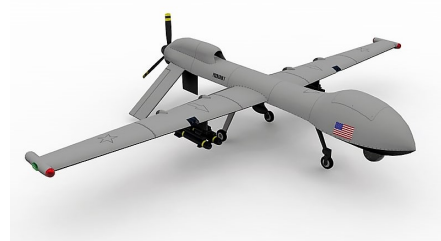


Figura 2.2: USAF Predator
(Turbosquid2018, 2013)

Para este tipo de aeronaves, el despegue se hace a través de una lanzadera, o una pista en caso de poder sustentar el despegue y para el aterrizaje.

Dron de ala rotatoria: Este tipo de aeronaves consta de una o varias hélices que rotan debido al o los motores. Una de las grandes ventajas sobre otros drones es que pueda volar en espacios reducidos (Krijnen & Dekker, 2014). Como se muestran en las Figuras 2.5 y 2.6, al tener sistemas basados en hélices o navajas rotatorias se les llama drones de ala rotatoria. A diferencia de los modelos ala fija, estos drones pueden volar en cualquier dirección, horizontalmente, verticalmente y también pueden mantener el vuelo suspendido (hovering) en una posición fija (Austin, 2011). Son más lentos y el rango de vuelo es limitado en comparación con los ala fija. También es llamado Multicóptero o Multirroto como el Bicóptero (dos rotores), Tricóptero (tres rotores), Quadricóptero (cuatro rotores), Pentacóptero (cinco rotores), Hexacóptero (seis rotores), Octacóptero (ocho rotores), Decacóptero (diez rotores), Dodecacóptero (doce motores) (Ubay & Iqbal, 2015). De entre ellos el más popular es el Quadricóptero por su mayor entendimiento de los modelo matemáticos que rigen su aerodinámica y por el costo reducido en comparación con otros multicópteros con mayor número de rotores. La ventaja que ofrece el multicóptero es la capacidad de despegue y aterrizaje vertical por sus siglas en inglés Vertical Take-Off and Landing (VTOL), según (Arjomandi *et al.*, 2006) los multicópteros que tienen esta única característica, se encuentran en una clasificación a parte. Ellos no requieren pista de aterrizaje convencional, y son usados donde hay los arbustos del campo o en el bosque, o algún tipo de impedimento para el uso de aeronaves que requieran despegue o aterrizaje horizontal (HTOL) (Arjomandi

et al., 2006). Los drones con VTOL, también son usados para la entrega de abastecimiento con una mejor precisión que las aeronaves que sueltan la carga en pleno vuelo horizontal.

2.2 Aplicaciones de los drones

Drones grandes y chicos pueden ser usados en una amplia variedad de aplicaciones militares, comerciales, monitoreo ambiental, telecomunicaciones y en búsqueda y rescate (Miller, 2006), agricultura y meteorología (Miller, 2006).

Uso militar

Desarrollan numerosas operaciones como señuelo para fuego terrestre o aéreo en prácticas simulando al enemigo. Para uso de inteligencia, reconocimiento y vigilancia, **ISR** ((Intelligence, surveillance and reconnaissance). Tienen un sistema de manejo y transporte de cargas variadas y tienen la capacidad de desarrollar aplicaciones de combate. Para el éxito de los conflictos bélicos es más probable que se acelere el uso de los drones en la estructura militar sobre todo para misiones de reconocimiento, como el RQ-4 Global Hawk que se puede ver en las figuras 2.7 y 2.8. El RQ-4 Global Hawk es una aeronave pilotada remotamente para grandes alturas y una larga duración de vuelo con un portafolio integrado de sensores que provee todos los datos climáticos, de inteligencia tanto de día como noche, vigilancia y reconocimiento (US-Air-Force, 2014).

Control mediante sensores

Al incorporar un juego de sensores, es posible informar de manera remota al usuario de lo que ocurre alrededor del dron. Sensores de campo electromagnético, visor de infrarrojos, sistemas de radar, biológicos, nucleares, químicos, **NBR** (Nuclear, Biological and Radiation) (Miller, 2006).

Telecomunicaciones

Como nodo (relay) de interconexión en sistemas de comunicación. Google Access estaba utilizando drones impulsados por energía solar en Nuevo México

con el objetivo de probar transmisiones de Internet de alta velocidad. Facebook privilegió la cobertura para brindar conectividad en áreas de difícil acceso.

Asistencia en desastres

Transportan cargas como medicamentos y vacunas en sitios donde el acceso es difícil o de riesgo para la salud por contagio. Los drones pueden ayudar en el estudio de desastres naturales y en esfuerzos de búsqueda y rescate y fuegos forestales. En Diciembre del 2004, la fuerza naval de la India envió los drones a las islas de Andaman y Nicobar en búsqueda de sobrevivientes del tsunami (Times, 2015). Drones grandes y minis sobrevolaron el río Missisipi en el estado de Louisiana tras el Huracán Katrina en 2005 para la valoración de daños y ayuda para el rescate (Murphy *et al.*, 2008).

Campo Comercial

Gracias a la capacidad de control de vuelo y movilidad en los drones, es posible grabar imágenes o vídeo para propósito comercial o particular. Ejemplo de esto se observa en los eventos deportivos, sociales y de filmación de películas. Ver figuras 2.9 y 2.10. Compañías de mantenimiento e inspectores de gobierno usan drones de bajo costo para el monitoreo e inspección del cableado de electricidad y de comunicaciones en áreas de acceso limitado. Drones con cámaras montadas y sensores pueden ser usados para detectar líneas caídas, cables rotos, torres dañadas, conectores desgastados y vegetación excesiva (Miller, 2006).

Agricultura

Control de plaga y el monitoreo de la salud y crecimiento de las plantas son dos de los usos del dron. En (Psirofonía *et al.*, 2017) se describe el uso de aplicaciones de detección de síntomas (decoloración en hojas) de plaga o infestación de enfermedades en los árboles de olivo, en palmeras datileras y para fumigar, donde trampas electrónicas cuentan el número de insectos y estos datos son enviados a un servidor que envía un dron para rociar de fumigante la zona afectada.

Asistencia médica

Entrega de defibriladores portátiles compactos para una respuesta rápida en casos de paros cardíacos (Fleck, 2016).

Investigaciones científicas

Los drones en esta área, algunos de mayor peso capaces de entrar en los huracanes, mandar los datos en tiempo real a la base de lo que está ocurriendo en el huracán. También en el monitoreo y predicción del clima (Oklahoman, 2015). Por otro lado, el uso de sensores para recabar imágenes en alta resolución y formar mapas de la distribución de temperatura en la superficie y de las propiedades de los materiales en zonas urbanas (Gaitani *et al.*, 2017; THIS *et al.*, 2018).

Por otro lado, existe un estudio realizado (Coutts *et al.*, 2016), donde también utilizan imágenes térmicas para la identificación de puntos calientes (el aire urbano de altas temperaturas), o *hotspots*, en la ayuda de la planeación y tomas de decisión para la re-forestación urbana. Esto asumiendo que los patrones de temperatura del suelo coinciden con los patrones de la temperatura del aire. Este estudio identifica *hotspots* en una escala de vecindarios y calles con uso de datos de muy alta resolución al utilizar sensores de temperatura infrarrojos. Sin embargo, este estudio muestra que no es una herramienta adecuada debido a la complejidad y limitaciones de los datos. Este estudio sugiere el uso de recursos satelitales más baratos, como el Landsat. Dos años antes, ya se había realizado un estudio en (Zhou *et al.*, 2014) utilizando datos del Landsat-7 con tres datasets a diferentes resoluciones cada uno, encontrando que no afectaba significativamente la relación entre la temperatura de la superficie del terreno (LST-Land Surface Temperature) y las variables cobertura de terreno-uso de terreno (LULC-land usage/land cover). El estudio concluyó que las variables relacionadas con la vegetación particularmente con la copa de los árboles, eran buenas predictivas del LST durante el verano y otoño.

Isla de Calor Urbano - ICU

Una ICU (UHI, Urban Heat Island) es simplemente el calor de un poblado o ciudad, la cual se encuentra presente en casi todas las ciudades de todas las regiones climáticas (Oke, 1995). Este fenómeno es designado como “Isla” porque los patrones isotérmicos de temperatura del aire cerca de la superficie se parecen a los contornos de una isla en un “mar” campestre de más fresca (Oke, 1995). Estas diferencias de temperatura afectan el uso de energía y la habitabilidad de las ciudades debido a que los techos de edificios se calientan más y por eso, elevan las demandas de refrigeración en los edificios (Akbari *et al.*, 2001). Colectivamente, las superficies oscuras y la reducción de vegetación calientan el aire sobre las áreas urbanas. (Akbari *et al.*, 2001). Según este estudio en una ciudad típica en una tarde de verano es 2.5°C más alto que en los alrededores rurales. Encontraron que la demanda pico de energía eléctrica sube de 2-4% por cada 1°C que sube después del umbral de los 15 a 20°C. Esto es, el uso adicional de aire acondicionado causado por el incremento de la temperatura en el aire es responsable del 5-10% en el incremento de la demanda pico de energía eléctrica en zonas urbanas.

La parte más baja de la atmósfera (capa límite), está directamente influenciada por el terreno, uso del suelo (áreas forestales, cosechas, urbanidad, etc.) y el ciclo diario que afecta la transferencia de calor, dispersión y de la contaminación, turbulencia y la meteorología en la ciudad y agricultura. (Jacob Jamey, 2016). El componente de temperatura de aire generalmente está subdividido en isla de calor a nivel copa de árboles e isla de calor de la capa límite y esta capa es accesible por los drones, ver Figura 2.4.

Existe una iniciativa en los Estados Unidos, llamada CLOUD-MAP (Collaboration Leading Operational UAS Development for Meteorology and Atmospheric Physics), es un equipo multidisciplinario enfocado en desarrollar e implementar sistemas aéreos no tripulados y su integración con sensores para las mediciones atmosféricas. Este experimento es la más grande iniciativa en utilizar los drones para estudiar el clima (Facility, 2017). El equipo está conformado por miembros de Oklahoma State University, University of Oklahoma,

University of Nebraska y University of Kentucky. Esta iniciativa empezó su operación a mediados del 2015 según (Jacob Jamey, 2016).

En Mexicali se han realizado estudios sobre islas de calor debido al crecimiento de la ciudad en una región desértica. En (Higuera & Cueto, 2014) se propone un modelo estadístico para estimar la isla de calor urbana en el aire. En (Casillas-Higuera *et al.*, 2014) se validó un modelo dinámico para la estimación de intensidad y hora en que ocurre la isla de calor urbana en base a los datos de cuatro estaciones meteorológicas ubicadas en zonas urbanas y suburbanas. Datos adicionales a través de drones podrían complementar las escasas fuentes de datos para la predicción del clima (Smith, 2017).



Figura 2.3: Pequeños drones en la meteorología de la capa límite (Smith, 2017)

2.3 Quadrirrotor

De acuerdo a (Luukkonen, 2011), un dron, también conocido como quadrirrotor, es un helicóptero con cuatro rotores que apuntan hacia arriba y están colo-

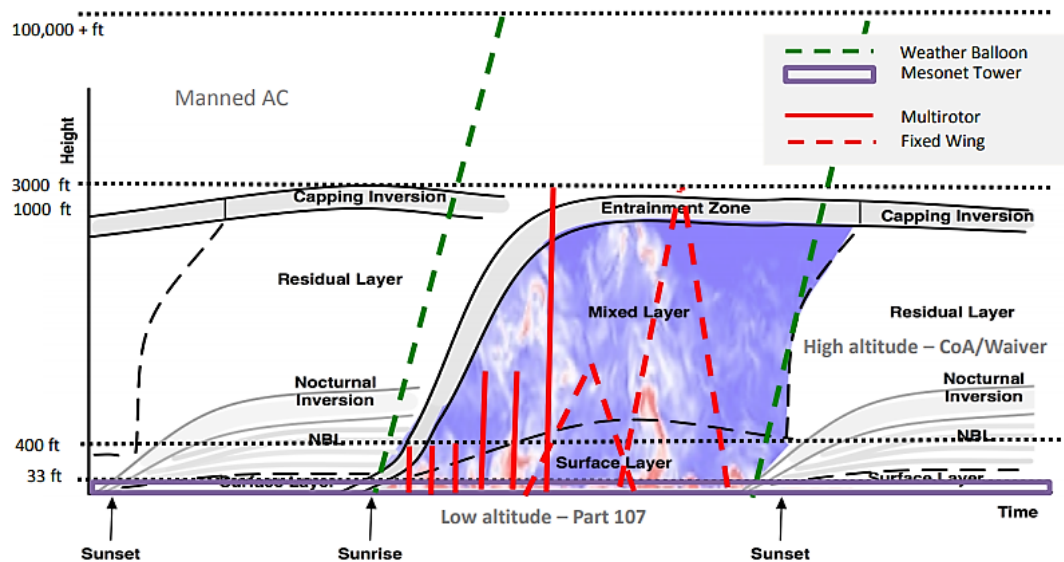


Figura 2.4: Pequeños drones en la meteorología de la capa límite (Smith, 2017)



Figura 2.5: Octacóptero Skyhawk (Shenzhen SkyhawkRC Technology Co., 2018)



Figura 2.6: Steady Drone Q4DX (akrcguy, 2014)



Figura 2.7: USAF RQ-4 Global Hawk (US-Air-Force, 2014)



Figura 2.8: USAF RQ-4 Global Hawk (US-Air-Force, 2014)



Figura 2.9: Dron profesional para vídeo filmación (DIEDRO, 2015)



Figura 2.10: Drones en eventos sociales (Ward, 2017)

cados en formación cuadrada equi-distantes al centro de masa de la aeronave. El quadrirrotor es controlado ajustando las velocidades angulares de cada uno de sus cuatro rotores. (Castillo *et al.*, 2007) dice que es un dron muy completo para experimentar sobre los fenómenos aerodinámicos que ocurren cuando realizan vuelo estacionario. El quadrirrotor a diferencia del helicóptero tradicional, tiene una estructura mecánica más simple y su modelo dinámico es el punto de partida para todos los estudios (Luukkonen, 2011). Diferentes métodos de control han sido investigados: controladores PID (proportional, integrative, derivative) (Bouabdallah *et al.*, 2004; Dikmen *et al.*, 2009; Zuo, 2010), control backstepping (Madani & Benallegue, 2006; Zemalache *et al.*, 2005), control H_∞ no-lineal (Raffo *et al.*, 2010), control LQR (Bouabdallah *et al.*, 2004), y control no-lineal con saturaciones anidadas (Castillo *et al.*, 2005; Escareno *et al.*, 2006). Los métodos de control requieren información precisa de las mediciones de posición y actitud provenientes de un giroscopio, un acelerómetro y otros dispositivos de medición como los receptores GPS, y los sensores sónicos y láser (He *et al.*, 2008; Martin & Salaün, 2010).

2.3.1 El Modelo Matemático del quadrirrotor

Como se mencionó anteriormente, el quadrirrotor es controlado al variar la velocidad angular de cada rotor. La fuerza producida por el motor i (f_i) es proporcional a la constante de ascenso (k) al cuadrado de la velocidad angular (w_i), esto es $f_i = kw_i^2$. Dado que cada motor gira en una dirección fija (al eje del rotor), la fuerza resultante f_i siempre es positiva (ver Figura 2.11).

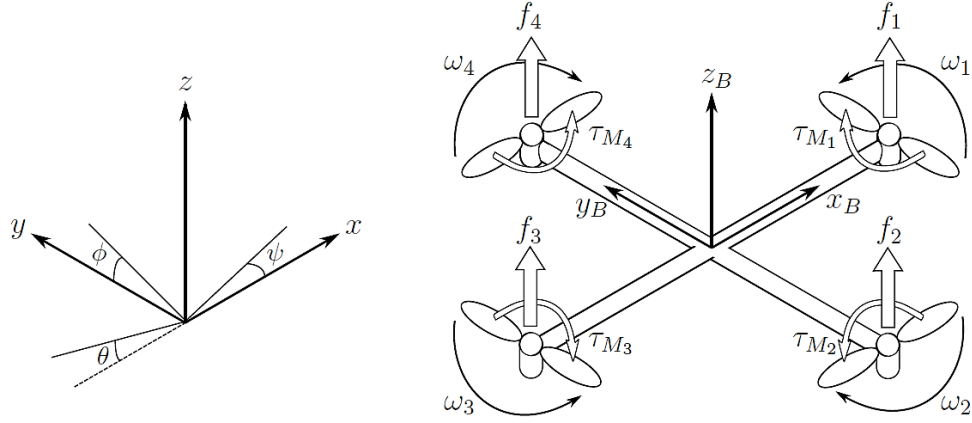


Figura 2.11: Las variables de Control del quadrrirrotor (en la izquierda, el sistema o marco inercial, en la derecha, el marco del chasis (Luukkonen, 2011)).

El vector de posición lineal (ξ) del quadrrirrotor se define en los ejes lineales del sistema inercial (x, y, z) (*inertial frame*). La actitud o vector de posición angular (η) se define en los ejes angulares del sistema inercial (θ, ϕ, ψ) de Euler. El ángulo para Pitch (θ) determina la rotación alrededor del eje y . El ángulo para Roll (ϕ) determina la rotación alrededor del eje x y el ángulo para Yaw (ψ) determina la rotación alrededor del eje z . El vector de posiciones lineales y angulares (q) está conformado por ξ y η .

$$\xi = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad \eta = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}, \quad q = \begin{bmatrix} \xi \\ \eta \end{bmatrix}. \quad (2.1)$$

El origen del chasis (*body frame*) se encuentra en el centro de la masa del quadrrirrotor. En el *body frame*, se determinan las velocidades lineales (V_B) y velocidades angulares (v).

$$V_B = \begin{bmatrix} v_{x,B} \\ v_{y,B} \\ v_{z,B} \end{bmatrix}, \quad v = \begin{bmatrix} p \\ q \\ r \end{bmatrix}. \quad (2.2)$$

La matriz de rotación del *body frame* al *inertial frame* es

$$R = \begin{bmatrix} C_\psi C_\theta & C_\psi S_\theta S_\phi - S_\psi C_\phi & C_\psi S_\theta C_\phi + S_\psi S_\phi \\ S_\psi C_\theta & S_\psi S_\theta S_\phi + C_\psi C_\phi & S_\psi S_\theta C_\phi - C_\psi S_\phi \\ -S_\theta & C_\theta S_\phi & C_\theta C_\phi \end{bmatrix}, \quad (2.3)$$

donde **seno de x (S_x)** = $\sin(x)$ y **coseno de x (C_x)** = $\cos(x)$. La **matriz de rotación (R)** es ortogonal por lo tanto, $R^{-1} = R^T$ la cual es la matriz de rotación del *inertial frame* al *body frame*.

La **matriz de transferencia para velocidades angulares (W_η)** del *inertial frame* al *body frame*, y del *body frame* al *inertial frame* is W_η^{-1} , como se muestra en

$$\dot{\eta} = W_\eta^{-1} v, \quad \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & S_\phi T_\theta & C_\phi T_\theta \\ 0 & C_\phi & -S_\phi \\ 0 & S_\phi/C_\theta & C_\phi/C_\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \quad (2.4)$$

$$v = W_\eta \dot{\eta}, \quad \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -S_\theta \\ 0 & C_\phi & C_\theta S_\phi \\ 0 & -S_\phi & C_\theta/C_\phi \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}$$

donde la **tangente de x (T_x)** = $\tan(x)$. La matriz W_η es invertible si $\theta \neq (2k-1)\phi/2, (k \in \mathbb{Z})$.

Considerando que el quadrirrotor tiene una estructura simétrica con los cuatro brazos alineados a sus ejes x e y , así tenemos que la matriz de inercia es la **I** donde $I_{xx} = I_{yy}$

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}. \quad (2.5)$$

La **velocidad angular del rotor i** (ω_i), crea la fuerza f_i en la dirección del eje del rotor. La velocidad angular y aceleración también crea un **torque del eje del rotor i** (τ_{M_i}).

$$f_i = k\omega_i^2, \quad \tau_{M_i} = d\omega_i^2 + I_M\dot{\omega}_i, \quad (2.6)$$

donde k , **constante de arrastre** (b) y **momento inercial del rotor** (I_M). Usualmente el efecto de la **aceleración angular** ($\dot{\omega}_i$) se considera muy pequeño, por lo que se omite.

La combinación de fuerzas de los rotores crea un **impulso** (T) en dirección del eje z del chasis. El **torque** (τ_B) está formado por el **torque de ϕ** (τ_ϕ), **torque de θ** (τ_θ) y **torque de ψ** (τ_ψ) en dirección de los correspondientes ángulos del *body frame*

$$T = \sum_{i=1}^4 f_i = k \sum_{i=1}^4 \omega_i^2, \quad T^B = \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix}, \quad (2.7)$$

$$\tau_B = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} lk(-\omega_2^2 + \omega_4^2) \\ lk(-\omega_1^2 + \omega_3^2) \\ \sum_{i=1}^4 \tau_{M_i} \end{bmatrix}, \quad (2.8)$$

donde la **distancia entre el rotor y el centro de la masa del quadrirrotor** (l). Las hélices frontal y trasera giran en sentido contrario a las manecillas del reloj. La izquierda y derecha giran en sentido de las manecillas del reloj. Esta configuración de pares opuestos evita una cola con rotor (la que se necesita en un helicóptero estándar).

De aquí se observa que el movimiento de *balanceo* (*roll*) se obtiene al decrementar la velocidad del segundo rotor e incrementar la velocidad del cuarto rotor, el quadrirrotor se moverá hacia la izquierda. Similarmente, el movimiento

de *inclinación* (*pitch*) se obtiene al decrementar la velocidad del primer rotor e incrementar la velocidad del tercer rotor, el quadrirrotor se moverá hacia el frente. El movimiento de *viraje* (*yaw*) se obtiene al incrementar las velocidades angulares de dos rotores opuestos y decrementar las velocidades de los otros dos.

La figura 2.12 muestra el modelo estructural en condiciones de vuelo estacionario, donde las hélices tienen la misma velocidad angular. (Bresciani, 2008). En color **negro** se presenta la estructura del quadrirrotor. El *body frame* fijo (por ser vuelo estacionario) se muestra en **verde** y en **azul** se representa la velocidad angular de las hélices como sigue: las flechas curvas representan el sentido de rotación y las flechas hacia arriba representan el valor de velocidad.

En el modelo de la figura 2.12 todas las hélices giran a la **velocidad de vuelo estacionario (Hovering)** (Ω_H) [rad s^{-1}] para contrarrestar la aceleración de la gravedad. Así que el quadrirrotor ejecuta un vuelo estacionario sin fuerzas o torques que lo muevan de su posición.

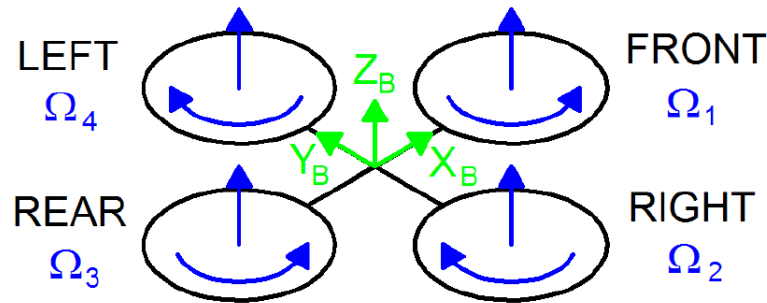


Figura 2.12: Modelo simplificado del quadrirrotor en vuelo estacionario (Bresciani, 2008)

De acuerdo a (Tedrake, 2009) un sistema de control sub-actuado es un sistema mecánico cuyas variables manipulables no permiten regular todos los grados de libertad del sistema (degrees of freedom, DOF). Un quadrirrotor tiene 6 DOF, equipado con 4 hélices no es posible alcanzar una posición deseada con todos los DOF, solamente 4 DOF por lo que quadrirrotor tiene un sistema de control sub-actuado (Bresciani, 2008). Sin embargo, gracias a su estructura, es muy fácil escoger las mejores 4 variables y entenderlas para hacer el controlador más fácil (Bresciani, 2008). Estas variables están relacionadas

a los cuatro movimientos básicos que permiten al quadrirrotor alcanzar cierta altura y actitud. Estos son los cuatro movimientos principales:

Throttle o Impulso [U_1] El **comando impulso (throttle)** (U_1) se genera al incrementar (o decrementar) todas las velocidades de las hélices en la misma cantidad. Este lleva a una fuerza vertical con respecto al marco del cuerpo que levanta o baja el quadrirrotor. Si el multi-rotor está en una posición horizontal, la dirección vertical del sistema inercial y del marco del cuerpo coinciden. De otra manera, el impulso proveído genera aceleraciones vertical y horizontal en el sistema inercial produciendo un **incremento de velocidad de la hélice** (Δ_A). La figura 2.13 muestra el comando *Impulso* en el bosquejo del quadrirrotor.

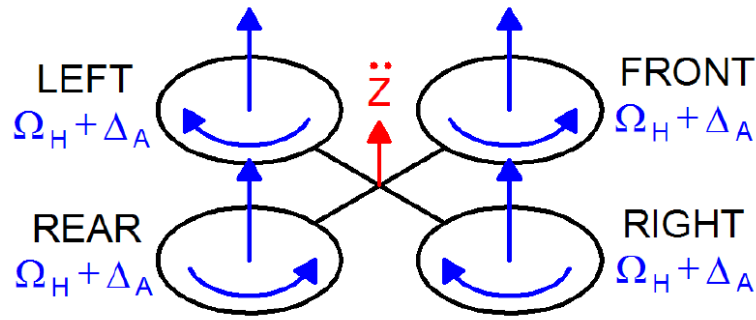


Figura 2.13: *Modelo simplificado del quadrirrotor en vuelo estacionario (Bresciani, 2008)*

En **azul** se especifica Ω_H , iguales para cada hélice. El Δ_A [rad s^{-1}] es una variable positiva la cuál representa un incremento respecto a la constante Ω_H . El Δ_A no puede tener un valor alto porque el modelo eventualmente será influenciado por fuertes variables no lineales o saturaciones.

Roll o vuelta [U_2] El **comando vuelta o movimiento lateral (roll)** (U_2) es generado a través del **decremento de velocidad de la hélice** (Δ_B) derecha y Δ_A en la hélice izquierda. Este lleva a un torque con respecto al eje x_B el cual hace que el quadrirrotor gire. Si se intercambian estos valores, se obtiene un giro contrario. El impulso vertical total es el mismo que en vuelo estacionario, por

lo tanto este comando lleva solamente a una aceleración del ángulo de *roll*. La figura 2.14 muestra el comando *roll* en el bosquejo del quadrrrotor. El efecto que produce es un desplazamiento lateral.

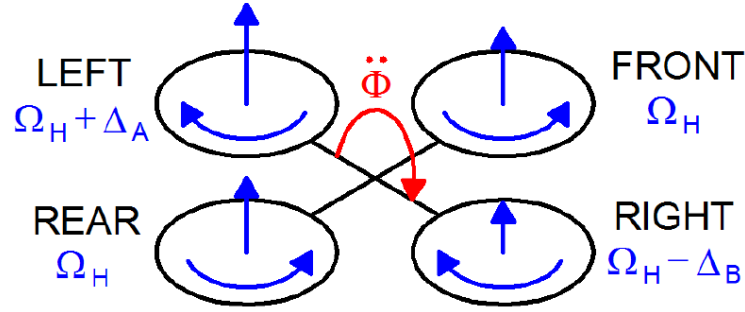


Figura 2.14: *Movimiento lateral en vuelo estacionario* (Bresciani, 2008)

Las variables Δ_A y Δ_B , se eligen para mantener el impulso vertical sin cambios. Se puede demostrar que para pequeños valores de Δ_A , $\Delta_B \approx \Delta_A$. Como en el caso anterior, estos valores no pueden ser muy altos porque el modelo eventualmente será influenciado por fuertes variables no lineales o saturaciones.

Pitch o inclinación [U_3] El comando inclinación o movimiento frente-atrás (*pitch*) (U_3) es muy similar al de *roll* y es generado a través del Δ_B en la hélice frontal y Δ_A en la hélice de atrás. Este lleva a un torque con respecto al eje y_B el cuál hace que el quadrrrotor gire. El impulso vertical total es el mismo que en vuelo estacionario, por lo tanto este comando lleva solamente a una aceleración del ángulo de *pitch*. La figura 2.15 muestra el comando *pitch* en el bosquejo del quadrrrotor. El efecto que produce es un desplazamiento frontal o trasero.

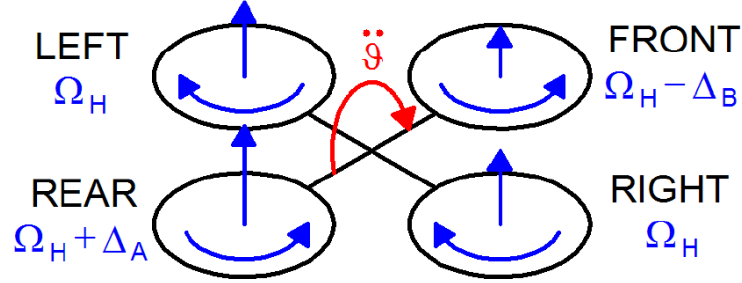


Figura 2.15: Movimiento frontal y trasero en vuelo estacionario (*Bresciani, 2008*)

Como en el caso anterior las variables Δ_A y Δ_B se eligen para mantener el impulso vertical sin cambios y no pueden ser muy grandes los valores. Para pequeños valores de Δ_A , $\Delta_B \approx \Delta_A$.

Yaw o giro [U_4] El comando giro (*yaw*) (U_4) es generado a través de incrementar (o decrementar) la velocidad de las hélices frontal-trasera y decrementar (o incrementar) las de izquierda-derecha. Este lleva a un torque con respecto al eje z_B el cual hace que el quadrirrotor gire. El movimiento de *Yaw* es generado gracias al hecho de que las hélices izquierda-derecha giran en sentido de las manecillas del reloj mientras que las frontal-trasera giran en sentido contrario a las manecillas del reloj. Entonces, cuando el torque vertical total no tiene balance, el quadrirrotor gira en su propio eje z_B . El impulso vertical total es el mismo que en vuelo estacionario, por lo tanto este comando lleva solamente a una aceleración del ángulo de *Yaw*. La figura 2.16 muestra el comando *Yaw* en el bosquejo del quadrirrotor. El efecto que se produce es una rotación del dron en el plano horizontal.

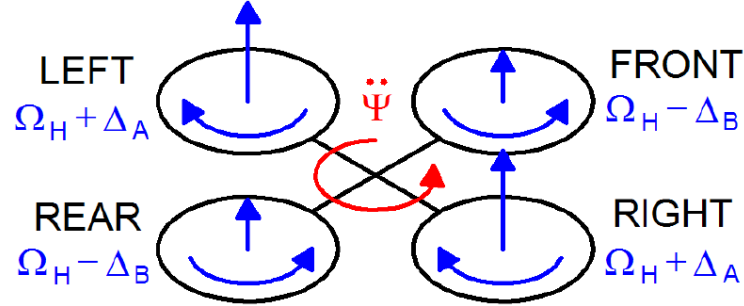


Figura 2.16: Giro entorno a z en vuelo estacionario (Bresciani, 2008)

Como en los dos casos anteriores, las variables Δ_A y Δ_B se eligen para mantener el impulso vertical sin cambios y no pueden ser muy grandes los valores. Para pequeños valores de Δ_A , $\Delta_B \approx \Delta_A$.

2.3.2 El modelo Newton-Euler

Esta sección provee la información específica del modelo de la arquitectura de un quadrirrotor empezando con la ecuación genérica de cuerpo rígido de 6 DOF. A continuación se definen dos marcos de referencia:

- el marco inercial de la tierra o sistema inercial (E -frame)
- el marco del cuerpo o chasis (B -frame)

Las ecuaciones de movimiento están mejor formuladas en el B -frame por las siguientes razones (Perez, 2006):

- La matriz de inercia es invariante en el tiempo.
- La simetría del quadrirrotor simplifica las ecuaciones.
- Las mediciones realizadas abordo son fácilmente convertidas al B -frame.
- Las fuerzas de control casi siempre están dadas para el B -frame.

La ecuación 2.9 describe la cinemática de un cuerpo rígido genérico de 6 DOF.

$$\dot{\xi} = J_{\Theta} v \quad (2.9)$$

donde $\dot{\xi}$ (vector de velocidad generalizado w.r.t. E-frame), v (vector de velocidad generalizado w.r.t. B-frame) y J_{Θ} (matriz generalizada).

El ξ (vector de posición generalizado w.r.t. E-frame) está compuesto por Γ^E (vector de posición lineal) y Θ^E (vector de posición angular) como se muestra en la Ecuación 2.10.

$$\xi = \begin{bmatrix} \Gamma^E & \Theta^E \end{bmatrix}^T = \begin{bmatrix} X & Y & Z & \phi & \theta & \psi \end{bmatrix}^T. \quad (2.10)$$

Similarmente, el v está compuesto por V^B (vector de velocidad lineal) y ω^B (vector de velocidad angular) como se muestra en 2.11.

$$v = \begin{bmatrix} V^B & \omega^B \end{bmatrix}^T = \begin{bmatrix} u & v & w & p & q & r \end{bmatrix}^T. \quad (2.11)$$

Además, J_{Θ} está compuesta de cuatro sub-matrices de acuerdo a la ecuación 2.12.

$$J_{\Theta} = \begin{bmatrix} R_{\Theta} & O_{3 \times 3} \\ O_{3 \times 3} & T_{\Theta} \end{bmatrix} \quad (2.12)$$

donde $O_{3 \times 3}$ (sub-matriz de ceros), $R_{\Theta}[-]$ (matriz de rotación) y T_{Θ} (matriz de transferencia o transformación) están definidas en las ecuaciones 2.3 y 2.4.

La dinámica de un cuerpo rígido genérico de 6 DOF toma en cuenta la m (masa del cuerpo) y su I (matriz de inercia) [$N \, m \, s^2$] que se define en 2.5 y con 2.6. La dinámica se describe en la ecuación 2.13.

$$\begin{bmatrix} m I_{3 \times 3} & O_{3 \times 3} \\ O_{3 \times 3} & I \end{bmatrix} \begin{bmatrix} \dot{V}^B \\ \dot{\omega}^B \end{bmatrix} + \begin{bmatrix} \omega^B \times & (m V^B) \\ \omega^B \times & (I \omega^B) \end{bmatrix} = \begin{bmatrix} F^B \\ \tau^B \end{bmatrix} \quad (2.13)$$

donde $I_{3 \times 3}$ (matriz identidad), $\dot{V}^B$ (vector de aceleración lineal w.r.t. *B-frame*) [$m s^2$] mientras que $\dot{\omega}^B$ (vector de aceleración angular w.r.t. *B-frame*) [$rad s^2$] y τ^B (vector de torques w.r.t. *B-frame*) [$N m$] definida en 2.8.

Un vector de fuerza generalizado (Λ) se define de acuerdo a la ecuación 2.14.

$$\Lambda = \begin{bmatrix} F^B & \tau^B \end{bmatrix}^T = \begin{bmatrix} F_x & F_y & F_z & \tau_x & \tau_y & \tau_z \end{bmatrix}^T \quad (2.14)$$

Por lo tanto, es posible re-escribir la ecuación 2.13 en forma de matrices (Perez, 2006).

$$M_B \dot{v} + C_B(v) v = \Lambda \quad (2.15)$$

donde $\dot{v}$ (vector de aceleración generalizado w.r.t. *B-frame*). M_B (matriz de inercia del sistema w.r.t. *B-frame*) y $C_B(v)$ (matriz Coriolis-centrípeta w.r.t. *B-frame*). La ecuación 2.16 muestra M_B .

$$M_B = \begin{bmatrix} m I_{3 \times 3} & O_{3 \times 3} \\ O_{3 \times 3} & I \end{bmatrix} = \begin{bmatrix} m & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 \\ 0 & 0 & m & 0 & 0 & 0 \\ 0 & 0 & 0 & I_{XX} & 0 & 0 \\ 0 & 0 & 0 & 0 & I_{YY} & 0 \\ 0 & 0 & 0 & 0 & 0 & I_{ZZ} \end{bmatrix} \quad (2.16)$$

Es fácil ver que M_B es diagonal y constante. La ecuación 2.17 muestra $C_B(v)$.

$$C_B(v) = \begin{bmatrix} O_{3 \times 3} & -m S(V^B) \\ O_{3 \times 3} & -S(I \omega^B) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & m w & -m v \\ 0 & 0 & 0 & -m w & 0 & m u \\ 0 & 0 & 0 & m v & -m u & 0 \\ 0 & 0 & 0 & 0 & I_{ZZ} r & -I_{YY} q \\ 0 & 0 & 0 & -I_{ZZ} r & 0 & I_{XX} p \\ 0 & 0 & 0 & I_{YY} q & -I_{XX} p & 0 \end{bmatrix} \quad (2.17)$$

En la ecuación anterior, el $S(\cdot)$ (operador anti-simétrico) se adoptó. Para un K - (vector genérico de tres dimensiones), la $S(k)$ (matriz anti-simétrica de k) se define según la ecuación 2.18.

$$S(K) = -S^T(K) = \begin{bmatrix} 0 & -K_3 & K_1 \\ K_3 & 0 & -K_1 \\ -K_2 & K_1 & 0 \end{bmatrix}, \quad K = \begin{bmatrix} K_1 \\ K_2 \\ K_3 \end{bmatrix} \quad (2.18)$$

La ecuación 2.15 es genérica y válida para todo cuerpo rígido que obedece a las simplificaciones previamente hechas. Sin embargo, para este estudio se utilizó para modelar el quadrirrotor, por lo tanto el último vector contiene información específica acerca de su dinámica.

Λ puede dividirse en tres componentes de acuerdo a la naturaleza de las contribuciones del quadrirrotor:

La primera contribución es $G_B(\xi)$ (vector gravitacional) proveniente de g (aceleración de la gravedad de la tierra) [$m s^{-2}$]. Se entiende que afecta solamente a las ecuaciones lineales y no las angulares debido a que es una fuerza y no un torque. La ecuación 2.19 muestra las transformaciones para obtener $G_B(\xi)$.

$$G_B(\xi) = \begin{bmatrix} F_G^B \\ 0_{3 \times 1} \end{bmatrix} = \begin{bmatrix} R_\Theta^{-1} F_G^E \\ 0_{3 \times 1} \end{bmatrix} = \begin{bmatrix} R_\Theta^T \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} \\ 0_{3 \times 1} \end{bmatrix} = \begin{bmatrix} mg s_\theta \\ -mg c_\theta s_\phi \\ -mg c_\theta s_\phi \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (2.19)$$

Donde F_G^B (vector de la fuerza gravitacional w.r.t. *B-frame*) [N] es el y F_G^E (vector de la fuerza gravitacional w.r.t. *E-frame*) [N]. $0_{3 \times 1}$ (vector vertical de ceros) [-]. Además, dado que $R_\Theta[-]$ es una matriz ortogonal, su inversa $R_\Theta[-]^{-1}$ es igual a su traspuesta $R_\Theta[-]^T$.

La *segunda contribución* toma en cuenta los efectos giroscópicos producidos por la rotación de la hélice. Dado que dos de ellas están rotando en sentido de las manecillas del reloj y las otras dos en sentido contrario, hay en general un desbalance cuando la suma algebraica de las velocidades de los rotores es diferente a cero. Si a esto le sumamos las pequeñas diferencias de *roll* y *pitch*, el quadrirrotor experimentará un torque giroscópico de acuerdo a la ecuación 2.20.

$$O_B(\nu)\Omega = \begin{bmatrix} 0_{3 \times 1} \\ -\sum_{k=1}^4 J_{TP} \left(\omega^B \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right) (-1)^k \Omega_k \end{bmatrix} =$$

$$\begin{bmatrix} 0_{3 \times 1} \\ J_{TP} \begin{bmatrix} -q \\ p \\ 0 \end{bmatrix} \end{bmatrix} \Omega = J_{TP} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ q & -q & q & -q \\ -p & p & -p & p \\ 0 & 0 & 0 & 0 \end{bmatrix} \Omega \quad (2.20)$$

donde $O_B(\nu)$ (matriz de hélices giroscópica) [+] y J_{TP} (momento de inercia rotacional total) [Nms^2] alrededor del eje de la hélice (calculada más adelante). Es fácil ver que los efectos giroscópicos producidos por la rotación de la hélice están relacionados a ecuaciones angulares y no a ecuaciones lineales.

La ecuación 2.21 usada en 2.20 define Ω_Σ (velocidad total de las hélices) [$rad\ s^{-1}$] y Ω (vector de velocidad de las hélices) [$rad\ s^{-1}$]

$$\Omega_\Sigma = -\Omega_1 + \Omega_2 - \Omega_3 + \Omega_4 \quad \Omega = \begin{bmatrix} \Omega_1 \\ \Omega_2 \\ \Omega_3 \\ \Omega_4 \end{bmatrix} \quad (2.21)$$

donde Ω_1 (velocidad de la hélice frontal), Ω_2 (velocidad de la hélice derecha), Ω_3 (velocidad de la hélice trasera) y Ω_4 (velocidad de la hélice izquierda).

La *tercera contribución* toma en cuenta las fuerzas y torques producidos directamente por las entradas principales de movimiento. Desde el punto de

vista de la aerodinámica, fuerzas y torques son proporcionales al cuadrado de la velocidad de las hélices. Por lo tanto, E_B (matriz de movimiento) [+] es multiplicada por Ω^2 para obtener $U_B(\Omega)$ (vector de movimiento) [+]. Y se derivan las contribuciones aerodinámicas b (constante de impulso) y arrastre b .

La ecuación 2.22 muestra la acción del vector de movimiento en la dinámica del quadrirrotor.

$$U_B(\Omega) = E_B \Omega^2 = \begin{bmatrix} 0 \\ 0 \\ U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \\ b l (\Omega_4^2 - \Omega_2^2) \\ b l (\Omega_3^2 - \Omega_1^2) \\ d(\Omega_2^2 + \Omega_4^2 - \Omega_1^2 - \Omega_3^2) \end{bmatrix} \quad (2.22)$$

recordando que l es la distancia quadrirrotor-hélice. U_1 , U_2 , U_3 y U_4 son los elementos del vector de movimiento de la ecuación 2.22. Su relación con la velocidad de las hélices viene del cálculo aerodinámico, el cual no se describirá en este trabajo. La expresión del torque producido por U_4 se simplifica descartando $\dot{\Omega}$ (vector de aceleración de las hélices). Por lo tanto, todos los movimientos tienen una expresión similar y son más fáciles de controlar.

Como se ha indicado anteriormente, (y mostrado en la ecuación anterior), es posible identificar una matriz constante E_B que multiplicada por el cuadrado de las velocidades de las hélices Ω^2 produce el vector de movimiento $U_B(\Omega)$. La ecuación 2.23 muestra esa E_B .

$$E_B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ b & b & b & b \\ 0 & -bl & 0 & bl \\ -bl & 0 & bl & 0 \\ -d & d & -d & d \end{bmatrix} \quad (2.23)$$

de la ecuación 2.15 es posible describir la dinámica del quadrirrotor considerando estas *tres contribuciones* de acuerdo a la ecuación 2.24.

$$M_B \dot{\nu} + C_B(\nu) \nu = G_B(\xi) + O_B(\nu) \Omega + E_B \Omega^2 \quad (2.24)$$

manipulando la ecuación 2.24 es posible despejar la derivada del vector generalizado de velocidad $\dot{\nu}$.

$$\dot{\nu} = M_B^{-1} (-C_B(\nu) \nu + G_B(\xi) + O_B(\nu) \Omega + E_B \Omega^2) \quad (2.25)$$

La ecuación 2.26 muestra la expresión anterior como sistema de ecuaciones y no en forma matricial.

$$\left\{ \begin{array}{l} \dot{u} = (v r - w q) + g s_\theta \\ \dot{v} = (w p - u r) - g c_\theta s_\phi \\ \dot{w} = (u q - v p) - g c_\theta s_\phi + \frac{U_1}{m} \\ \dot{p} = \frac{I_{YY} - I_{ZZ}}{I_{XX}} q r - \frac{J_{TP}}{I_{XX}} q \Omega + \frac{U_2}{I_{XX}} \\ \dot{q} = \frac{I_{ZZ} - I_{XX}}{I_{YY}} p r + \frac{J_{TP}}{I_{YY}} p \Omega + \frac{U_3}{I_{YY}} \\ \dot{r} = \frac{I_{XX} - I_{YY}}{I_{ZZ}} p q + \frac{U_4}{I_{ZZ}} \end{array} \right. \quad (2.26)$$

donde las entradas de velocidad de las hélices están dadas por la ecuación 2.27.

$$\left\{ \begin{array}{l} U_1 = b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \\ U_2 = b l (\Omega_4^2 - \Omega_2^2) \\ U_3 = b l (\Omega_3^2 - \Omega_1^2) \\ U_4 = d(\Omega_2^2 + \Omega_4^2 - \Omega_1^2 - \Omega_3^2) \\ \Omega_\Sigma = -\Omega_1 + \Omega_2 - \Omega_3 + \Omega_4 \end{array} \right. \quad (2.27)$$

El sistema dinámico del quadrrrotor en la ecuación 2.26 es w.r.t. *B-frame*. Como se indicó anteriormente, esta referencia es ampliamente utilizada en ecuaciones de cuerpos sólidos de 6 DOF. Sin embargo, en este caso puede ser útil expresar la dinámica con respecto a un *sistema híbrido* compuesto de ecuaciones lineales w.r.t. *E-frame* y a ecuaciones angulares w.r.t. *B-frame*. Por lo tanto las siguientes ecuaciones serán expresadas en un nuevo marco de referencia híbrido que llamaremos *H-frame*. Esta nueva referencia es adoptada porque es fácil de expresar las dinámicas combinadas con el control (en particular para la posición vertical en el marco de referencia inercial de la tierra). La ecuación 2.28 muestra el **vector generalizado de velocidad w.r.t. *H-frame* (ζ)**.

$$\zeta = \begin{bmatrix} \dot{\Gamma}^E & \omega^B \end{bmatrix}^T = \begin{bmatrix} \dot{X} & \dot{Y} & \dot{Z} & p & q & r \end{bmatrix}^T \quad (2.28)$$

Las dinámicas del sistema en el *H-frame* pueden ser re-escritas en forma matricial de acuerdo a la ecuación 2.29.

$$M_H \ddot{\zeta} + C_H(\zeta) \dot{\zeta} = G_H + O_H(\zeta) \Omega + E_H(\xi) \Omega^2 \quad (2.29)$$

donde $\ddot{\zeta}$ (**vector generalizado de aceleración del quadrrrotor w.r.t. *H-frame***). Y siguiendo con las definiciones de todas las matrices y vectores utilizados en la ecuación 2.29. M_H (**matriz de inercia del sistema w.r.t. *H-frame***) es igual a M_B definidas en las ecuaciones 2.16 y 2.30.

$$M_H = M_B = \begin{bmatrix} m I_{3 \times 3} & O_{3 \times 3} \\ O_{3 \times 3} & I \end{bmatrix} = \begin{bmatrix} m & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 \\ 0 & 0 & m & 0 & 0 & 0 \\ 0 & 0 & 0 & I_{XX} & 0 & 0 \\ 0 & 0 & 0 & 0 & I_{YY} & 0 \\ 0 & 0 & 0 & 0 & 0 & I_{ZZ} \end{bmatrix} \quad (2.30)$$

Por el contrario, la [matriz Coriolis-centripeta w.r.t. \$H\$ -frame \(\$C_H\(\zeta\)\$ \)](#) no es igual a $C_B(\nu)$, que se define en la ecuación 2.31.

$$C_H(\zeta) = \begin{bmatrix} 0_{3 \times 3} & O_{3 \times 3} \\ 0_{3 \times 3} & -S(I \omega^B) \end{bmatrix} = \begin{bmatrix} m & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 \\ 0 & 0 & m & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & I_{ZZ} r & -I_{YY} q \\ 0 & 0 & 0 & -I_{ZZ} r & 0 & I_{XX} p \\ 0 & 0 & 0 & I_{YY} q & -I_{XX} p & 0 \end{bmatrix} \quad (2.31)$$

El [vector gravitacional w.r.t. \$H\$ -frame \(\$G_H\$ \)](#) se define en la ecuación 2.32. Puede verse que afecta a las tres ecuaciones lineales en vez de solamente la tercer ecuación como en el caso previo.

$$G_H = \begin{bmatrix} F_G^E \\ 0_{3 \times 1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -mg \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (2.32)$$

Los efectos giroscópicos producidos por la rotación de la hélice no varían porque solamente afecta a las ecuaciones angulares en relación al *B-frame*. Entonces, la **matriz giroscópica de las hélices w.r.t. *H-frame* ($O_H(\zeta)$)** se define de acuerdo a las ecuaciones 2.20 y 2.33.

$$O_H(\zeta)\Omega = O_B(v)\Omega = \begin{bmatrix} 0_{3 \times 1} \\ J_{TP} \begin{bmatrix} -q \\ p \\ 0 \end{bmatrix} \end{bmatrix} \Omega = J_{TP} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ q & -q & q & -q \\ -p & p & -p & p \\ 0 & 0 & 0 & 0 \end{bmatrix} \Omega \quad (2.33)$$

La **matriz de movimiento w.r.t. *H-frame* ($E_H(\xi)$)** es diferente de E_B porque la entrada U_1 afecta las tres ecuaciones lineales por medio de $R_\Theta[-]$. La multiplicación entre la matriz de movimiento y el cuadrado del vector de velocidad de las hélices se muestra en la ecuación 2.34.

$$E_H(\xi)\Omega^2 = \begin{bmatrix} R_\Theta & 0_{3 \times 3} \\ 0_{3 \times 3} & I_{3 \times 3} \end{bmatrix} E_B\Omega^2 = \begin{bmatrix} (S_\psi S_\phi + C_\psi S_\theta C_\phi) U_1 \\ (C_\psi S_\phi + S_\psi S_\theta C_\phi) U_1 \\ (C_\theta C_\phi) U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} \quad (2.34)$$

Es posible despejar el **vector de aceleración generalizado w.r.t. *H-frame* ($\ddot{\zeta}$)**.

$$\ddot{\zeta} = M_H^{-1} (-C_H(\zeta)\dot{\zeta} + G_H + O_H(\zeta)\Omega + E_H(\xi)\Omega^2) \quad (2.35)$$

La ecuación 2.36 muestra la expresión anterior, no en su forma matricial, pero como un sistema de ecuaciones.

$$\left\{ \begin{array}{l} \ddot{X} = (S_\psi S_\phi + C_\psi S_\theta C_\phi) \frac{U_1}{m} \\ \ddot{Y} = (-C_\psi S_\phi + S_\psi S_\theta C_\phi) \frac{U_1}{m} \\ \ddot{Z} = -g + C_\theta C_\phi + \frac{U_1}{m} \\ \dot{p} = \frac{I_{YY} - I_{ZZ}}{I_{XX}} q r - \frac{J_{TP}}{I_{XX}} q \Omega + \frac{U_2}{I_{XX}} \\ \dot{q} = \frac{I_{ZZ} - I_{XX}}{I_{YY}} p r + \frac{J_{TP}}{I_{YY}} p \Omega + \frac{U_3}{I_{YY}} \\ \dot{r} = \frac{I_{XX} - I_{YY}}{I_{ZZ}} p q + \frac{U_4}{I_{ZZ}} \end{array} \right. \quad (2.36)$$

donde las entradas de velocidad de las hélices son las mismas que en el sistema w.r.t. *B-frame* y dadas por la ecuación 2.27.

2.4 El filtro Kalman

2.4.1 Introducción

En 1960, R.E. Kalman publicó su famoso artículo describiendo una solución recursiva al problema de filtrado de datos discretos lineales (Kalman, 1960). Desde entonces, debido a los grandes avances en la computación, el *Filtro Kalman* ha sido tema de extensas investigaciones y aplicaciones, particularmente en el área de navegación autónoma o asistida (Bishop et al., 2001).

¿Que es el Filtro Kalman? Un filtro Kalman es simplemente un *algoritmo recursivo óptimo de procesamiento de datos* (Maybeck, 1982). También define el termino *óptimo* dependiente del criterio utilizado para evaluar su desempeño. Muestra que bajo ciertos supuestos, el filtro Kalman es óptimo con respecto a (virtualmente) cualquier criterio que tenga sentido. Un aspecto de esta optimalidad es que el filtro Kalman incorpora toda la información que pueda ser proveída. Procesa todas las mediciones, sin importar su precisión, para estimar el valor actual de las variables de interés, con el uso de: (1) conocimiento

del sistema y las mediciones de la dinámica del dispositivo, (2) la descripción estadística de los ruidos del sistema, errores de medición, y la incertidumbre de los modelos de la dinámica, y (3) cualquier información disponible de las condiciones iniciales de las variables de interés. Por ejemplo, para determinar la velocidad de una aeronave, uno podría usar un radar Doppler, o las indicaciones de velocidad de un sistema inercial de navegación, o la presión de pitot y estática e información relativa del viento en un sistema aéreo de datos. En vez de ignorar cualquiera de estas salidas, un filtro Kalman se podría construir para combinar todos estos datos y el conocimiento de la dinámica de sus sistemas para generar la mejor estimación global de la velocidad (Maybeck, 1982).

2.4.2 Estimación del Proceso

El filtro Kalman apunta al problema general de intentar estimar el *estado de un proceso controlado por tiempo discreto k* (x_k) ($x_k \in \mathbb{R}^n$) que es gobernado por la ecuación estocástica lineal diferencial

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1}, \quad (2.37)$$

con una *medición en el tiempo discreto k* (z) ($z_k \in \mathbb{R}^m$), esto es

$$z_k = Hx_k + v_k. \quad (2.38)$$

El *ruido del proceso* (w_k) y el *ruido de la medición* (v_k) se asumen independientes entre si, w_k es ruido blanco y v_k está normalmente distribuida.

$$p(k) \sim N(0, Q), \quad (2.39)$$

$$p(v) \sim N(0, R). \quad (2.40)$$

En la práctica, la *matriz de covarianza del ruido de proceso* (Q) y la *matriz de covarianza del ruido de medición* (R) pueden cambiar con cada paso o medición, sin embargo asumiremos que son constantes.

La **matriz de $n \times n$ (A)** en la ecuación diferencial 2.37 relaciona el estado en el **tiempo previo ($k - 1$)** al estado en el **tiempo actual (k)**, en la ausencia de la función principal o el ruido de proceso. En la práctica, A puede cambiar con cada paso de tiempo, pero aquí asumiremos que es constante. La **matriz de $n \times l$ (B)** relaciona la entrada de control opcional $u \in \mathbb{R}^l$ a la variable de estado x . La matriz H de $m \times n$ en la ecuación de medición 2.38 relaciona el estado a la medición z_k . En la práctica H puede cambiar con cada paso de tiempo o medición, pero aquí se asume constante.

2.4.3 El origen Computacional del Filtro

Definimos $\hat{x}_k^- \in \mathbb{R}^n$ (notar el símbolo de menos (-) super-escrito) para ser nuestra estimación de estado *a priori* en el paso k dado el conocimiento del proceso previo al paso k , y $\hat{x}_k \in \mathbb{R}^n$ para ser nuestro estado estimado *a posteriori* en el paso k dada la medición z_k . Podemos entonces definir los errores estimados *a priori* y *a posteriori* como sigue

$$e_k^- \equiv x_k - \hat{x}_k^-, \quad y \quad (2.41)$$

$$e_k \equiv x_k - \hat{x}_k. \quad (2.42)$$

Entonces la covarianza *a priori* del error estimado es

$$P_k^- = E[e_k^- e_k^{-T}], \quad (2.43)$$

y la covarianza *a posteriori* del error estimado es

$$P_k = E[e_k e_k^T]. \quad (2.44)$$

Para obtener las ecuaciones del filtro Kalman, empezamos encontrando una ecuación que calcule un estado estimado *a posteriori* $\hat{x}_k$ como una combinación lineal de un estimado *a priori* $\hat{x}_k^-$ y una diferencia con cierto peso entre la medición actual z_k y la medición de predicción $H\hat{x}_k^-$ como se muestra en la ecuación 2.45.

$$\hat{x}_k = \hat{x}_k^- + K(z_k - H\hat{x}_k^-) \quad (2.45)$$

La diferencia $(z_k - H\hat{x}_k^-)$ en 2.45 se llama medición de *innovación*, por el *residuo*. El residuo refleja la discrepancia entre la medición predicha $H\hat{x}_k^-$ y la medición actual z_k . Un residuo de cero significa que las dos mediciones están en completo acuerdo.

La matriz K de $n \times m$ en 2.45 es la *ganancia* o *factor de ganancia* (*blending factor*) que minimiza la covarianza de error *a posteriori* 2.44. Esta minimización puede realizarse primeramente sustituyendo 2.45 en la definición de e_k , y luego sustituyendo eso en 2.44, y realizando las demás operaciones algebraicas necesarias para despejar K resultando en una forma minimizada de 2.44 como sigue

$$K_k = P_k^- H^T (H P_k^- H^T + R)^{-1} \quad (2.46)$$

$$= \frac{P_k^- H^T}{H P_k^- H^T + R} \quad (2.47)$$

En 2.46 podemos ver que si la covarianza del error de medición R tiende a cero, la ganancia K pesa más en el residuo. Específicamente,

$$\lim_{R_k \rightarrow 0} K_k = H^{-1}.$$

Por otro lado, a medida que la covarianza del error estimado *a priori* P_k^- tiende a cero, la ganancia K pesa menos en el residuo, específicamente,

$$\lim_{P_k^- \rightarrow 0} K_k = 0.$$

Otra forma de interpretar el peso de K es que si la covarianza del error de medición R tiende a cero, la medición actual z_k es más y más confiable, mientras que la medición predicha $H\hat{x}_k^-$ es menos y menos confiable. Por otro lado, a medida que la covarianza *a priori* del error estimado P_k^- tiende a cero, la medición actual z_k es menos y menos confiable, mientras que la medición predicha $H\hat{x}_k^-$ es más y más confiable.

2.4.4 El Algoritmo del Filtro de Kalman Discreto

El filtro de Kalman estima un proceso utilizando una forma de control de retroalimentación: el filtro estima el estado del proceso en un tiempo dado y entonces obtiene retroalimentación en la forma de mediciones (con ruido). Como tal, las ecuaciones del filtro de Kalman se dividen en dos grupos: ecuaciones de *Actualización del tiempo* (“Predicción”) y ecuaciones de *Actualización de la Observación* (“Corrección”). Las ecuaciones de predicción son las responsables de proyectar (en el tiempo) el estado actual y las estimaciones de error de covarianza para obtener las estimaciones *a priori* para el siguiente paso en el tiempo. Las ecuaciones de corrección son las responsables de la retroalimentación -ej. incorporar una nueva medición en la estimación *a priori* para obtener una estimación *a posteriori* mejorada.

Así, el algoritmo de estimación se asemeja al algoritmo predictivo-correctivo para resolver problemas numéricos como se muestra en la figura 2.17.

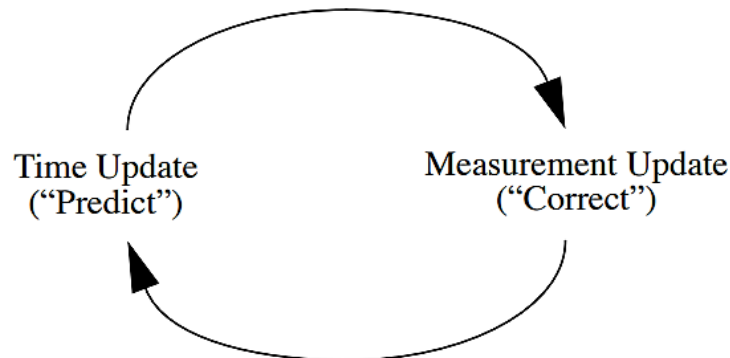


Figura 2.17: Ciclo del filtro Kalman discreto. La *predicción* proyecta la estimación del estado actual en el tiempo. La *corrección* ajusta la estimación proyectada por una medición actual en ese tiempo. (Bishop *et al.*, 2001)

Las ecuaciones para la predicción y corrección se presentan en la Tabla 2.1 y la Tabla 2.2.

Tabla 2.1: Ecuaciones de predicción de filtro Kalman discreto

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1} \quad (2.48)$$

$$P_k^- = AP_{k-1}A^T + Q \quad (2.49)$$

Observe otra vez como las ecuaciones de predicción en la Tabla 2.1 proyectan el estado y la covarianza estimada del paso $k - 1$ al paso k . A y B vienen de 2.37, mientras que Q viene de 2.39.

Tabla 2.2: Ecuaciones de corrección de filtro Kalman discreto

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} \quad (2.50)$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \quad (2.51)$$

$$P_k = (I - K_k H)P_k^- \quad (2.52)$$

La primera en el proceso de corrección, es calcular la ganancia Kalman, K_k . Note que la ecuación dada en 2.50 es la misma que 2.46. El siguiente paso es realizar una medición actual del proceso para obtener z_k , y entonces generar un estado estimado *a posteriori* incorporándola en 2.51. Vuelva a notar que 2.51 es igual a 2.45. El paso final es obtener una covarianza de error estimada *a posteriori* por 2.52.

Después de cada par predicción-corrección, el proceso se repite con la estimación anterior *a posteriori* usada para proyectar o predecir la nueva estimación *a priori*. La naturaleza recursiva de este algoritmo es una de las características más atractivas del filtro Kalman - realiza implementaciones prácticas mucho más factibles que (por ejemplo) una implementación de un filtro Wiener (Brown & Hwang, 1997) el cual es diseñado para operar todos los datos

directamente para cada estimación. El filtro Kalman en su lugar, condiciona recursivamente la estimación actual en todas las mediciones pasadas. La figura 2.18 ofrece una imagen de la operación del filtro, combinando el diagrama de alto-nivel de la figura 2.17 con las ecuaciones de las tablas 2.1 y 2.2.

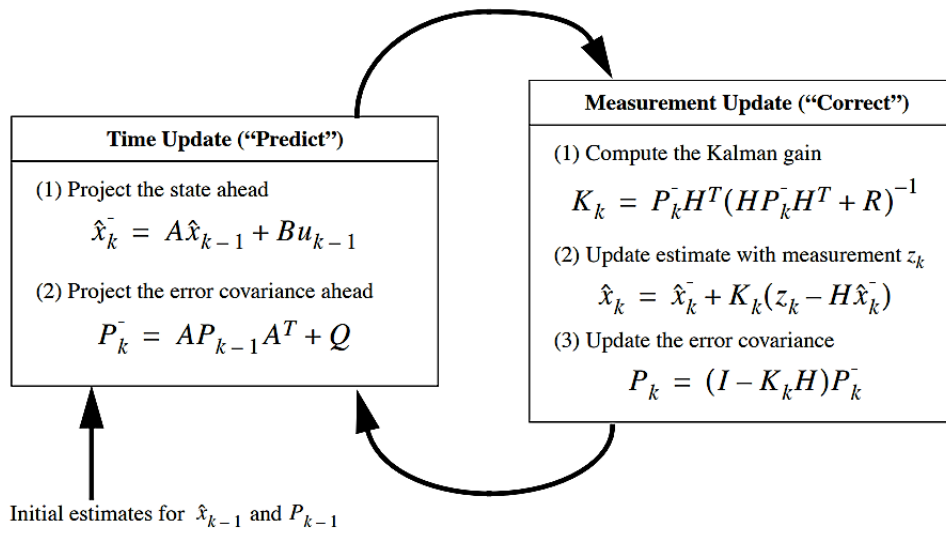


Figura 2.18: Imagen completa de la operación del filtro Kalman, combinando el diagrama de alto-nivel de la figura 2.17 con las ecuaciones de las tablas 2.1 y 2.2 (Bishop *et al.*, 2001).

2.4.5 Parámetros del filtro y ajustes finos

En la implementación del filtro, la medición de la covarianza del ruido R es usualmente medida antes de ejecutar el filtro. Medir la covarianza R es generalmente posible porque de cualquier forma, se tiene que medir el proceso (mientras se ejecuta el filtro) así que deberíamos ser capaces de tomar muestras *off-line* para determinar la varianza del ruido de mediciones.

La determinación de la covarianza de ruido del proceso Q es generalmente más difícil debido a que no tenemos la habilidad de observar directamente el proceso que estamos estimando. Algunas veces, un modelo de proceso simple, puede producir resultados aceptables si se "inyecta" suficiente incertidumbre en el proceso vía la selección de Q . Ciertamente uno esperaría que las mediciones del proceso sean confiables.

En cualquier caso, se tenga o no una base racional para la selección de parámetros, a menudo un desempeño superior del filtro (estadísticamente hablando) se puede obtener *ajustando* los parámetros del filtro Q y R . El ajuste normalmente se realiza “off-line”, con la ayuda de otro filtro Kalman en el proceso generalmente referido como *identificación del sistema*.

En conclusión, notamos que bajo ciertas condiciones donde Q y R son de hecho constantes, la covarianza estimada del error P_k y la ganancia Kalman K_k se estabilizarán rápidamente y se mantendrán constantes. Si este es el caso, estos parámetros pueden ser pre-procesados ejecutando el filtro “off-line”, o por ejemplo, determinando el valor del estado estable P_k descrito en (Simon, 2001).

2.5 Sistema de Posicionamiento Global

El sistema de posicionamiento global (Global Positioning System - GPS) es un sistema para determinar la posición de un usuario en el *espacio*, y como tal es ampliamente utilizado en la navegación - y es de libre acceso. Dado que GPS se comunica vía ondas de radio las cuales viajan a una velocidad conocida (velocidad de la luz), GPS también es un medio para determinar la posición de un usuario en determinado *tiempo* (Moore & Crossley, 1999). El GPS está compuesto de tres partes principales, referidas como segmentos:

1. Satélites en órbita (space segment)
2. Estaciones terrestres de control y monitoreo, y (control segment)
3. Los receptores GPS de los usuarios. (user segment)

Actualmente, existen 24 satélites en la constelación GPS. A diferencia de los satélites de televisión, los cuales siempre se encuentran en la misma posición orbital de la tierra (geo-síncrono), los satélites GPS están en constante movimiento, están dispuestos en seis planos orbitales de aproximadamente 12 horas los cuales se encuentran en órbita a una altura de 10,898 millas náuticas (Moore & Crossley, 1999), ver figura 2.19. Desde cualquier posición en la superficie de la tierra, hay suficientes satélites *a la vista* para proveer un servicio

de ubicación y tiempo. A pesar de esto, la disponibilidad de los satélites no siempre es del 100% (Moore & Crossley, 1999).

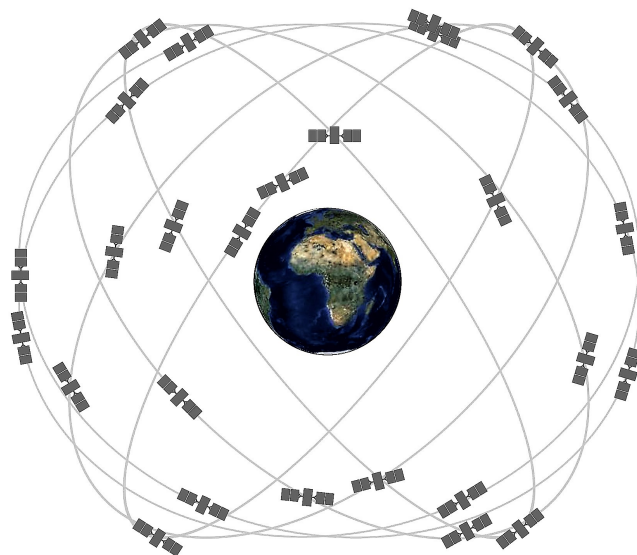


Figura 2.19: Constelación de satélites GPS (GPS.gov, 2018)

Los satélites GPS transmiten una señal de alta-frecuencia que contiene paquetes de información con la hora precisa a la cual se transmitieron estos paquetes. Los receptores recogen esta señal y utilizan un sistema para triangular y calcular la ubicación, comparando la diferencia de tiempo entre transmisión y recepción de cada paquete, de este modo se calcula la distancia a cada satélite. A medida que el receptor (del usuario) se mueve, las distancias de los satélites cambian, generando una pequeña diferencia en el tiempo, la cual se utiliza para actualizar la ubicación (ESDS, 2013).

El receptor GPS obtiene la señal de tres satélites para calcular una posición (**latitud, longitud**) y con la señal de al menos un satélite más, se puede calcular la altitud, obteniendo una vista en 3D (ESDS, 2013).

2.5.1 Estándar NMEA

La Asociación Nacional de Electrónica Marítima (National Marine Electronics Association - NMEA) ha desarrollado una especificación que define la interfase

entre varios equipos de electrónica marina. El estándar permite a la electrónica marina enviar información a computadoras y a otros equipos marinos. La comunicación del receptor GPS está definida dentro de estas especificaciones. Muchos programas para computadora que proveen información de posición en tiempo real, están configurados para el formato NMEA. Estos datos incluyen la posición, velocidad y tiempo procesados por el receptor GPS. La idea de NMEA es el envío de una línea de datos llamada *instrucción*. Hay instrucciones estándares para cada categoría de dispositivos. Todas las instrucciones estándar tienen dos letras para el prefijo que define el dispositivo que utiliza esa instrucción. Para los receptores GPS el prefijo es **GP**, el cual es seguido de una secuencia de tres letras que define el contenido de la instrucción. Además, NMEA permite que fabricantes de equipo GPS puedan definir sus propias instrucciones. Todas las instrucciones propietarias empiezan con la letra **P** y es seguida de tres letras que identifican el fabricante de esta instrucción. Por ejemplo, una instrucción de la marca Garmin empezaría con **PGRM** y una de la marca Magellan empezaría con **PMGN** (DePriest, 2013).

2.5.2 Formato del mensaje NMEA

Para entender la estructura de un mensaje NMEA, examinaremos el mensaje popular \$GPGGA. Este mensaje en particular fue la salida de un receptor GPS tipo RTK - Real Time Kinematic se recibe en ASCII:

```
\$GPGGA,181908.00,3404.7041778,N,07044.3966270,  
W,4,13,1.00,495.144,M,29.200,M,0.10,0000*40
```

Todos los mensajes NMEA empiezan con el símbolo \$, y cada dato está separado por una coma.

GP indica que es una posición GPS.

GGA es el nombre del mensaje con el cual se espera obtener datos de tiempo, ubicación y fijación (fix).

181908.00 indica el tiempo UTC en horas, minutos y segundos.

3404.7041778 es la latitud en formato DDMM.MMMMM (D=degrees o grados, M=Minutos) el número de decimales es variable.

N denota latitud norte (*North*)

07044.3966270 es la longitud en formato DDDMM.MMMMM (D=degrees o grados, M=Minutos) el número de decimales es variable.

W denota longitud oeste (*West*)

4 denota el indicador de calidad:

1 = coordenada sin corregir

2 = coordenada corregida diferencialmente (ej. sistema WAAS, DGPS)

4 = coordenada RTK (precisión de centímetros)

5 = RTK flotante (precisión de decímetros)

13 denota el número de satélites utilizados para obtener la coordenada.

1.00 denota el HDOP (horizontal dilution of precision).

495.144 denota la altitud de la antena del receptor GPS.

M denota la unidad de medida de la altitud (ej. Metros o Pies)

29.200 denota la separación geoidal (se resta de la altitud de la antena para calcular la altura por encima de la elipsoide)

M denota la unidad de medida de la altitud (ej. Metros o Pies).

0.10 denota la edad de la corrección (si la hay).

0000 denota el número de identificación de la estación de corrección. (si la hay).

***40** denota el checksum, siempre empieza con un asterisco (*).

Esta instrucción \$GPGGA es una de las más comunes, un receptor GPS continuamente está recibiendo secuencias de diferentes instrucciones. Las que más se observan en la recepción (Gakstatter, 2015) son:

\$GPGLL Posición geográfica, Latitud / longitud y tiempo.

\$GPRMC Datos de tránsito /Mínimo de datos recomendados específicos de GPS. Esta instrucción indica si los datos recibidos son válidos.

\$GPGSA GPS DOP (dilution of precision) y el número de satélites activos (a la vista).

\$GPGSV Información detallada como el azimut y elevación de cada satélite detectado.

\$GPVTG Velocidad sobre terreno y el desplazamiento de trayectoria.

\$GPGST Precisión horizontal y vertical estimada.

2.6 Diseño del Hardware

En los siguientes apartados se discute cada uno de los componentes del hardware.

2.6.1 Microcontrolador

Un microcontrolador en su forma más fundamental es todo un sistema computarizado encapsulado en un solo circuito integrado. Uno de los principales retos en el diseño basado en un microcontrolador, es escoger el mejor controlador para un diseño específico. El objetivo es escoger el microcontrolador más económico que tenga los parámetros y características deseadas para la aplicación que se tenga. Los microcontroladores van desde procesadores de 4-bits con características limitadas o completas, hasta procesadores de 32-bits a alta velocidad ([Barrett & Pack, 2005](#)).

Un diagrama de bloques básico de un microcontrolador genérico se muestra en la Figura [2.20](#). Los puertos se usan para acceder al mundo exterior. Normalmente son bidireccionales y también tienen una función alterna como puertos de conversión analógicos-a-digitales, comunicación serial, sistemas temporizados y componentes de almacenamiento. También cuentan con Rutinas de Servicio de Interrupción (ISR en inglés) a través de pines específicos, lo

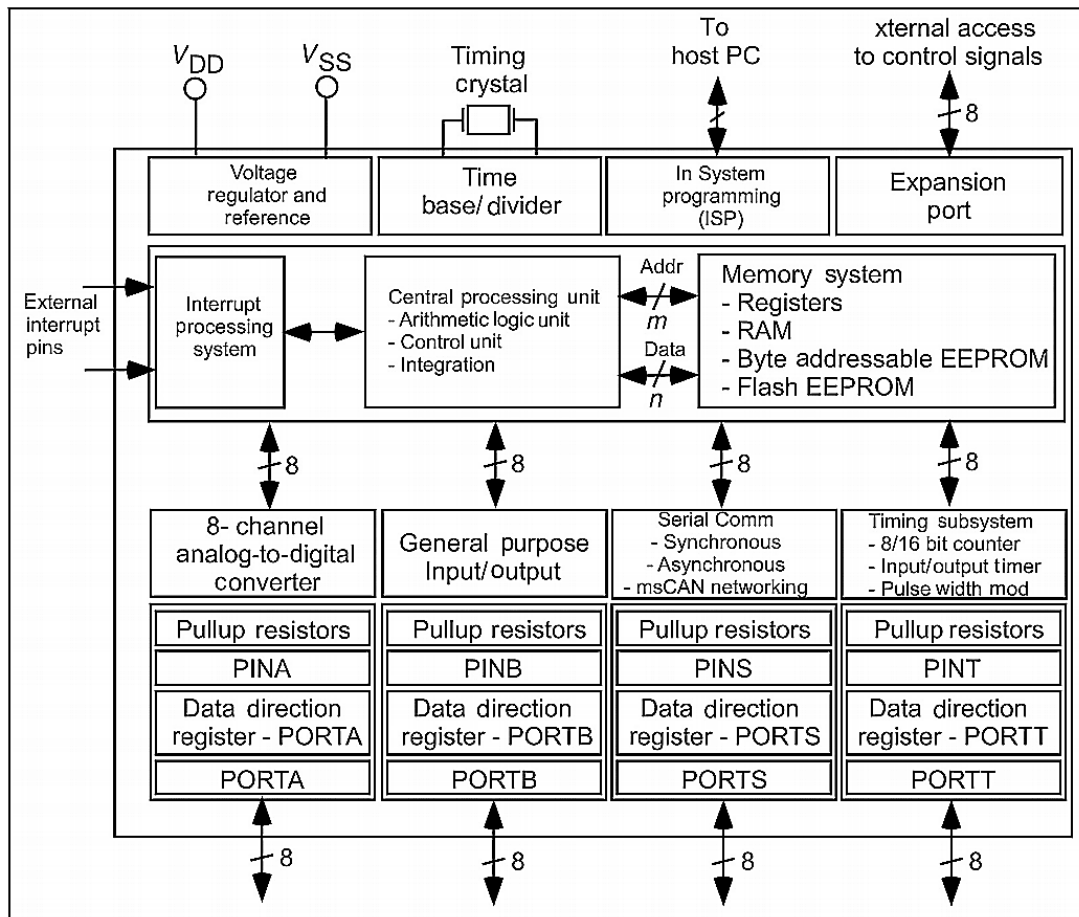


Figura 2.20: Diagrama de bloques del microcontrolador. Los microcontroladores están equipados con los sub-sistemas mostrados.

que permite al microcontrolador dar respuesta a eventos de alta prioridad. Es programado a través de características ISP (In System Programming) usando una computadora anfitriona. La base de tiempo para el microcontrolador es por medio de un cristal oscilador externo [Barrett & Pack \(2005\)](#).

El CPU (central processing unit) dentro del microcontrolador es un circuito secuencial complejo cuya función principal es ejecutar programas que están almacenados en su EEPROM Flash. Un programa es simplemente una serie de instrucciones para ejecutar una tarea específica. Los programas son desarrollados por el diseñador utilizando herramientas de desarrollo de programas (*sketches*), normalmente están en una PC [Barrett & Pack \(2005\)](#). Una vez que

el programa está listo, se descarga al microcontrolador, y este se vuelve un sistema independiente.

Una vez reseteado, el CPU obtiene secuencialmente una instrucción del programa de la memoria, la decodifica (lo que se supone debe hacer) y la ejecuta. La unidad central de procesamiento (CPU) es el centro principal de control en todo el microcontrolador. Mientras responde a las diversas instrucciones del programa, el CPU llamará a los sub-sistemas para ejecutar las tareas.

Conjunto de Registros

Los microcontroladores tienen un complemento de registros designado como conjunto de registros (Register Set). El conjunto de registros es la interfase entre el usuario y los diferentes sub-sistemas en el microcontrolador. Cada registro tiene una colección de flip-flops. Cada flip-flop puede ser puesto en uno o cero lógico. Cada flip-flop puede verse como un interruptor configurado por software. Los flip-flops están clasificados por sub-sistema como lo muestra la Figura 2.21. Usualmente todos los registros asociados a un sub-sistema dado van juntos. Para configurar un sub-sistema específico, el diseñador determinará la configuración adecuada para cada bit dentro del registro.

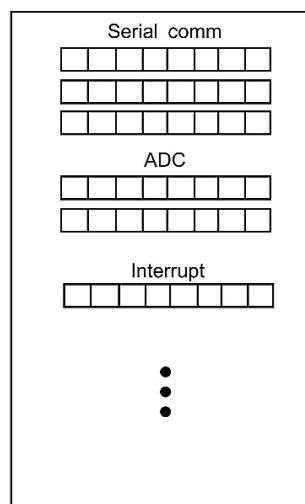


Figura 2.21: Conjunto de Registros de un microcontrolador (Register Set)

La función de cada registro y cada bit de registro están cuidadosamente definida en la documentación del microcontrolador.

Estructura del Bus

Los sub-sistemas dentro del microcontrolador están conectados con algunos buses. Un bus es un conjunto de conductores en paralelo que tiene una función similar. Los microcontroladores están equipados con un bus de direcciones, un bus de datos y un bus de control.

Bus de Direcciones El bus de direcciones provee una conexión entre la unidad central de procesamiento y el sub-sistema de memoria en el microcontrolador. El número de conductores en el bus de direcciones define el límite superior de las posiciones de memoria que puedan ser linealmente direccionadas por el microcontrolador. La primera dirección en el sub-sistema de memoria serán todos en ceros mientras que la dirección final serán todos unos lógicos. El número de direcciones de memoria se determina calculando $2^{\text{lineas de direccion}}$. Por ejemplo, un microcontrolador con bus de direcciones de 16 bits es capaz de direccionar 65,536 registros. La primera dirección en este espacio de memoria es $(0000)_{16}$ mientras que la última dirección será de $(FFFF)_{16}$.

Bus de Datos El bus de datos como su nombre lo dice, se usa para enviar datos a los diferentes sub-sistemas dentro del microcontrolador. Los microcontroladores comúnmente disponen un ancho de 4, 8, 16 o 32 bits en el bus de datos. El ancho en las líneas de datos generalmente determina el tamaño del argumento que el microcontrolador puede procesar. Por ejemplo, el valor más grande de un entero sin signo que pueda almacenarse en un microcontrolador con ancho de 8 bits en la línea de datos es 255. Debemos recalcar que un microcontrolador de 32 bits no es mejor que uno de 4 bits. Recordemos que el objetivo principal del diseñador es seleccionar el microcontrolador más barato que pueda cubrir los requerimientos del diseño.

Bus de Control Los microcontroladores están equipados con líneas para enviar y recibir un conjunto de señales de control designados como bus de control. Estas líneas de señal llevan señales de control a diversos sub-sistemas en todo el microcontrolador. La mayoría de las señales de control son internas (dentro del encapsulado). Sin embargo, a veces son convenientemente previstas en un puerto del microcontrolador de tal manera que componentes externos puedan agregarse al procesador. Las señales de control son enviadas por el CPU en respuesta de las instrucciones del programa para asegurar que la instrucción se ejecute apropiadamente.

Memoria

Como se mencionó anteriormente, el número de posiciones únicas direccionables de memoria en un microcontrolador está determinado por el ancho del bus de direcciones. Esta extensión de memoria direccionable usualmente contiene varios tipos de memoria: **SRAM** (Static Random Access Memory), byte-direccionable **EEPROM** (Electrically Erasable Programmable Read Only Memory), y Flash EEPROM. Cabe recalcar que estos componentes de memoria se encuentran dentro de los límites del microcontrolador.

Para el seguimiento de las posiciones de memoria en uso y el tipo de memoria en el sistema, se utiliza una herramienta visual llamada mapa de memoria. El mapa de memoria provee el tamaño en bytes de cada componente y su dirección de inicio y fin dentro del sistema de memoria.

Base de Tiempo

Un microcontrolador es una máquina compleja de estados sincrónicos que responde a instrucciones de programa en una predecible secuencia *obtener-decodificar-ejecutar*. La velocidad del secuenciado de acciones del microcontrolador es controlada por un base de tiempo externa. La base de tiempo puede ser suministrada por un cristal de cuarzo externo, un oscilador programable, un resonador de cerámica o una base de tiempo interna. Algunos microcontroladores están diseñados para operar a una frecuencia específica como 8 MHz

mientras que otros con un diseño flexible, operan en un amplio rango de frecuencias.

Sub-sistema de Tiempo La fuente principal de reloj es enviada a todo el microcontrolador para proveer sincronía a los sub-sistemas. Para entender las características de este sub-sistema, se explican los términos asociados con la señal de tiempo ilustrado en la Figura 2.22.

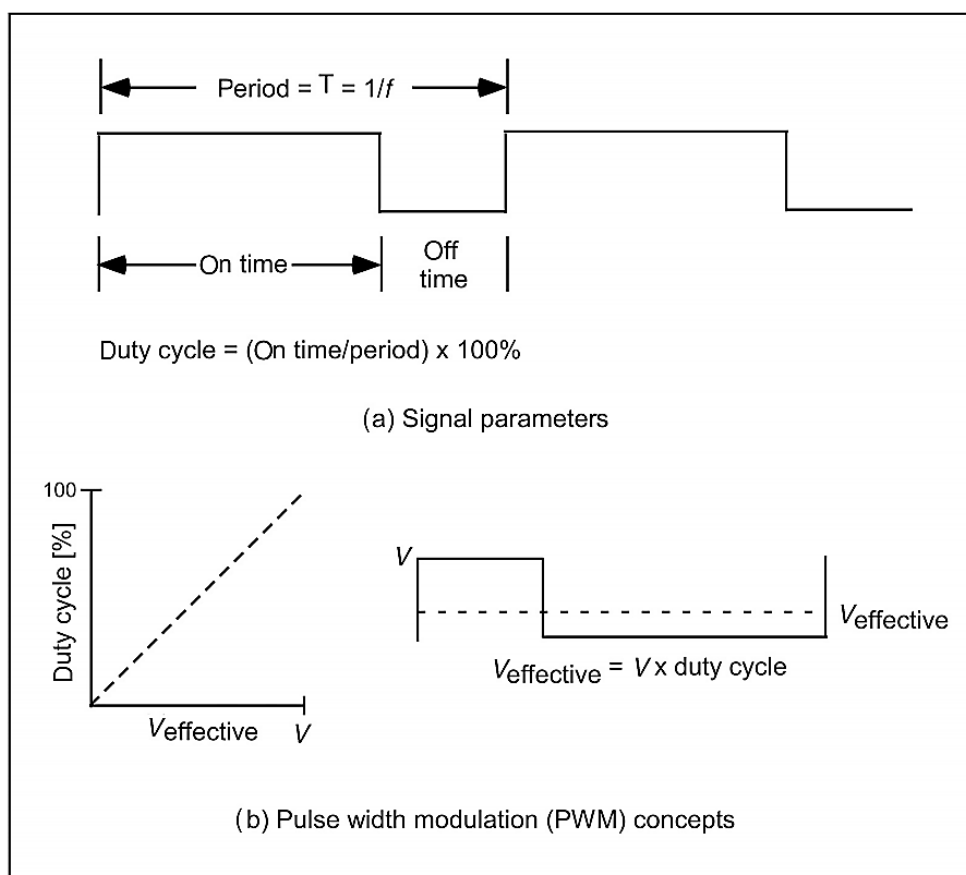


Figura 2.22: Conceptos de Temporización. El voltaje DC efectivo que se entrega a una carga se puede variar a través de ajustar el ciclo de trabajo de una señal PWM digital.

- *Frecuencia:* Frecuencia de señal es el número de ciclos por segundo completados por una señal repetitiva. Se expresa en unidades de Hertz (Hz).

- *Periodo*: El periodo es el incremento de tiempo en segundos requeridos para que la señal repetitiva complete un solo ciclo. El periodo es recíproco de la frecuencia ($T = 1/f$).
- *Ciclo de Trabajo*: Indica el porcentaje de tiempo donde la señal está activa en un solo periodo.
- *Modulación por Ancho de Pulsos*: (también conocida como **PWM** por sus siglas en ingles de *Pulse Width Modulation* Es una técnica en la que se modifica el *ciclo de trabajo* de una señal periódica, ya sea para transmitir información o para controlar la cantidad de energía que se envía a una carga, por ejemplo, controlar la velocidad de motores.

Aplicaciones Los microcontroladores están equipados con un sistema temporizado multi-canal. Los canales dentro del sistema temporizado se pueden configurar para:

- Medir señales de entrada como el periodo y ciclo de trabajo.
- Generar salida de pulsos de precisión o señales repetitivas.
- Contar pulsos entrantes presentes en una señal de entrada.
- Generar señales PWM.

Puertos

Los microcontroladores están equipados con una serie de puertos para dar acceso al exterior del microcontrolador. Estos están organizados en puertos de entrada/salida de 8 bits como se muestra en la Figura 2.20. Un registro de puerto está equipado con un registro de dirección (si es de salida o entrada) para cada pin del puerto. La mayoría de los puertos tienen funciones alternas como la conversión de analógico-a-digital, comunicación serial e interfase de redes.

Convertidores Analógico-a-Digital

Los microcontroladores están equipados con un sub-sistema de conversión analógico-a-digital (ADC). Este sub-sistema convierte continuamente las señales analógicas variantes del exterior a una representación binaria aceptable para el uso del microcontrolador. Estos convertidores comúnmente tienen de 8 a 10 bits de resolución. Por lo tanto, una señal continua es convertida a una serie de instantáneas digitales de la señal analógica. Se debe determinar que tan frecuente se inicia una conversión ADC para una aplicación dada.

2.6.2 Sistemas de Comunicación

La electrónica embebida es acerca de circuitos de enlazado interno (procesadores u otros circuitos integrados) para crear un sistema combinado. Para intercambiar información, estos circuitos individuales deben compartir un protocolo de comunicación común. Cientos de protocolos de comunicación han sido definidos para alcanzar este intercambio de datos, y en general, cada uno puede separarse en dos categorías: paralelo y serial.

Paralelo vs. Serial

Las interfases paralelas transfieren múltiples bits al mismo tiempo. Usualmente requiere de **buses** de datos de ocho, dieciséis, o más conductores. Los datos son transferidos en oleadas inmensas de 1s y 0s. Ver la Figura [2.23](#).

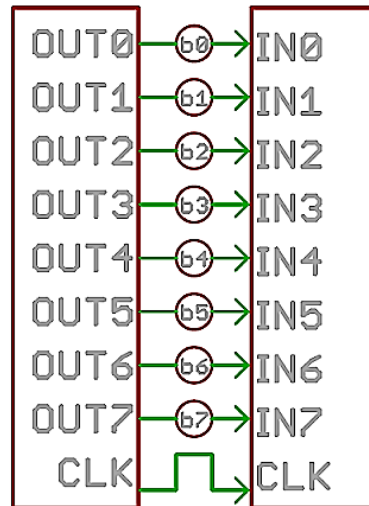


Figura 2.23: Bus de datos de 8-bits, controlado por una señal de reloj, transmitiendo un byte cada pulso de reloj, se requieren 9 conductores (Jimbo, 2018)

Las interfases seriales mandan sus datos, un solo bit a la vez. Estas interfases pueden operar hasta con un solo conductor, usualmente 4 conductores. Ver la Figura 2.24.

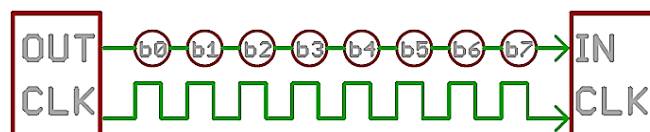


Figura 2.24: Ejemplo de una interfase serial, transmitiendo un bit cada pulso de reloj con dos conductores (Jimbo, 2018).

La comunicación paralela ciertamente tiene sus beneficios. Es rápida, sencilla, y relativamente fácil de implementar. Pero requiere muchas más líneas de entrada/salida. Los que han utilizado un microcontrolador con sus limitaciones de puertos podrán entender el sacrificar la velocidad por el escaso

número de recursos disponibles y preciados. Así que se opta por el uso de comunicación serial.

Serial Asincrónica

A través de los años, docenas de protocolos seriales se han fabricado para alcanzar necesidades particulares de sistemas embebidos. USB (universal serial bus) y Ethernet, son unas de las interfases seriales mejor conocidas. Otras populares son **SPI** (*Serial Peripheral Interface*), **I2C** (*inter-integrated circuit*) y el serial estándar. Cada una de estas interfases se pueden clasificar en dos grupos: *síncronas* y *asíncronas*.

Una interfase serial *síncrona* siempre va acompañada de una señal de reloj, de manera que todos los dispositivos en un bus serial síncrono, puedan compartir una señal de reloj común. Esto lo convierte en uno de los más sencillos y a veces más rápidas transferencias seriales. Ejemplos de interfases son SPI e I2C.

Asíncrono significa que los datos son transferidos sin la ayuda de una señal de reloj externa. Este método de transmisión es perfecto para minimizar los conductores y pines de entrada/salida, y significa que requerirá tener una transmisión y recepción confiables de datos. El protocolo serial que se tratará aquí es la forma más común de transferencia asíncrona. Es tan común, que cuando decimos “serial” estamos hablando de este protocolo.

El protocolo serial “sin-reloj” es ampliamente utilizado en la electrónica embebida. Si se quiere utilizar un módulo GPS, Bluetooth, XBee, LCD u otros dispositivos externos para un proyecto, tendrán habilitados este protocolo serial.

El protocolo *serial asíncrono* tiene un número de reglas/mecanismos integrados que aseguran transferencias robustas y libres de error. Estos mecanismos son:

- bits de datos,
- bits de sincronización,
- bits de paridad,

- y los baudios (bits por segundo, bps).

A través de estos mecanismos de control, nos damos cuenta que no hay forma de enviar datos serialmente. El protocolo es altamente configurable. La parte critica es asegurar que ambos dispositivos en el bus serial estén configurados con los mismos parámetros del protocolo.

Baudios

Los baudios establecen qué tan rápido los datos son enviados por la línea serial. Se expresan en unidades de bits-por-segundo (bps). Si se invierte el baudio, nos damos cuenta que tanto se tarda en transmitir un bit. Este valor determina qué tanto el transmisor mantiene la línea en nivel alto/bajo o a qué frecuencia el receptor muestrea la línea.

Los baudios pueden ser cualquier valor razonable. El único requisito es que ambos dispositivos operen al mismo baudio. Uno de los más comunes valores, especialmente cuando la velocidad no es critica es 9600 bps. Otros valores “estándar” son 1200, 4800, 19200, 38400, 57600 y 115200. Entre más alto vaya, la velocidad aumentará. Sin embargo, hay límites de qué tan rápido se puede transferir. Usualmente no se miran velocidades mayores a 115200, lo cual es rápido para la mayoría de los microcontroladores. Si aumentas este valor se empezaran a ver errores en el lado del receptor, dado que los relojes y el muestreo no se sostienen a más velocidad.

La estructura de los datos

Cada bloque (usualmente un byte) de dato transmitido es enviado en un *paquete* o *trama* de bits. El paquete es creado agregando bits de sincronización y de paridad al dato. Ver figura 2.25.



Figura 2.25: Algunos símbolos en el paquete de datos tienen tamaños de bits configurables (Jimb0, 2018).

Los bits principales en un paquete serial es el dato que transporta. La cantidad de datos en cada paquete puede ser desde 5 a 9 bits. Ciertamente el estándar es un byte de 8-bits. Un dato de 7-bits puede ser más eficiente que uno de 8, especialmente si se transfieren caracteres ASCII de 7-bits.

Después de definir la longitud del caracter, ambos dispositivos seriales también definen qué extremidad del dato (*endianness*) se envía primero, el bit más-significativo o menos-significativo.

Bits de sincronización

Los bits de sincronización son dos o tres bits especiales transferidos con cada paquete de datos. Ellos son el **bit de inicio** y el/los **bit(s) de terminar**. Siempre hay un solo bit de inicio, pero el número de bits de terminar es configurable a uno o dos (comúnmente a uno).

El bit de inicio siempre indicado por una línea de dato sin actividad (idle) yendo de 1 a 0, mientras que el bit de terminar cambiará al estado de inicio manteniendo el 1.

Bits de paridad

La paridad es una forma muy simple de verificar un error a bajo nivel. El bit de paridad puede ser *par* o *impar*. Para producir el bit de paridad, todos los 5-9 bits del byte de dato se suman, y el resultado decidirá si el bit es ajustado o no. Por ejemplo, si la paridad es par, y un datos como 0b01011101, el cual tiene un número impar de 1s (5 en total) el bit de paridad sería puesto a 1. Y a la inversa, si el modo fuera impar, el bit de paridad sería puesto a 0.

La paridad es *opcional* y nos es ampliamente utilizado. Puede ser de ayuda para transmitir en ambientes con mucho ruido, pero también sería un poco más lenta la transferencia y requeriría que el transmisor y receptor implementaran rutinas de manejo de errores (normalmente un dato recibido fallido debe ser re-enviado).

Alambrado y Hardware

Un bus serial consiste de solo dos líneas, una para enviar datos y otra para recibir. Como tal, los dispositivos seriales deben tener dos pines: el receptor, **RX** y el transmisor, **TX**. Ver figura 2.26.

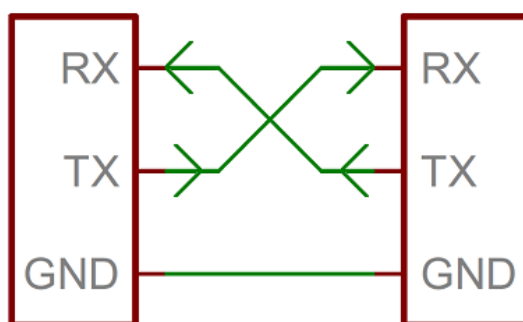


Figura 2.26: Conexión serial UART(Jimb0, 2018).

Es importante notar que esos nombres RX y TX son con respecto al dispositivo mismo. De tal manera que el RX de un dispositivo deberá conectarse al TX del otro dispositivo, y viceversa. Dicho de otro modo, el transmisor debe hablar con el receptor y no con otro transmisor.

Una interfase serial donde ambos dispositivos podrían enviar y recibir datos se le denomina **full-duplex** o **half-duplex**. Full-duplex significa que ambos dispositivos pueden enviar y recibir simultáneamente. Half-duplex significa que los dispositivos deben tomar turnos para enviar y recibir. Algunos buses seriales podrían tener solamente una conexión entre el transmisor y el receptor, por ejemplo un dispositivo LCD siempre recibe datos sin necesidad de enviar algún dato de regreso al dispositivo de control. Este tipo de comunicación serial se le llama **simplex**. Todo lo que se necesita es un solo alambre que se conecte del TX del dispositivo maestro, al RX del receptor.

Implementación de Hardware

Ya sabemos que cables se necesitan. Pero, ¿Como se implementa a nivel de señales? Hay varios estándares para la señalización serial. Los más conocidos

son: Lógica TTL - Transistor-Transistor-Logic y RS-232.

Cuando los microcontroladores y otros circuitos integrados de bajo nivel se comunican en forma serial usualmente lo hacen con lógica TTL (transistor-transistor logic). Las señales seriales TTL existen en el rango de voltaje de alimentación de un microcontrolador - 0V a 3.3V o 5V. Una señal a nivel VCC (3.3V, 5V, etc.) indica una línea en reposo o un bit con valor de 1, or un bit de terminar. Una señal de 0V (GND) representa, un bit de inicio o un bit de dato de valor 0. Ver Figura 2.27.

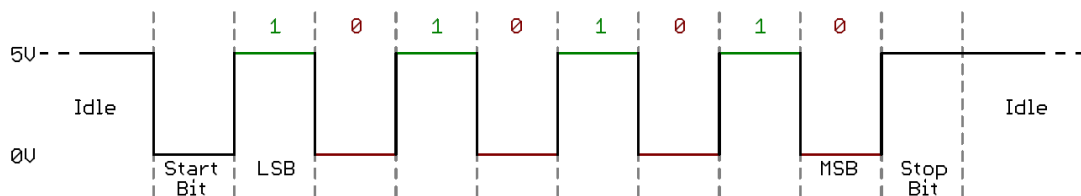


Figura 2.27: TTL serial (Jimb0, 2018).

El RS-232, puede ser encontrado en algunos de las computadoras y periféricos del pasado, es como el TTL serial invertido. Las señales RS-232 están en el rango de -13V a +13V. entonces la especificación permite cualquier valor desde $\pm 3V$ hasta $\pm 25V$. En estas señales un voltaje bajo (-5V, -13V, etc.) indica una línea en reposo, un bit de terminar o un bit de dato en 1. Una señal en alto significa un bit de inicio o un bit de dato en 0. En esencia lo opuesto al TTL serial. Ver Figura 2.28.

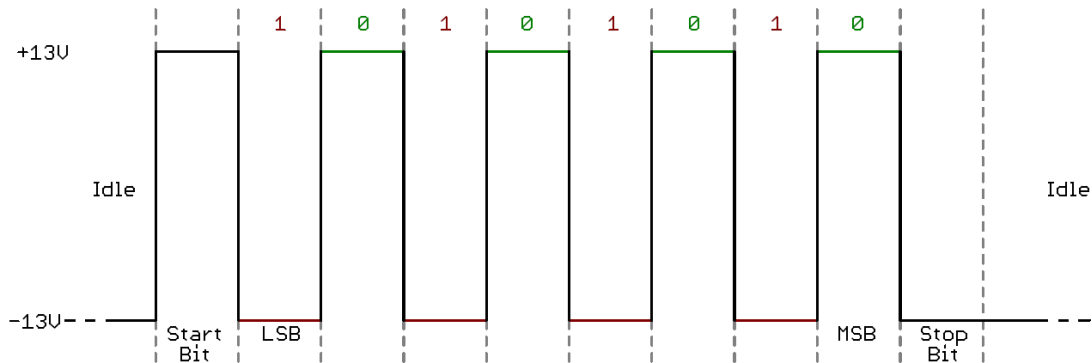


Figura 2.28: RS-232 serial (Jimbo, 2018).

El estándar TTL es más fácil de implementar en circuitos embebidos. Sin embargo, el voltaje a nivel bajo es más susceptible a pérdidas de voltaje a través de largas líneas de transmisión. RS-232, o estándares más complejos como el RS-485, están más preparados para transmisiones de largo alcance.

Cuando se conectan dos dispositivos seriales juntos, es importante estar seguros que manejan los mismos niveles de voltaje. No se puede conectar directamente un dispositivo serial TTL a uno con bus RS-232. Se tienen que acondicionar las señales.

UART

Un Transmisor/Receptor Asincrónico Universal (UART) es un bloque de circuitos responsables de implementar la comunicación serial. En esencia, el UART actúa como un intermediario entre las interfases paralelas y seriales. En un extremo del UART está el bus paralelo y sus pines de control, en el otro están los dos conductores seriales, RX y TX. Ver Figura 2.29.

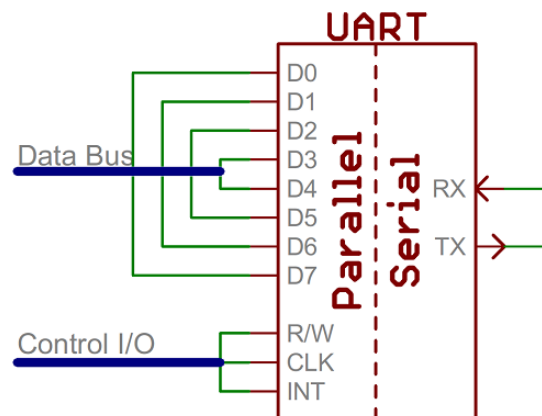


Figura 2.29: Interfase UART simplificada (Jimb0, 2018).

Los UART se encuentran como un IC en su propio encapsulado. Pero también se encuentran integrados en la mayoría de los microcontroladores. Se tendría que verificar la hoja de datos para saber si tienen UARTs. Algunos no tienen, algunos tienen uno, y otros tienen varios. Por ejemplo, el Arduino Uno con el microcontrolador Atmega328 tiene un solo UART, mientras que el Arduino Due tiene cuatro UART.

Los UARTs son los responsables de enviar y recibir datos en serie. El UART del transmisor debe crear un paquete de datos- agregando los bits de sincronización y paridad- y enviar ese paquete a su línea TX con una precisión en el tiempo (de acuerdo a los baudios). El UART del receptor, tiene que muestrear la línea RX a la velocidad esperada, tomar los bits y acondicionar el dato.

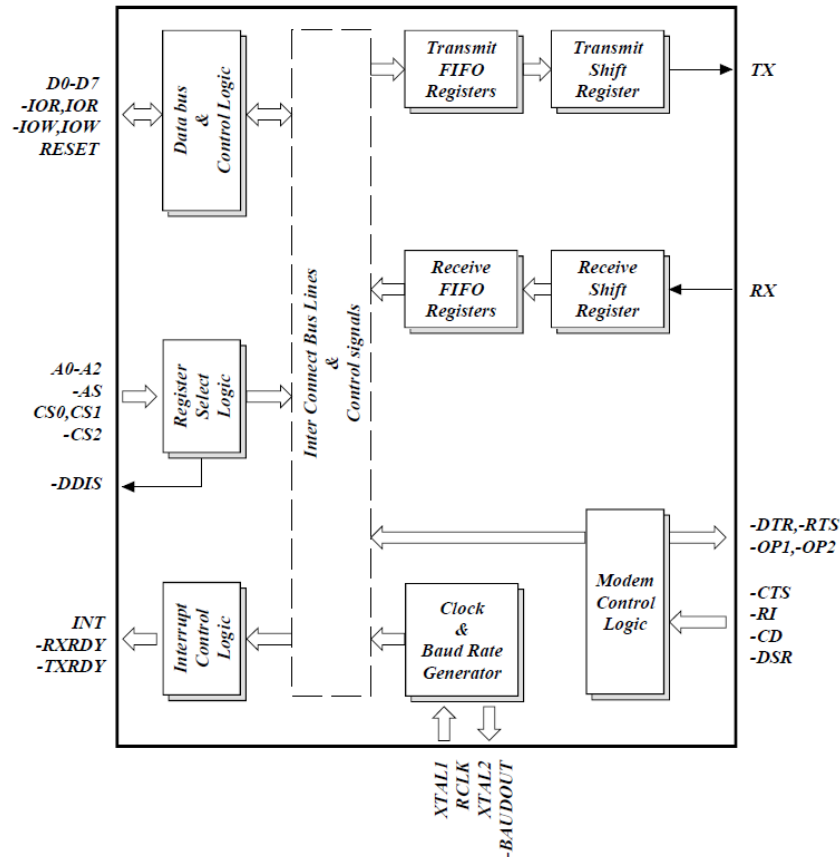


Figura 2.30: Diagrama de bloques interno del UART (Jimbo, 2018).

Algunos UART más avanzados tienen un buffer donde reciben los datos, de donde son tomados. Los UART normalmente liberan los datos según el sistema de colas FIFO (First In, First Out). El buffer puede ser tan pequeño de unos cuantos bits, o tan grande de algunos Kbytes.

UART software

Si un microcontrolador no tiene UART (o no tiene los suficientes) la interfase serial puede emularlo vía software. Este es el método usado por las librerías de Arduino como el SoftwareSerial.h o Softserial.h. La emulación es un recurso que requiere de uso intensivo del procesador, y no es tan preciso como el UART

de hardware. Pero saca del apuro. Con este método se utilizan otros pines disponibles donde cada uno tenga una función determinada.

2.6.3 SPI - Serial Peripheral Interface

Introducción

La interfase serial periférica (SPI) es un bus utilizado comúnmente para enviar datos entre microcontroladores y periféricos pequeños como los registros de corrimiento, sensores y tarjetas SD. Utiliza líneas de datos y reloj por separado, junto con una línea de selección de dispositivo (Select Line) con el que se quiere comunicar (MikeGrusin, 2018).

Un puerto serial común, con líneas TX y RX, se le denomina *asíncrono* porque no hay control sobre cuando es enviado el dato o no se garantiza que ambos lados de la línea de comunicación tengan la misma configuración. Debido a que las computadoras se basan en que están sincronizadas con una señal de “reloj” (el cristal principal pegado a una computadora que maneja todo), esto puede ser un problema cuando dos sistemas con alguna diferencia en las velocidades de reloj, traten de comunicarse entre sí (MikeGrusin, 2018).

Para trabajar en este problema, conexiones seriales asincrónicas agregan otros bits de inicio y fin a cada byte que ayuda al receptor sincronizarse al dato a medida que llega. Además, ambos lados deben ponerse de acuerdo en la velocidad de transmisión anticipadamente. Diferencias mínimas no son problema porque el receptor se re-sincroniza al inicio de cada byte (MikeGrusin, 2018).

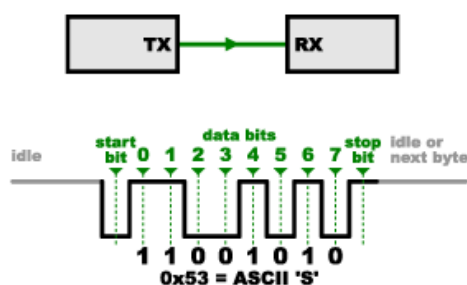


Figura 2.31: Trama de bits serial UART(MikeGrusin, 2018).

El protocolo serial asincrónico funciona bien, pero tiene que realizar mucha preparación con los bits de inicio y fin de cada byte y el hardware requerido para enviar y recibir datos. Ya hemos trabajado con este protocolo y el resultado de no igualar las configuraciones en ambos dispositivos resulta en “caracteres basura” para el receptor. Esto es porque está muestreando los bits a tiempos muy específicos. Si el receptor lee a destiempo, solo mirará bits erróneos. En la Figura 2.31 las líneas punteadas indican el momento exacto en que son leídos los bits.

Una solución sincrónica

El protocolo SPI trabaja de una forma un poco diferente. El bus de datos está sincronizado, significa que usa líneas separadas de datos y un reloj que mantiene ambos lados en perfecta sincronía. La señal de reloj oscila para indicar al receptor exactamente cuando muestrear los bits en la línea de datos. Esto podría ser en transición de subida o bajada en la señal de reloj. La hoja de datos especifica cuál modo utilizar. Cuando el receptor detecta esta transición, mirará inmediatamente a la línea de datos para leer el siguiente bit, ver la Figura 2.32. Dado que el reloj es enviado a la vez que el dato, especificar la velocidad no es importante, a pesar de que los dispositivos tengan una velocidad mayor a la cual puedan operar.

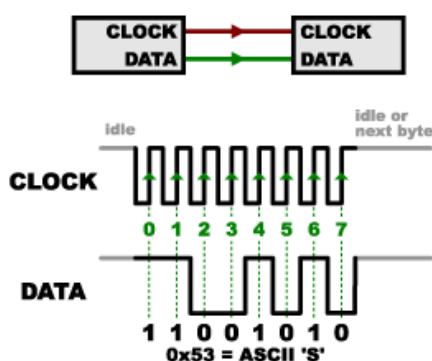


Figura 2.32: Trama de bits de datos serial sincrónica y pulsos de reloj (MikeGrusin, 2018).

Una razón por la cual es popular el protocolo SPI, es que el hardware del receptor puede ser un simple registro de corrimiento. Esto es más sencillo y barato que lo requerido por el UART.

Recibiendo Datos

En el bus SPI, un lado genera la señal de reloj (usualmente CLK o SCK, por **S**erial **C**loc**K**). El lado que genera la señal de reloj es denominado “maestro”, y el otro lado es llamado “esclavo”. Siempre hay un solo maestro, usualmente es el microcontrolador, y puede haber múltiples esclavos.

Cuando el dato es enviado desde el maestro al esclavo, lo envía en la línea de datos denominada *MOSI* (Master Out / Slave In). Si el esclavo necesita enviar una respuesta al maestro, el maestro continuará enviando un arreglo *predefinido* de pulsos de reloj, y el esclavo pondrá el dato en una tercer línea de datos llamada *MISO* (Master In / Slave Out). Ver Figura [2.33](#).

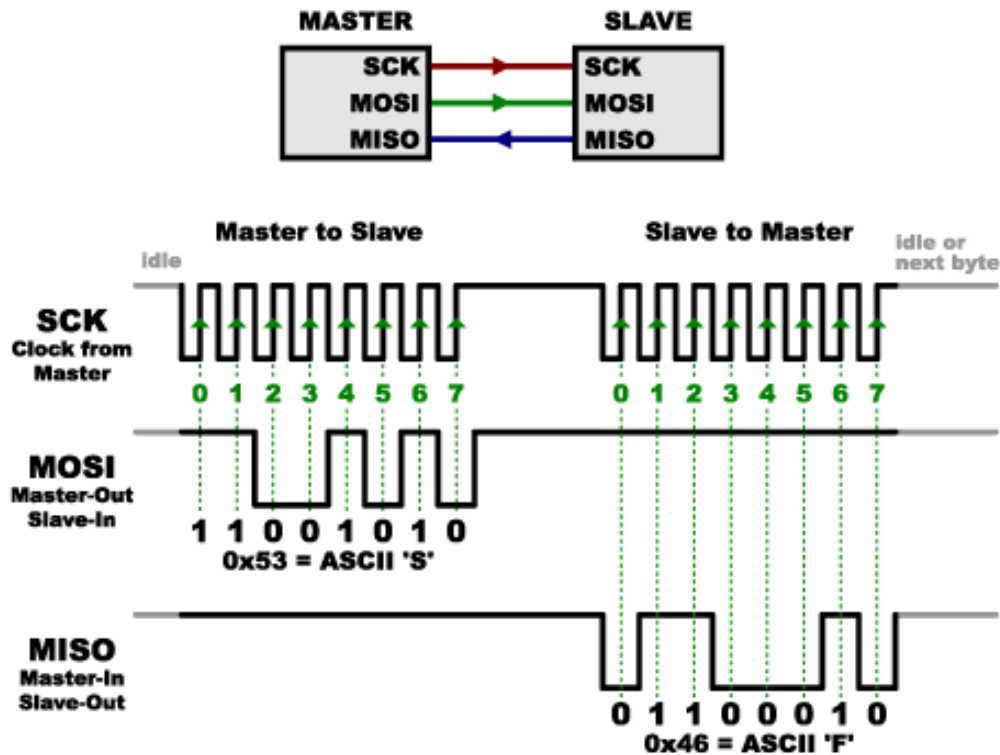


Figura 2.33: Trama de bits datos de entrada y salida y los pulsos de reloj en Bus SPI (MikeGrusin, 2018).

Se le llama arreglo *predefinido* porque el maestro siempre genera la señal de reloj, debe saber con anticipación cuando un esclavo necesita regresar datos y cuantos datos se regresarán. Esto es muy diferente al bus serial asincrónico, donde cantidades aleatorias de datos pueden ser enviados en cualquier dirección en cualquier momento. En la práctica esto no es un problema, debido a que el bus SPI es generalmente utilizado para hablar con los sensores que tienen una estructura específica de instrucciones. Por ejemplo, si envías una instrucción para leer un dato a un dispositivo, se espera que envíe de regreso, por decirlo así, dos bytes.

Se puede observar que el bus SPI es “Full Duplex” (porque tiene líneas separadas de transmisión y recepción de datos) donde en ciertas situaciones se pueden enviar y recibir datos al mismo tiempo. Es una buena costumbre leer la hoja de datos del chip que se desee trabajar.

Slave Select - SS

Una última línea a tomar en cuenta, la cual es denominada Selección de Esclavo (Slave Select). Esta línea indica al esclavo que debe despertar y recibir/enviar datos y también es utilizado cuando múltiples esclavos están presentes en el bus y se pueda seleccionar el que se desee.

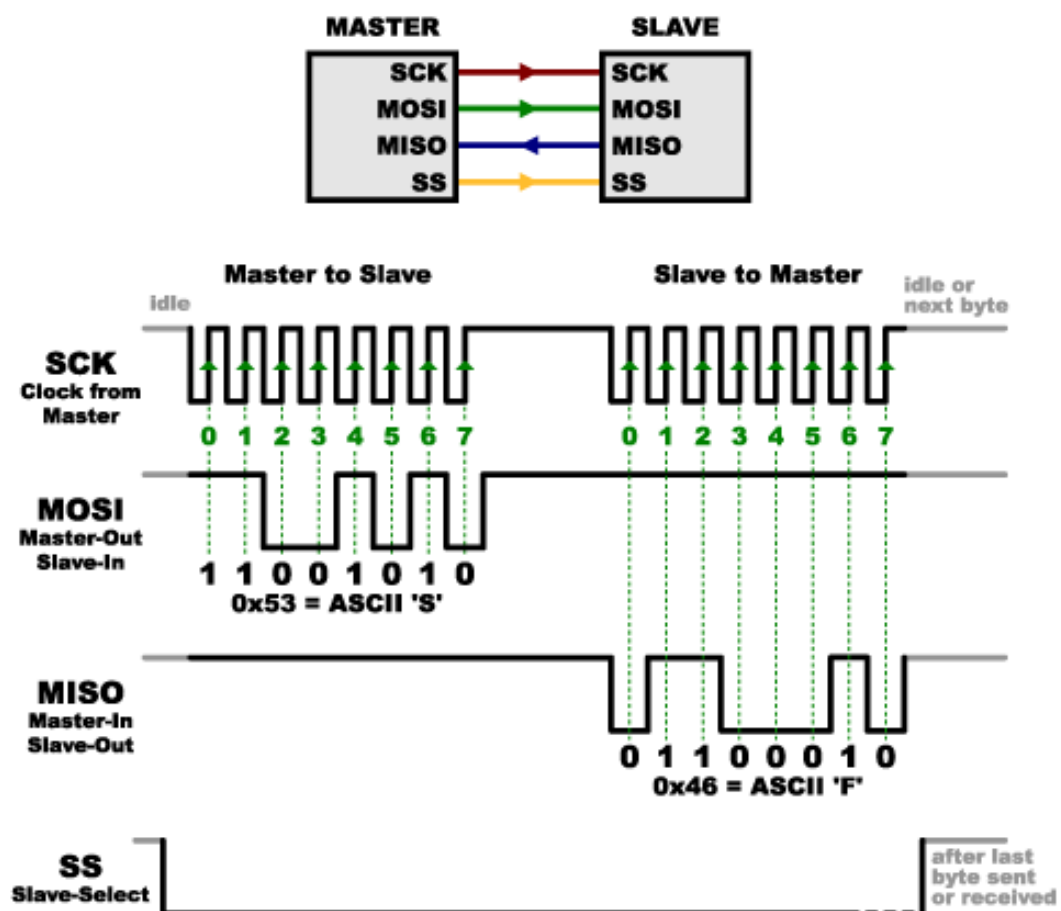


Figura 2.34: Trama de bits datos de entrada y salida y los pulsos de reloj en Bus SPI utilizando SS (MikeGrusin, 2018).

La línea SS normalmente se mantiene en alto, lo cual hace desconectar el esclavo del bus SPI. Solo antes de enviar el dato, la línea se activa con valor lógico bajo, con lo que se activa el esclavo. Cuando dejas de usar el esclavo, la línea regresa a su estado alto.

Esclavos múltiples

En general, cada esclavo necesitará una línea SS por separado. Para hablar con un esclavo en particular, se tendrá que activar la línea SS de ese esclavo y desactivar las líneas de los demás esclavos. Ver Figura 2.35.

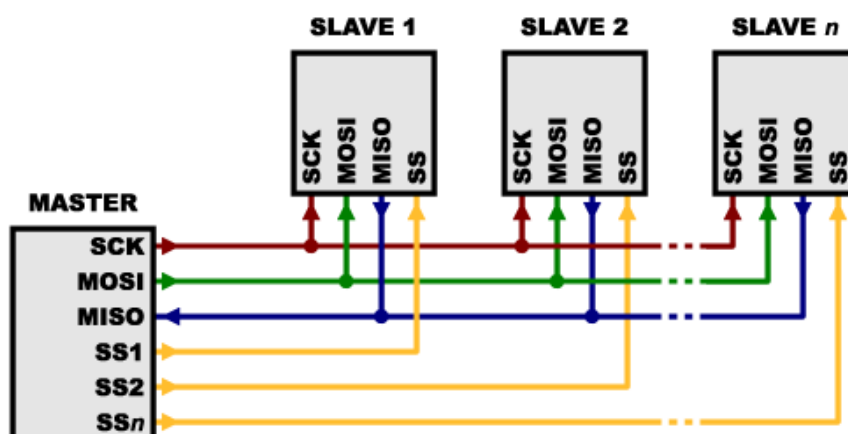


Figura 2.35: Conexión multi-esclavos en bus SPI (MikeGrusin, 2018).

2.6.4 Inter-Integrated Circuit - I2C

El protocolo I2C es un protocolo que permite múltiples ICs esclavos comunicarse con uno o más maestros. Al igual que el protocolo SPI, es solo para comunicaciones a corta distancia dentro de un solo dispositivo. Al igual que las interfases seriales asincrónicas (RS-232 o UART), solo requiere de dos conductores para el intercambio de información (SFUPTOWNMAKER, 2018).

El protocolo I2C requiere solamente dos conductores, como el serial asincrónico, pero estos dos hilos pueden manejar hasta 1008 dispositivos esclavos. También, a diferencia del SPI, I2C puede manejar sistemas multi-maestros, permitiendo más de un maestro comunicarse con todos los dispositivos en el bus (lo único es que la comunicación entre dispositivos maestros no es posible y que deben tomar turnos para el uso del bus) (SFUPTOWNMAKER, 2018).

Las velocidades de transferencia están entre el serial asincrónico y el SPI; muchos dispositivos I2C pueden comunicarse a 100 o 400KHz. Hay un manejo

extra con el uso de I2C, por cada 8-bits de dato a enviar, se envía un bit extra de meta-datos (ACK-NACK). Ver Figura 2.36.

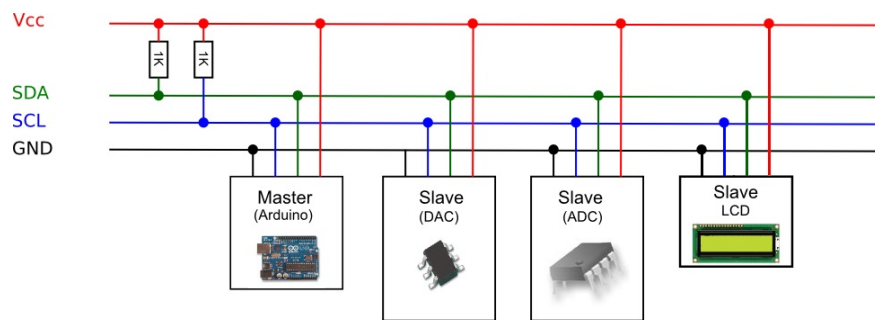


Figura 2.36: Conexión Maestro y esclavos usando mismo bus de datos, en protocolo I2C (SFUPTOWNMAKER, 2018).

El hardware del bus I2C es más complicado que del SPI, pero menos que el serial asincrónico.

Las señales del I2C

Cada bus I2C consiste de dos señales: SCL y SDA. SCL es la señal de reloj, y SDA de datos. La señal de reloj siempre es generada por el maestro que tiene el bus. Algunos dispositivos esclavos pudieran forzar la señal de reloj hacia bajo, para evitar que el maestro siga enviando datos. A esto se le llama *estiramiento del reloj* (SFUPTOWNMAKER, 2018).

A diferencia de las conexiones en UART o SPI, los drivers del bus I2C son de “colector abierto”, lo que significa que pueden jalar la señal correspondiente a nivel bajo de voltaje, pero no pueden llevarla a nivel alto, el hacerlo así puede dañar los drivers del bus o que haya excesiva disipación de potencia en el sistema. Para evitar esto, cada línea de señal deberá contar con un *resistor pull-up* para restablecer la línea a alto cuando ningún dispositivo quiera bajar la señal (SFUPTOWNMAKER, 2018), en la figura anterior también se puede observar esto.

Existen todavía mucha más información sobre este protocolo:

- niveles de señal

- el protocolo
- la condición de inicio
- la trama de direccionamiento (address frame)
- la trama de datos
- la condición de stop
- el direccionamiento de 10-bits
- la condición repetida de inicio
- el estiramiento del reloj.

Inicialmente se contemplaba el uso de este protocolo o bus de comunicación. Sin embargo, durante las pruebas de funcionamiento de sensores, específicamente el sonar, se observó un error. Después de muchas pruebas e investigación, se identificó un problema de programación a bajo nivel en la librería de Arduino (Nemonic, 2015). Se decidió no utilizar esta opción. Afortunadamente la mayoría de los dispositivos, tienen más de un tipo de bus de comunicación lo que los hace atractivos para implementar, ofrece una flexibilidad para el diseño del sistema. Así que no se seguirá detallando este protocolo.

2.6.5 Aplicación

Existe una amplia variedad de drones, tipos, autonomía de vuelo, capacidad de carga (si es que habrá). Se deberá considerar la habilidad del usuario y la del dron para sostener una estabilidad adecuada. Como ya se ha visto anteriormente, la aplicación podría ser para vuelo en interiores, mientras que otros solamente podrán volar en exteriores. Aquellos que hacen acrobacias, los que son para competencia en circuitos con visión en primera persona (FPV - First person view) con alguna cámara de última generación. Por otro lado, podría considerarse la fotografía o vídeo de vista aérea. El determinar los componentes que llevará, reducirá la lista a lo necesario. Una vez determinado el tipo de dron y la tarea a realizar, hay que escoger componentes con especificaciones exactas, por ejemplo, si es requerido que lleve una cámara, qué tipo y la resolución. Si tiene estabilidad de imagen, etc. así como otras características,

el tipo de controlador que llevará, si se necesitará que el dron tenga capacidad GPS, el tamaño, qué tanto ensamble requerirá, etc. Algunos drones tienen mayor tiempo de vuelo (autonomía de vuelo) o tipos de batería diferentes. Así como los accesorios extras que puedan traer.

Como resultado, seleccionar el dron correcto significa escoger un dron que tenga las características necesarias. Si quisiera volar el dron inmediatamente, se escogería un dron RTF (ready to fly). Si quisiera tomar excelentes fotografías, seleccionaría un dron con una cámara de alta-definición. Si la durabilidad es lo importante, seleccionaría un dron que tenga guardas para las hélices o recomendaciones de drones que tenga conocimiento de resistencia al impacto. Sin importar las características que seleccionen, lo importante es escoger un dron que sirva a su propósito y que no tenga características extras que no vayan a usar.

2.6.6 ¿Cuales son las partes de un dron?

Un quadrirrotor incluirá los elementos en la siguiente lista para poder volar:

- **Chasis**
- **Motores**
- **ESC**
- **Hélices**
- **Batería**
- **Controlador de vuelo**
- **Radio Control o Transceptor de Radio**

2.6.7 Chasis

El esqueleto de un quadrirrotor es el cuadro o chasis, los motores y hélices. Veamos primero el chasis. Viene en una variedad de tamaños y rango de pesos. La mayoría tienen la misma forma básica, en forma de X. Para el aficionado que desea montar algo de peso extra, como una cámara, se recomienda un chasis más robusto. Sin embargo, agregar más refuerzo, significa más peso al aparato,

lo cual requerirá hélices más grandes y motores más potentes para crear la elevación necesaria de todo el peso del dron. Debe de haber un balance entre la velocidad de vuelo, la maniobrabilidad y tiempo de vuelo.

El chasis puede ser de aluminio, fibra de carbono o madera. El material más utilizado para los brazos es aluminio. Son relativamente ligeros, rígidos y baratos, pero suelen transmitir las vibraciones del motor, a diferencia de los brazos de fibra de carbono. Este es un dato importante dado que las vibraciones se traducen en ruido para los sensores. La fibra de carbono tiene mejor absorción de las vibraciones del motor y presenta mayor rigidez.

2.6.8 Batería

El dron de ala rotatoria experimenta alto consumo de energía comparado con otras configuraciones de ala. Como multirrotor, tiene más capacidad de carga a costa del consumo elevado de energía, por lo tanto, de resistencia Schmidt (2011). Con rígidas restricciones de tamaño, peso y energía, el mercado de drones para aficionados es dominado por las baterías LiPo, que ofrecen compromiso óptimo a moderado de energía específica (energía por unidad de volumen), alta potencia específica (potencia por unidad de volumen) y alto ciclo de vida Mulgaonkar *et al.* (2014).

Las baterías LiPo (Polímero de Litio) son un tipo de batería recargables, y la razón principal que el vuelo eléctrico sea una opción viable respecto a los modelos de combustible. Las características de estas baterías son:

Ventajas:

- Son ligeras y se pueden hacer de casi cualquier forma y tamaño.
- Tienen gran capacidad, (energía específica).
- Tienen una tasa de descarga alta para alimentar sistemas con alta demanda de corriente.

Desventajas:

- Su electrolito es volátil. Puede explotar o incendiarse.

- Requiere un cuidado especial y adecuado para que sea durable. La carga, descarga y almacenamiento afecta a la batería (tiempo de vida). Se realiza con un cargador especial para baterías LiPo.

Voltaje o Clasificación “S”

A diferencia de las baterías NiCd o NiHm que contienen celdas de 1.2 volts, las baterías LiPo son de 3.7 y 4.2 volts cuando están totalmente cargadas. La siguiente Tabla 2.3 es de voltajes y el número de celdas por batería:

Tabla 2.3: Voltajes de Batería

Voltaje Batería	#Celdas	Volts/cell	Series (S)
3.7 volts	= 1 celda	× 3.7 volts	1S
7.4 volts	= 2 celdas	× 3.7 volts	2S
11.1 volts	= 3 celdas	× 3.7 volts	3S
14.8 volts	= 4 celdas	× 3.7 volts	4S
18.5 volts	= 5 celdas	× 3.7 volts	5S
22.2 volts	= 6 celdas	× 3.7 volts	6S

El parámetro entre paréntesis es el número de celdas conectadas en serie dentro de la batería. También se pueden conectar en paralelo para aumentar su capacidad. Esto se indica mediante un número seguido de la letra **P** (Positive-Imports, 2014). Ejemplo: 3S2P indica 2 arreglos en paralelo de 3 celdas en serie por arreglo.

Capacidad o mAh

La capacidad indica cuánta energía puede mantener la batería, en miliampere-h (mAh). Esta es una manera de indicar la cantidad de carga medida en miliamperes que se pueden poner en la batería durante 1 hora para que se descargue completamente. Por ejemplo una batería LiPo que tiene 1000mAh sería descargada en una hora con carga de 1000 miliamperes colocado en él. Si

esta misma batería tenía una carga de 500 miliamperes tomaría dos horas para descargarla (admin, 2014).

Tasa de descarga o “C”

Esta es probablemente la característica más sobre-valorada y menos comprendida dentro de la clasificación de las baterías. La tasa de descarga es simplemente la rapidez con que una batería puede ser descargada de forma segura. El flujo de los iones del ánodo al cátodo indicará la velocidad de la descarga. En el mundo de las baterías LiPo, la clasificación se denomina C (Positive-Imports, 2014).

Una batería con tasa de descarga 10C, puede descargarse a un ritmo de 10 veces la capacidad de la batería, 15C = 15 veces, , 20C = 20 veces, y así sucesivamente.



Figura 2.37: Batería LiPo 7.4V (2S) 1500mAh 25C (Modeltronic, 2018)

Voltaje y Corriente máximo de carga

Una celda LiPo de 3.7 volts está cargada al 100% cuando alcanza los 4.2 volts. Cargarla más puede destruir las celdas de la batería y podría incendiarse. Un cargador inteligente se detendrá cuando la batería alcanza los 4.2 volts por celda. Un cargador inteligente equilibrará la carga por cada celda.

Es importante utilizar un cargador especial para este tipo de batería y seleccionar el voltaje correcto o el número de celdas. Si se tiene una batería de 2

celdas seleccione 7.2 volts o 2S. Si tiene seleccionado 11.1 Volts (3S) por error y trata de cargar una batería 2S, podría dañarse y acortar el tiempo de vida de la batería. La mayoría de los cargadores inteligentes, utilizan un método de carga de **corriente constante / voltaje constante (cc/cv)**. La corriente constante se aplica a la batería durante la primera parte del ciclo de carga. A medida que el voltaje en la batería se acerca al 100%, el cargador reduce la corriente de carga y luego aplica una tensión constante. Dejará de cargar cuando el 100% del voltaje de carga iguale a la constante configurada(4.2 volts por celda) (ad-min, 2014).

Corriente de carga

Seleccionar una corriente de carga correcta es crítico cuando se trata de una batería LiPo. La indicación es usar una corriente no mayor a 1C. Por ejemplo si la batería es de 2000mAh entonces la corriente de carga a usar es de 2000mA. Hay cargadores que no tienen capacidad de entregar la corriente requerida, por lo que se utilizará el máximo nominal permitido a costa de una carga lenta.

Proceso de Carga

El proceso de carga aplica una corriente inversa a la generación de forma que estos átomos estables, al aplicarse la misma energía que han usado para unirse, comienzan a separarse otra vez en iones por un proceso de rotura de enlaces. Mientras que el cargador está conectado, el dispositivo se alimenta de dicho cargador y los iones siguen separándose hasta que se completa el proceso y no se unen porque está actuando la corriente inversa que lo impide. Este proceso se compone de tres fases (Positive-Imports, 2014):

Subida de corriente - subida tensión Se mide el voltaje actual de la batería, si este es inferior a 2.75v el cargador la interpreta como dañada. Si el voltaje es superior a 2.75v, empieza el proceso de carga a una intensidad mínima que va subiendo hasta la seleccionada. El voltaje de la batería también va subiendo. Esta fase es muy corta y en algunos cargadores no llega a ser visible.

Subida de tensión - intensidad constante Cuando se alcanza la intensidad de carga seleccionada, la tensión va aumentando hasta los 3.6 o 4.22v por elemento. Esta es la fase de carga que proporciona una larga vida (400-500 ciclos). La batería carga aproximadamente el 80% de su capacidad.

Bajada de intensidad - tensión constante Una vez alcanzada la tensión máxima, la intensidad de carga baja lentamente sin sobrepasar la tensión máxima. Cuando la intensidad de carga baja al mínimo, normalmente 100mA, el proceso de carga ha finalizado. Esta fase no es tan larga como la anterior y viene a completar el último 20% aproximado de capacidad en la batería. A medida que baja la intensidad, el proceso es más lento. No conectar la batería más tiempo del necesario, se aconseja definir un límite máximo y no abusar.

2.6.9 Motores

Probablemente el siguiente componente más importante es el motor. Los motores se clasifican en unidades “Kv”, las cuales equivalen al número de revoluciones por minuto que un motor puede alcanzar cuando la corriente de un 1 volt lo alimenta sin impedimentos. Entre más alto sea el Kv, más rápido puede girar el motor. Sin embargo, más no significa mejor. Un motor más rápido requiere mucha más energía de la batería, causa que el tiempo de vuelo disminuya. Más RPMs también reduce la vida del motor a largo plazo. A esto se le llama que los motores se *desgasten* o *quemen*. Los motores sin escobillas (*brushless*) son los que se utilizan en aplicaciones de drones eléctricos, debido a su alta eficiencia. A diferencia de los motores con escobillas, los primeros no generan grandes niveles de ruido. Pero el costo de los motores sin escobillas son más altos.



Figura 2.38: Motor DC 2212 1000KV para multirrotor

Componentes de motores sin escobillas

Un motor sin escobillas consiste de dos secciones principales (ver la Figura 2.39)

- **Rotor** - La parte que rota y tiene los imanes instalados.
- **Estátor** - La parte que no gira y tiene los electroimanes.

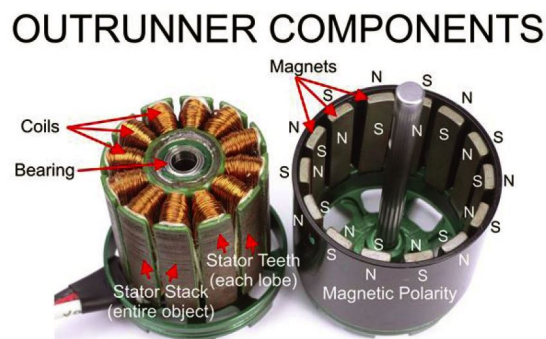


Figura 2.39: Componentes de un motor Outrunner sin escobillas: Rotor y Estátor. En este tipo de motor la carcasa está unida al rotor (Sam, 2014)

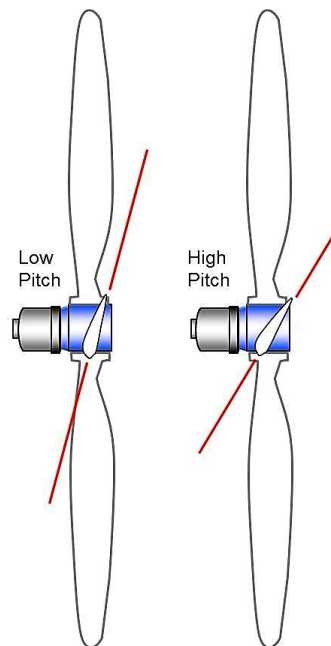
Hay una siguiente clasificación de los motores sin escobillas que son: *Outrunner* e *Inrunner*. Sin embargo, con los multi-rotors y muchas aeronaves de ala fija, solo encontrarán motores outrunner como en la Figura 2.39. La diferencia entre los dos, es que un outrunner tiene el rotor en la parte exterior (carcasa), y un motor inrunner tiene el rotor en su interior con imanes fijos, mientras la carcasa se mantiene estacionaria (Sam, 2014).

Los *outrunner* son motores que están diseñados para trabajar a bajas revoluciones en aplicaciones de alto torque. En cambio los motores *inrunner* están diseñados para trabajar a altas revoluciones, en aplicaciones de bajo torque. En este tipo de motor es similar a un motor DC con escobillas, pero los imanes están fijos al rotor.

Estos motores son controlados mediante un placa electrónica, llamada **ESC** (Electronic Speed Controller) genera señales de pulso modulado que se envían a los motores y regulan su velocidad.

2.6.10 Hélices

Las hélices crean un gran efecto en velocidad a la cual vuela el quadrirrotor, la carga que puede llevar, y la velocidad de maniobras. Para influir sobre estos atributos se puede incrementar o decrementar la longitud de las hélices y el *pitch* de la hélice. El *pitch* es la forma e inclinación de la hélice. Como ya se ha visto, cada quadrirrotor, tiene dos sentidos en la rotación, en sentido de las manecillas del reloj y al contrario. Por lo tanto, para cada sentido de rotación, la hélice debe ser la indicada para poder generar el impulso necesario así como para contrarrestar las fuerzas de los momentos entre sí. Cuando se selecciona el tipo de hélice, normalmente estas vienen en pares una para cada sentido de rotación. Entre más grande sea la hélice y/o el Pitch más energía se tomará para hacerla girar.



Efecto del largo y Pitch de la hélice Las hélices generan impulso por la rotación y movimiento del aire. Cuando la hélice gira rápidamente, mueve más aire y por lo tanto genera mayor impulso. Al incrementar el largo de la hélice o el *pitch*, mayor cantidad de aire mueve y mayor energía es requerida

para girar, también se traduce en mayor capacidad de carga. Una hélice con un diámetro o *pitch* menor, puede girar más rápido, porque el motor no necesita impulsar más aire, por lo tanto genera menor energía. Estos tienden a trabajar más suavemente y se sienten con mayor respuesta a los controles.

Un *pitch* mayor en la hélice mueve más cantidad de aire, la cual puede crear turbulencia y estelas. Cuando las hélices giran menos rápido, uno puede encontrar la falta de sensación de rapidez. Pero genera mayor impulso y mayor velocidad máxima a expensas del mayor consumo de corriente, aun así la eficiencia del quadrirrotor es mayor (Mueller, 2017).

Los fabricantes utilizan estas medidas como referencia en la descripción de sus hélices. Hay dos formatos que se utilizan comúnmente

$$L \times P \times B \quad \text{o} \quad LLPP \times B$$

donde L (longitud de la hélice), P (pitch) y B (número de hojas que tiene la hélice).

2.6.11 Controlador de Velocidad del Motor

La siguiente parte requerida es el *Controlador de Velocidad del Motor* o *Electronic Speed Controller (ESC)*. Para cada motor en un multirrotor debe haber un ESC.

El ESC provee la adecuada corriente modulada a los motores, y estos producen rangos adecuados de giro para la elevación y maniobrabilidad. Son pocas cosas a considerar en un ESC en comparación de otros componentes dado que son partes muy estándares, solo las siguientes características a considerar

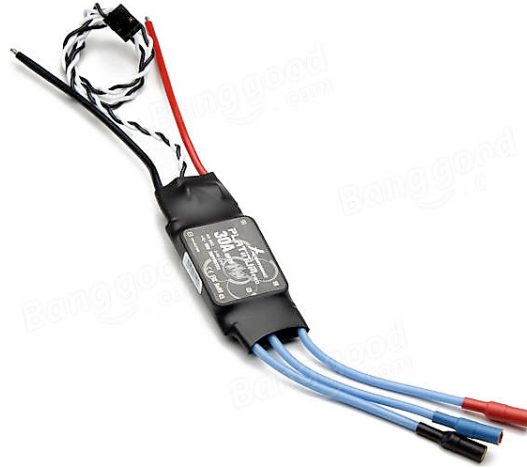


Figura 2.40: ESC de 30A (banggood.com, 2018)

- que tenga el firmware SimonK el cual es diseñado para trabajar con tiempos de alta precisión cuando otros rotores están involucrados.
- El circuito eliminador de batería, *battery circuit eliminator (BAC)*, el cual permiten al control de vuelo y componentes transductores conectarse al ESC en vez de la conexión directa a la batería. Esta característica puede o no tenerla.
- que tenga la capacidad para soportar la demanda máxima de amperes del motor.

En un extremo del ESC tiene dos cables (**rojo** y **negro**) que van conectados a la batería o a la placa de distribución de energía. También salen dos hilos conductores que son de control, a este cable llega la señal modulada del controlador de vuelo que define la velocidad del rotor. Si el ESC tiene la característica BAC, tendrá un tercer hilo. En el otro extremo hay tres conductores **azules** de salida (mayor calibre) que conectan a tres entradas de cada motor. Estos conductores **azules** pueden ser conectados de forma segura en cualquier orden. Si los motores giran en un sentido diferente al especificado en el modelo de la Figura 2.11, entonces se intercambian las conexiones de dos hilos del motor para invertir la secuencia trifásica (el sentido de giro). Ver Figura 2.41.

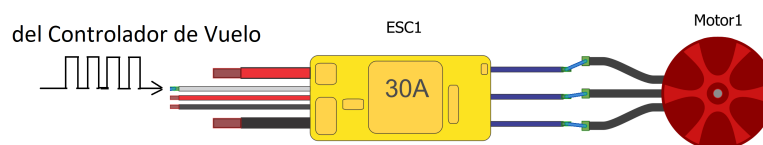


Figura 2.41: Diagrama simbólico de la conexión ESC-Motor

2.6.12 Controlador de Vuelo

Un controlador de vuelo (*flight controller (FC)*) es el cerebro de la aeronave. Es básicamente un circuito impreso con sensores que detectan cambios de orientación en el dron. También recibe instrucciones del usuario, y controla los motores para poder mantener el quadrirrotor al aire. La variedad en controladores de vuelo puede ser abrumadora para los principiantes. La mayoría de los FCs tienen un set básico de sensores con giroscopio y acelerómetro. Algunos podrían incluir algo más avanzado con el barómetro y magnetómetro (que es utilizado como brújula en la navegación).

El FC también es un concentrador para muchos otros periféricos, como el GPS, LED, sonares, etc. Para los que gustan de construir quadrirrotores, escoger un controlador de vuelo es más una opción personal en muchos sentidos, no por la diferencia a escoger de entre varios procesadores de PC en el mismo rango de poder, sino porque cada FC puede integrar opciones únicas del fabricante, para personalizarse o no.

En la mayoría de los quadrirrotores en el mercado comercial tienen integrado la *unidad de medición inercial (IMU)* al controlador de vuelo. También existe la IMU fabricada en placa independiente para usarse por microcontroladores de uso general o específico. Estas unidades se encargan de medir la orientación, velocidad y la fuerza de gravedad. Con estos datos, la electrónica de control ajusta las velocidades de los motores. La IMU está equipada con un giroscopio de 3-ejes, y un acelerómetro de 3-ejes. Esta combinación es conocida como una IMU 6DOF (6 Degrees Of Freedom). Las IMUs más recientes, tienen integrado magnetómetro de 3-ejes y altímetro siendo la IMU de 10DOF.

Firmware del Controlador de Vuelo

Además de la diferencia en hardware, los controladores de vuelo podrían utilizar un firmware diferente que se especializa en aplicaciones diversas. Los más recientes firmware pueden ser configurados vía software en una computadora, smartphone, o incluso desde el controlador de radio. La *personalización* o *tuning* es el término que se usa en un multirrotor de aficionado cuando se cambian los coeficientes del controlador PID, rangos, y otros parámetros de configuración. El *tuning* en los quadrirrotores hace que se puedan volar de una forma personalizada.

Sin embargo, la GUI (Graphics User Interface) y sus parámetros son diferentes de un firmware a otro, así que hay cierta curva de aprendizaje para llegar a cada firmware. La siguiente es una lista que en su mayoría están dedicados a los drones de carreras y acrobacias (Liang, 2013):

- Baseflight
- Cleanflight
- Betaflight
- iNav
- Raceflight
- KISS
- OpenPilot0
- LibrePilot
- TauLabs
- dRonin
- DJI Naza

Baseflight es uno de los primeros que se usaron ampliamente, de 32-bits, basados en el firmware Multiwii de 8-bits. En el presente, Baseflight no se ha actualizado y se considera como des-actualizado o muerto. A pesar de ello, fundó las bases para algunos de los firmwares que se utilizan actualmente.

Cleanflight este firmware se ramificó del Baseflight, tratando de ser la versión “limpia” para mejorar el mantenimiento y desarrollo futuro. Rápidamente el uso se esparció y convirtió en un firmware del que se dependía. A su vez y desde el principio, Cleanflight se ramificó en Betaflight e iNav. Muchas características se agregaron, probaron y retro-alimentaron en Cleanflight creando así el Cleanflight 2.0 para un mejor firmware de control.

Betaflight se derivó de Cleanflight enfocándose en nuevas características innovadoras y de desempeño de vuelo para los mini-quads, de ahí el nombre Betaflight. En la actualidad, es probablemente el software de control de vuelo más usado. Soporta una amplia gama de controladores de vuelo. La configuración por defecto vuela muy bien. Está equipado con características para competidores de alto nivel, pilotos de acrobacia y estilo libre así como para principiantes. Es de código-abierto (*open-source*) y es enriquecido por contribuyentes actualizando el código. A pesar de ser uno de los mejores firmware, es uno de los más complejos para configurar, aprender y usar.

iNav se enfoca más hacia la navegación y características de autonomía como misiones de vuelos a puntos de destino (*waypoints*) y funciones de retorno a la base (Return to Home - RTH). Se deriva de Cleanflight y también es *open-source* lo que hace haya buen rango de contribución y actualizaciones. Si se está interesado en este estilo de vuelo, esta es definitivamente una opción a explorar.

Raceflight se derivó de Betaflight y Cleanflight. Raceflight se enfoca en vuelos de carreras y acrobacia. Desde su creación fue de código cerrado (*closed-source*) y posteriormente los desarrolladores cambiaron a *open-source* y es reconocido por su muy amigable interfase de usuario y por su facilidad de configuración para los principiantes, pero con un reducido grupo de controladores de vuelo.

KISS fue desarrollado por Flyduino, el cual ha estado fabricando componentes multirrotor desde el 2011. Es de código cerrado y un tanto dependiente de la interfase de hardware. Es un firmware conocido y dependiente por los usuarios porque se desarrolla hacia donde progresan las necesidades de los usuarios. La interfaz gráfica es más sencilla que la de Betaflight y Cleanflight, haciendo que la configuración sea una de las más rápidas para poner el dron en el aire.

OpenPilot es uno de los primeros firmware *open-source* desarrollados para multirrotor que influenció uno de los mayores avances en las opciones de configuración en control de vuelo. OpenPilot se discontinuó y remplazó por LibrePilot en 2015.

LibrePilot se ramificó de OpenPilot, enfocándose en investigación y desarrollo para el uso de muchas diferentes aplicaciones, incluso la robótica. Este firmware no es muy popular en la comunidad de los mini-quads, ni tiene el soporte de muchas tarjetas controladoras como otros firmwares. Ha mostrado recientes actualizaciones y tiene un confiable equipo de desarrolladores. Es *open-source*.

TauLabs también derivó de OpenPilot. Se enfocó en investigación y desarrollo de drones. Orientado a profesionales, investigadores así como estudiantes, este proyecto no solo es para investigadores más especializados, sino ofrece una plataforma de entrada rápida y fácil en la configuración para cualquier grupo que haya usado un dron en su investigación. También tiene un limitado soporte técnico.

dRonin este firmware es un sub-derivado de OpenPilot. Tiene actualizaciones regulares, es *open-source* y tiene un equipo de desarrollo con la visión de mejorar el desempeño de las carreras y de acrobacia. Tiene un auto-ajuste para el control PID y un “wizard” de fácil y rápida configuración. Tiene un limitado grupo de controladores de vuelo pero con excelentes características.

DJI Naza para multi-rotores es un sistema de piloto automático diseñado para los serios entusiastas de los multi-rotores. Provee una excelente auto-nivelación y sostenimiento de altitud, lo que los hace fácil de volar, tanto para los profesionales como para aficionados. Esta versión Naza-M Lite puede ser instalada en una variedad de formas desde el quadrirrotor al hexarrotor.

Tiene buen desempeño. Es muy fácil de aprender. Requiere una cantidad mínima de programación y tiene una excelente estabilidad. Es uno de los más nutridos en control de vuelo. Ofrece características como RTH, altitud sostenida, vuelo libre, etc. Especialmente las funciones de altitud sostenida y de RTH son muy precisas y es un buen dispositivo para vídeo aéreo y FPV. Tiene la capacidad de volar hacia una dirección desde el punto de vista del operador sin importar la orientación del dron.

El Controlador Principal (MC) del Naza, es el cerebro del sistema, se comunica con todos los ESCs y con el transmisor RC para llevar a cabo la funcionalidad de piloto automático. Este dispositivo cuenta con entradas de control compatibles con las salidas individuales de un receptor de radio control común, ver [2.6.13](#).

El kit viene con unidad monitoreo de voltaje (Versatile Unit). Soluciona el problema de alta demanda en consumo de energía. Provee y monitorea el consumo del controlador y otros dispositivos electrónicos. Tiene un LED que indica los diferentes estados de operación y una interfase USB para la configuración de la unidad Naza y la actualización del firmware. También incluye un módulo GPS/brújula para el sensado de la posición y dirección.

2.6.13 Transceptor de Radio

El FC recibe y transmite las señales del control remoto (*Radio Control*) a través del *transceptor de radio* al cual está conectado. Hay diferentes tipos de controles con buenas características, una característica a tener en cuenta es el número de canales, desde cuatro hasta nueve.

Un canal es una entrada de control o salida de datos. Si el quadrirrotor no tiene canales, solamente tendría control para realizar un vuelo estacionario. Se necesita un mínimo de cuatro canales para mover un quadrirrotor,

dos disponibles para cada palanca del control. Cada canal adicional permite añadir controles para los accesorios (control de brazo articulado para la cámara - gimbal), habilitar o deshabilitar modo de vuelo (GPS, RTH, Failsafe), etc.

Un transceptor popular de bajo costo y consumo es el **nRF24L01 de 2.4GHz**. Son muy usados por su funcionalidad. El futuro de nuestro prototipo nos llevará a una mejor autonomía de vuelo, pero las pruebas y experimentación requerirán de un control inalámbrico. El NRF24L01 se conecta directamente a una interfase serial periférica (*SPI*) (que veremos más adelante) a través de los mismos pines en el controlador de vuelo. Este transceptor utiliza protocolos de comunicación SPI para transferir datos para y desde el controlador. Un modelo básico y económico es el que se muestra en la Figura 2.42, básicamente es el chip NRF24L01 y sus componentes necesarios para su funcionamiento. El otro modelo en la Figura 2.43 es más completo, aparte del NRF24L01 posee un circuito amplificador de potencia, uno de bajo ruido (LNA) además una antena con conector SMA (SubMiniature version A) que en conjunto le permiten lograr un rango de hasta 1000 metros.

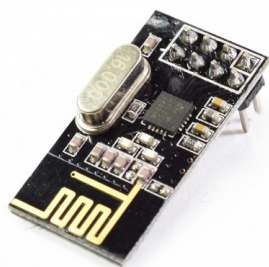


Figura 2.42: NRF24L01 tarjeta básica (Tutoriales, 2016)



Figura 2.43: NRF24L01 tarjeta de potencia y filtros (Tutoriales, 2016)

Este es el medio más popular para el control inalámbrico de un multirroto. Sin embargo, en este estudio no se utilizará.

2.6.14 Bluetooth®

La tecnología inalámbrica Bluetooth es un estándar popular en el área de las comunicaciones, y es uno de los campos más desarrollados de las tecnologías inalámbricas. La tecnología Bluetooth maneja la parte inalámbrica del canal de comunicación (Bluetooth, 2018). También se menciona, que opera en la banda ISM (*Industrial, Scientific and Medical*) de 2.4GHz, está equipada con opciones múltiples de radio que permite a los desarrolladores construir productos que alcancen los requerimientos únicos de conectividad del mercado. Transmite y recibe datos inalámbricamente entre dispositivos. Entrega los datos recibidos y recibe los datos a transmitir hacia y de algún sistema anfitrión a través de una interfase controladora anfitriona (*Host Controller Interface - HCI*) (Semiconductors, 2004). Dos de los HCI más populares son UART y USB.

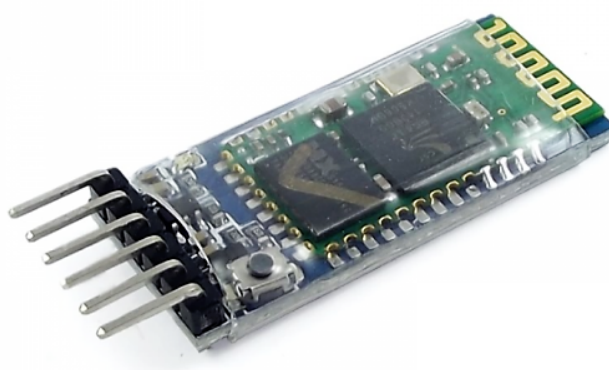


Figura 2.44: módulo HC-05 Bluetooth 2.0

Los dispositivos que mayormente utilizan esta tecnología, pertenecen a los sectores de las telecomunicaciones y la informática personal, como el smart-phone, laptop, desktop, impresora, cámara digital, bocina portátil, karaoke, manos libres, audífonos, reproductores MP3, vídeo-juegos, automotriz e industrial (Semiconductors, 2004). El HCI puede ser UART o USB. Sin embargo el protocolo de transporte en UART es más sencillo, que resulta en menos código, una solución de hardware con mayor costo-eficiencia y un alto desempeño del UART. El rendimiento de datos en la interfase UART es casi compa-

rable con la del USB. Se podría utilizar WIFI que tiene una elevada velocidad de transmisión, pero su protocolo es más complejo y para la aplicación en el dron-prototipo, es suficiente el uso de Bluetooth, el cual estuvo contemplado como la idea original.

Los dispositivos que incorporan este protocolo pueden comunicarse entre sí cuando se encuentran dentro de su alcance. Las comunicaciones se realizan por radiofrecuencia de forma que los dispositivos no tienen que estar alineados y pueden incluso estar en habitaciones separadas si la potencia de transmisión es suficiente. La especificación de Bluetooth define un canal de comunicación a un máximo 720 Kbit/s (1 Mbit/s de capacidad bruta) con rango óptimo de 10 m. Opera en la frecuencia de radio de 2.4 a 2.48 GHz con amplio espectro y saltos de frecuencia con posibilidad de transmitir en Full-Duplex con un máximo de 1600 saltos por segundo, se dan entre un total de 79 frecuencias con intervalos de 1 MHz; esto permite dar seguridad y robustez (Arduino, 2016). El hardware que compone el dispositivo Bluetooth está compuesto por dos partes:

- un dispositivo de radio, encargado de modular y transmitir la señal.
- un controlador digital, compuesto por una CPU, un procesador de señales digitales (DSP – Digital Signal Processor) llamado Link Controller (o controlador de Enlace) y de las interfaces con el dispositivo anfitrión.

El LC o Link Controller se encarga del procesamiento de la banda base y del manejo de los protocolos de la capa física; además, se encarga de las funciones de transferencia asincrónica, la codificación de audio y el cifrado de datos.

Un módulo popular y de bajo costo es el HC-05, compatible con Bluetooth 2.0 que puede usarse como maestro o esclavo. El HC-06 se usa como esclavo. Este es un módulo muy fácil de conectar: línea de tierra (**GND**), voltaje de alimentación de 5V (**VCC**), **TX**, **RX**, el pin **STATE** cambia a estado alto con la actividad del módulo, no es necesario usarlo. Este último pin solo lo tiene el HC-05. Ver Figura 2.45 y 2.46. Este módulo maneja lógica TTL de 3.3V, si se utiliza un microcontrolador con lógica de 5V, la señal de salida TX tendrá que ser acondicionada para enviarla a la entrada RX del módulo.



Figura 2.45: Pinout del Módulo Bluetooth

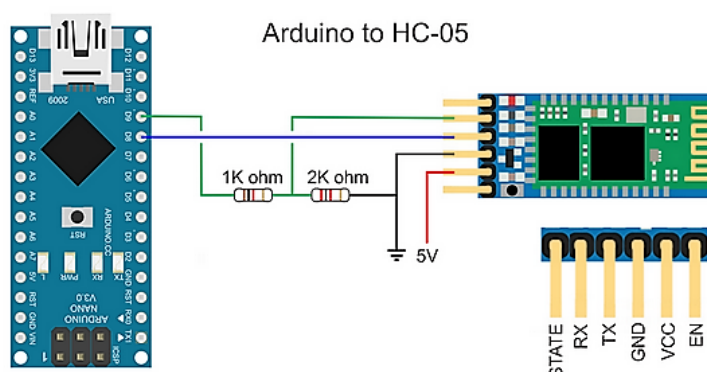


Figura 2.46: Conexión de un módulo Bluetooth

2.6.15 Módulo micro SD

micro SD (Secure Digital) es una tarjeta de memoria no-volátil. Se usa para almacenar archivos de datos, audio, vídeo, gráficas, etc. el módulo es, técnicamente una memoria flash. Tener una opción de almacenamiento removible es una gran ventaja. La mayoría de los microcontroladores tienen un limitado almacenamiento interno. Por ejemplo, el chip de Arduino Mega (Atmega2560) tiene solamente 4 Kbytes de memoria EEPROM. Si se quiere registrar datos, gráficas o audio, se necesitará al menos un Megabyte de almacenamiento, y probablemente la tarjeta con menos capacidad sea de 64 MB. El microSD es el mismo utilizado en las cámaras digitales, reproductores de MP3, memorias flash, comúnmente llamadas SD o microSD, ver Figura 2.48.

En ellas caben gigabytes de datos en un espacio más pequeño que una moneda. Son tan populares que se pueden comprar tanto en tiendas de electrónica, como de autoservicio. Los lectores de las tarjetas SD pueden ser leídas desde el smartphone, desktop y laptop pudiendo mover los datos de aquí para allá. Digamos, del GPS de un Arduino a la computadora (Lady-ada, 2017).



Figura 2.47: tarjeta microSD (Lady-ada, 2017)

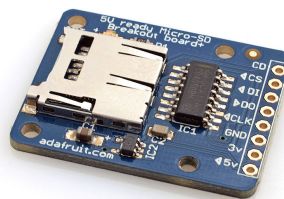


Figura 2.48: Módulo micro SD (Lady-ada, 2017)

Precauciones

Al trabajar con tarjetas SD, tener presente lo siguiente: Primero, son dispositivos estrictamente de 3.3V y el gasto de energía cuando hay escritura en la tarjeta puede ser considerablemente alto, hasta 100mA (o más). Esto significa que debe tener una buena fuente de 3.3V para la tarjeta. Segundo, manejar una lógica de 3.3V para la interfase a los pines. Las tarjetas SD son sensibles a los pines de comunicación - las nuevas tarjetas se disparan en la transición de estado lógico y requieren de una señal muy “cuadrada”, especialmente la entrada RX deberá recibir lógica de 3.3V, el conectarla a un voltaje mayor, por ejemplo de 5V, dañará el módulo. Los divisores de voltaje utilizando resistencias y un alambrado largo tendrán un efecto dañino en la velocidad de transición.

Cada tarjeta SD tiene una interfase SPI el cual es fácil usar en cualquier microcontrolador. El modo SPI requiere de cuatro pines lo que hace sencillo conectarlo. Existen los módulos microSD y los SD, ambos tienen la misma interfase, código, estructura, etc. todo es igual, solo cambia el tamaño.

Conexiones

Las tarjetas SD requieren mucha capacidad para transferencia de datos, ellas dan el mejor desempeño conectadas a los pines SPI de hardware en el microcontrolador. Estos pines son mucho más rápidos que intentando emular pines I/O de uso general para la interfase.

2.6.16 Sensor Ultrasónico

Los sensores ultrasónicos miden la distancia mediante el uso de ondas ultrasónicas. El cabezal emite una onda ultrasónica y recibe la onda reflejada que retorna desde el objeto. Los sensores ultrasónicos miden la distancia contando el tiempo entre la emisión y recepción. El dato calculado, se presenta en uno o varios puertos de comunicación del sensor, esto dependerá del fabricante, ver la Figura 2.49 y 2.50.

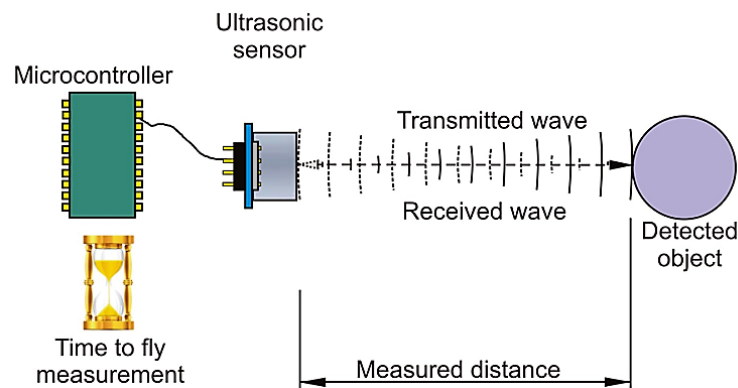


Figura 2.49: Emisión-Recepción de ondas ultrasónicas (Kelemen *et al.*, 2015)

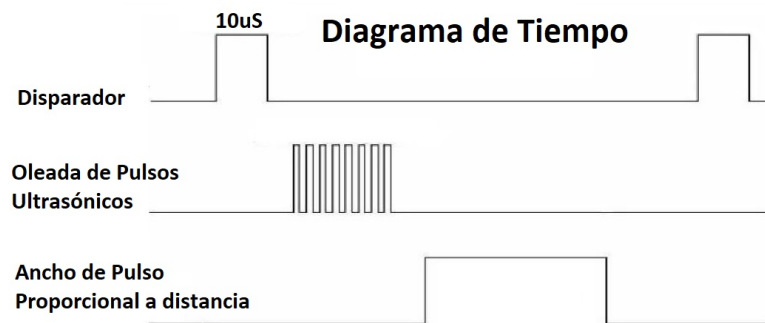


Figura 2.50: Diagrama de Tiempo de un sensor ultrasónico (Electrical Engineering, 2016)

Este tipo de sensores utiliza un elemento ultrasónico único, para la emisión y recepción. En un sensor ultrasónico de modelo reflectivo, un solo oscilador emite y recibe las ondas ultrasónicas, alternadamente. Esto permite la miniaturización en el cabezal del sensor.

Cálculo de la distancia

La distancia se puede calcular con la siguiente fórmula:

$$L = \frac{1}{2} \times T \times C$$

donde L es la distancia, T es el tiempo emisión-recepción, y C es la velocidad del sonido. (El valor se multiplica por 1/2 ya que T es el tiempo de recorrido ida y vuelta).

Los objetos transparentes son detectables dado que las ondas ultrasónicas pueden reflejarse en una superficie de vidrio o líquido y regresar al cabezal, incluso los objetos transparentes pueden ser detectados. La detección no se ve afectada por la acumulación de polvo o suciedad. La detección de presencia es estable, incluso para objetos tales como bandejas de malla o resortes.

Capítulo 3

Materiales y Métodos

Como se ha mencionado anteriormente, un dron tiene muchas aplicaciones. Pero antes de seleccionar el dron, primero se debe definir la aplicación. ¿Cuál es el uso que tendrá? En función de la respuesta se seleccionarán los componentes.

Las siguientes son las características deseables que un dron debe tener:

1. *Resistencia y Proteccion.* Las primeras veces que se vuela un dron, al menos una caída sufre. También que tenga guardas de proteccion en las hélices es un punto extra. Esta consideración es por seguridad de las personas y del aparato y para que la inversión realizada no quede en la basura en el primer intento.
2. *Localizable.* Que cuente con características que ayuden al usuario a localizar el dron. Una característica muy conocida es el de regreso al inicio, ya sea por la detección de carga baja de bateria o porque haya perdido comunicación con el radio control.
3. *Que sean modelos o componentes actuales.* Que estén comercialmente disponibles y si es posible saber, si son componentes que el proveedor vaya a discontinuar.
4. *Repuestos.* El tiempo promedio de una bateria es de 10 minutos. Por lo tanto, tener baterías extras es una forma de hacer prácticas de vuelo prolongadas. También tener al menos un par de hélices opuestas en caso

de algún accidente. Como se describió anteriormente en [2.6.10](#), son 2 pares distintos debido al sentido de rotación.

Al diseñar el dron, las principales necesidades a enfrentar para hacerlo funcional , son las siguientes:

1. *Chasis.*
2. *Sistema Eléctrico de Propulsión.*
3. *Sensores y Microcontrolador.*
4. *Software.*

Para este estudio, se ha seleccionado un dron tipo quadrirrotor por ser el más sencillo desde el punto de vista del modelado y control. Son simétricos y asimila el principio más simple de operación para controlar los movimientos de desplazamiento lateral, frontal, giro *roll*, *pitch*, *yaw* y de traslación, compensado con la cantidad de elementos que lo hacen económico, de fácil armado y porque en el mercado, hay una considerable oferta de kits para armado de quadrirrotores. En las siguientes secciones se especifican cada una de ellas.

3.1 Tamaño del Chasis

El primer paso es saber el tamaño del chasis. Esto determinará qué hélices y motores podrá usar el quadrirrotor. La distancia entre motores opuestos es el tamaño del chasis. La distancia centro a centro en milímetros del motor frontal izquierdo al motor trasero derecho, es el tamaño del chasis.

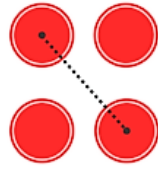


Figura 3.1: El tamaño del chasis se mide del centro de un motor al centro del motor opuesto ([LearningRC, 2017](#))

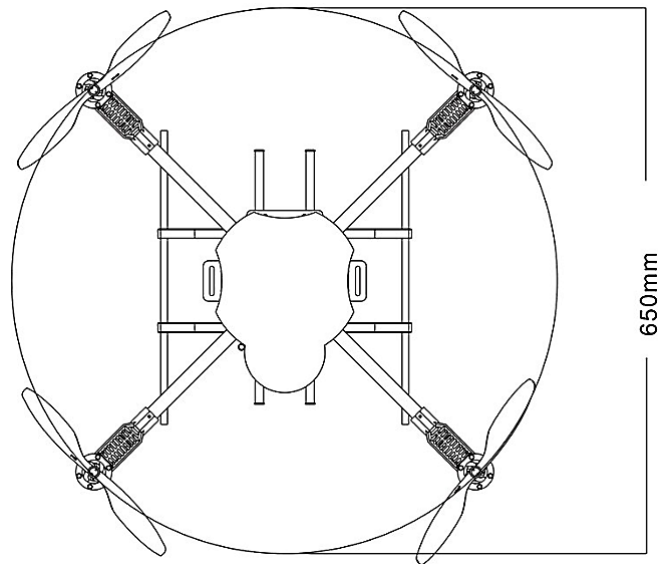


Figura 3.2: Chasis de 650mm para Quadrirotor ([unmannedtech, 2012](#))

Inicialmente se eligió un modelo con marco de aluminio, el *Aeroquad*, ver Figura 3.3, se seleccionaron los componentes, se hizo el trámite para su compra y el proveedor final, informó que no había en existencia y ni fecha para surtir. De hecho la URL ya no existe ([aeroquad, 2018](#)). Así que la selección del chasis, fue un modelo de fibra de carbono, el **Tarot 650** ver Figura 3.4.



Figura 3.3: Aeroquad - discontinuado (aeroquad, 2018)



Figura 3.4: Tarot 650: Chasis fibra de carbono (QWinOut, 2017)

Este chasis es parte de un kit con la mayoría de los componentes necesarios para armar el chasis y parte del sistema eléctrico de propulsión. Ver Tabla 3.1.

Tabla 3.1: Tarot 650 Kit (QWinOut, 2017)

Cantidad	Artículo
1	Chasis Tarot 650
1	Conector macho tipo T
1	Tarjeta de Distribución de voltaje
4	ESC marca HOBBYWING 30A Opto
4	Motores DC 750KV sin escobillas
2	Juegos de hélices TL2830 1455 y 1455R
12	Conectores bala de 3.5mm

3.2 Sistema Eléctrico de Propulsión

El Sistema Eléctrico de Propulsión, usa energía almacenada en una batería, que provee energía a los ESC para su funcionamiento y para alimentar los motores eléctricos que mueven las hélices. Comenzamos con una descripción de los elementos en el siguiente orden: primero la batería, ESCs y hélices. Después se utilizó una herramienta de simulación para determinar qué motores son adecuados. Lo que nos ayuda a prevenir una mala inversión, recordando que los motores son costosos.

3.2.1 Batería

Un chasis de 650mm es uno de los más grandes quadrirrotores en el mercado. Generalmente los modelos grandes son más estables, pueden utilizar hélices más grandes, con giros más lentos y por tanto, más eficientes, menor consumo de corriente, pero el costo es mayor. (jeffjblack, 2014).

Un ejemplo sería el modelo Tarot 650 Sport, compatible con hélices de 15 pulgadas, motores de 320KV y una batería LiPo 6S (22.2V) con capacidad de 4000 a 5000mAh, para un máxima fuerza de ascenso, (helipal, 2018). La Tabla 3.2 muestra algunas configuraciones que hay en el mercado.

Tabla 3.2: Modelos con chasis de 650mm

Chasis	Hélice $L \times P$	Motor KV	ESC	Voltaje	mAh
650	1555	320	40A	6S	4000-5000
650	1050	850	20A	3S/4S	3500-4000
650	1245	880	30A	3S/4S	4000-5000
650	1447	400	30A	3S	6000-8000
650	1555	480	30A	4S	5000

Tabla comparativa de modelos 650
(Alex (Unmanned Tech), 2013; helipal, 2018; hobby wing.com, 2018; rcgroups, 2017; Samaritakis, 2015)

En la Tabla 3.2, La hélice más próxima a la 1455 es la 1447, tiene el mismo largo y una diferencia en Pitch de .8 mm por debajo, lo que significa que la hélice del dron-prototipo es capaz de impulsar mayor cantidad de aire (mayor Pitch, mayor impulso de aire), tiene una configuración de voltaje de 3S = 11.1 *volts* y su rango es de 6000 a 8000 mAh, la mayoría de las configuraciones en la Tabla, se encuentran entre 3S y 4S, y controladores ESCs iguales al del prototipo (30A). En base a esta información se optó por una batería de 8000 mAh, con una configuración 3S (11.1V). Se adquirió una batería que cumplía con estas características ver Tabla 3.3 y Figura 3.5.

Tabla 3.3: Características de la Batería (HobbyStar, 2018)

Característica	Valor
Marca	HobbyStar
Capacidad	8000mAh
Voltaje	11.1V (3S)
Tasa de descarga	75C/100C
Corriente de Carga	2C
Calibre cables	12AWG
Conector de Balance	JST-XH
Dimensiones	138x46x35.5mm
Peso	475g



Figura 3.5: Batería LiPo

3.2.2 Hélices

Para el propósito de este estudio, se utilizaron hélices Tarot Carbon Fiber 1455 y su contra-parte 1455R ya que venían incluidas en el kit Tarot. La hélice es de 14 pulgadas de largo y 5.5 pulgadas de *Pitch* y dos hojas con un peso de 17.8 gramos por pieza ver Figura 3.6 y están balanceadas a mano y con precisión (eCalc, 2018)



Figura 3.6: Hélices Tarot Carbon Fiber 1455 y 1455R (eCalc, 2018)

Están hechas de tela de fibra de carbono y el diseño de las puntas de la hélice (blended winglet) mejoran la eficiencia hasta un 13%. Ver Figura 3.7.



Figura 3.7: Punta de ala mezclada (Blended winglet) (eCalc, 2018)

3.2.3 ESC

Como ya se dijo anteriormente, la mayoría de los ESC de las configuraciones obtenidas (Tabla 3.2) son de 30A y convenientemente en el kit adquirido tuvimos un juego de 4 ESCs de 30A, Ver Figura 3.8. Este elemento se consideró un factor constante en las pruebas con **eCalc**. Tiene las siguientes características:

Tabla 3.4: Características del ESC (banggood.com, 2018)

Característica	Valor / Descripción
Marca	Hobbywing
Salida	30A continua 40A disparo hasta 10 segundos
Voltaje de Entrada	2-6 celdas a de litio ó 5-18 celdas de batería NiMH
BEC	No disponible
Frecuencia muestreo Aceleración	50Hz a 432Hz
Tamaño	55mm (Largo) × 25mm (Ancho) × 12mm (Alto)
Peso	31g



Figura 3.8: Electronic Speed Controller - ESC (banggood.com, 2018)

Los ESCs tienen un microprocesador integrado de alto desempeño, son optimizados para aplicaciones multirrotor, son compatibles con todos los tipos de motor. La respuesta de aceleración es rápida, suave, lineal y precisa. Tiene protección contra cierres por bajo voltaje, protección por sobre-calentamiento

y protección por pérdida de señal de aceleración. (banggood.com, 2018).

En la Tabla 3.5 podemos ver cuales componentes tenemos caracterizados.

Tabla 3.5: Variables de Entrada a **eCalc**

Chasis de 650mm	punto (1) cubierto.
peso total:2000 gramos.	
Batería de 8000mAh	punto (2) parcialmente cubierto.
Hélices de 1455	
ESC de 30 Amperes	

Para completar el sistema de propulsión, fue necesario definir las características del motor y usando el **eCalc**.

3.2.4 ¿Qué es *eCalc*?

eCalc es un programa en la web que toma los datos de diseño del multirrotor y ayuda en la selección de un motor adecuado. **eCalc** se basa en física común y un modelo matemático que permite simular las características de vuelo resultantes del multicóptero, simplemente seleccionando un *setup* de entre la base de datos. **eCalc** es una herramienta que puede contestar preguntas como ¿este motor servirá? o ¿cuál será el tiempo de vuelo? o algo más básico, ¿se elevará el dron?. También puede utilizarse para un cambio virtual de una parte del dron y “ver” cómo funciona. Por ejemplo, ¿qué pasa si cambio las hélices de 1340 a 1455?.

A pesar de que existen compañías que desarrollan modelos multi-rotores listos para volar, muchas otras marcas ofrecen componentes para que uno pueda construir a la medida y resultan en modelos muy diferentes de un usuario a otro.

En la página del simulador de *xcopterCalc* - *Calculador de Multicópteros* ([Mueller, 2018](#)) hay cinco secciones con campos de datos para ingresar val-

3. MATERIALES Y MÉTODOS

3.2 Sistema Eléctrico de Propulsión

ores diferentes, uno a la vez, realizar el cálculo y observar el comportamiento de la variable. Ver Figura 3.9:

- General
- Celdas Bateria
- Variador (ESC)
- Motor
- Hélice

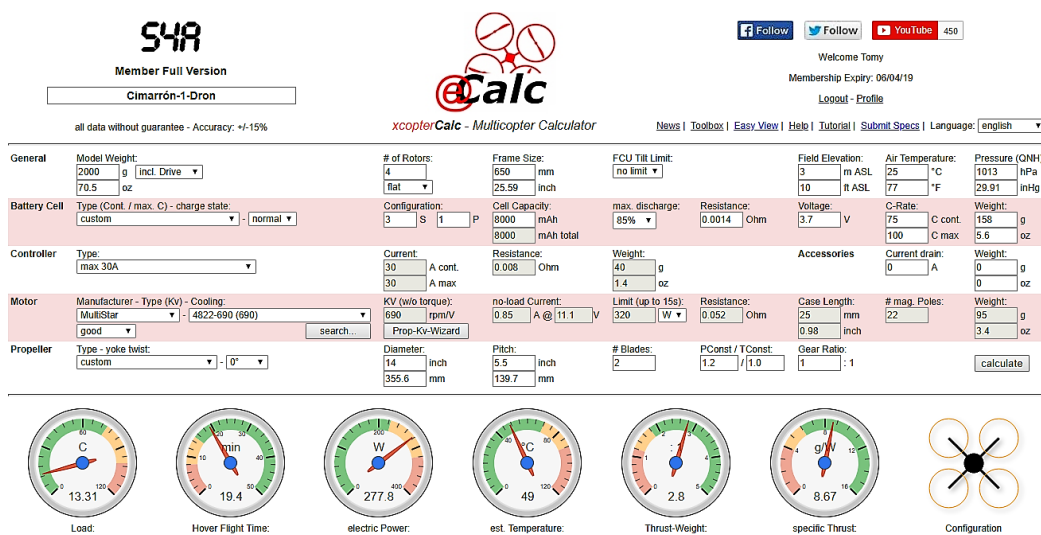


Figura 3.9: eCalc - Multicópteros (Mueller, 2017)

En la sección **General** y de acuerdo a la Tabla 3.5 se ingresó como peso total del modelo 2000 gramos un cantidad estimada “con Motorización”. 4 rotores simples. Tamaño del armazón (chasis) es de 650mm. Se deja el parámetro de inclinación “sin límite”. Y una altura promedio en Mexicali de 3 metros sobre el nivel del mar (Ayuntamiento de Mexicali, 2010). La presión atmosférica se dejó en los 1013 hPa (hectopascasles) como se encontraba. Ver Figuras 3.10.

3. MATERIALES Y MÉTODOS

3.2 Sistema Eléctrico de Propulsión

Peso del modelo:

2000	g	con Motorización ▼
70.5	oz	

Nº de rotores:

4
simple ▼

Tamaño del armazon:

650	mm
25.59	inch

Limited de inclinación de la FCU:

sin límite ▼

(a) Peso

(b) Tamaño

Altura del campo

6	m ASL
20	ft ASL

Temp. aire

25	°C
77	°F

Presion (QNH):

1013	hPa
29.91	inHg

(c) Presión Atmosférica

Figura 3.10: Sección General de eCalc (Mueller, 2017)

En la sección **Celdas de Batería** se ingresó tipo *personalizada*, con carga *llena*, configuración de **3S-1P** y una capacidad de 8000 mAh. Una resistencia de 0.014 Ohms, el peso que se ingresa es por celda, así que la batería de 474 gr. por celda queda en 158 gr. El voltaje también es por celda, igual a 3.7V y su capacidad de descarga se indica en 100C continua y de pico. Ver Figuras 3.11.

Tipo (Cont. / max. C) - nivel de carga:

personalizada ▼	llena ▼
-----------------	---------

Configuración:

3	S	1	P
---	---	---	---

Capacidad por celda:

8000	mAh
8000	mAh total

descarga max.

85% ▼

Resistencia:

0.0014	Ohm
--------	-----

(a) Tipo y Configuración

(b) Capacidad por Celda

Voltaje:

3.7	V
-----	---

capacidad C de descarga:

100	C continua
100	C de pico

Peso:

158	g
5.6	oz

(c) Voltaje y C

Figura 3.11: Sección Celdas de Batería en eCalc (Mueller, 2017)

En la sección **ESC** se seleccionó el ESC genérico que viene descrito como “max 30A”. Los demás valores: Amperes continuos, Amperes máximo, resistencia y peso se establecen por defecto y no se pueden editar. En los Accesorios, se dejan en cero todos los valores puesto que nuestra aplicación no requiere un consumo significativo y el peso ya se consideró en el peso total del dron. Ver Figuras 3.12.

3. MATERIALES Y MÉTODOS

3.2 Sistema Eléctrico de Propulsión

Tipo: max 30A ▼	Accesorios	Consumo de corriente: 0 A	Peso: 0 g 0 oz
--------------------	------------	------------------------------	----------------------

(a) ESC genérico

(b) Accesorios

Figura 3.12: Sección Controlador de Velocidad del Rotor en eCalc (Mueller, 2017)

En la sección **Hélice** se seleccionó una hélice personalizada en 0° con un diámetro de 14 pulgadas y Pitch (Paso) de 5.5 pulgadas. El número de palas o navajas de la hélice se establece en dos. Ver Figuras 3.13.

Tipo de hélice: personalizada ▼	0° ▼	Diámetro: 14 inch 355.6 mm	Paso: 5.5 inch 139.7 mm	número de palas: 2	Const de Potencia/Empuje: 1.2 / 1.0	Gear Ratio: 1 : 1
------------------------------------	------	----------------------------------	-------------------------------	-----------------------	--	----------------------

(a) Largo y Pitch

(b) Número de Navajas o palas

Figura 3.13: Configuración de la Hélice

Con esta información capturada se puede ejecutar el *Asistente KV hélice* al pulsar el botón que se encuentra en la segunda columna de la sección **Motor**. Este asistente es la “calculadora” que arroja las características deseables para el motor. Al pulsar el botón, se despliega una ventana emergente, ver Figura 3.14.

Wizard - eCalc - the most reliable RC Calculators on the Web for electric Motors - Goo...
Es seguro | <https://www.ecalc.ch/wizard.php?m=2000&r=4&config=flat&frame=650&pv=11.1...>

Asistente KV hélice

Peso total: 2000 g

Nº de rotores: 4 flat ▼

Tamaño del armazon: 650 mm

Batería - Tensión nominal: 11.1 V

Hélice - Diámetro: 14 inch max. 18"

Hélice - Paso: 5.5 inch max. 9.2"

Hélice - número de palas: 2

Calcular

KV recomendados: 570 ... 830 rpm/V encontrar...

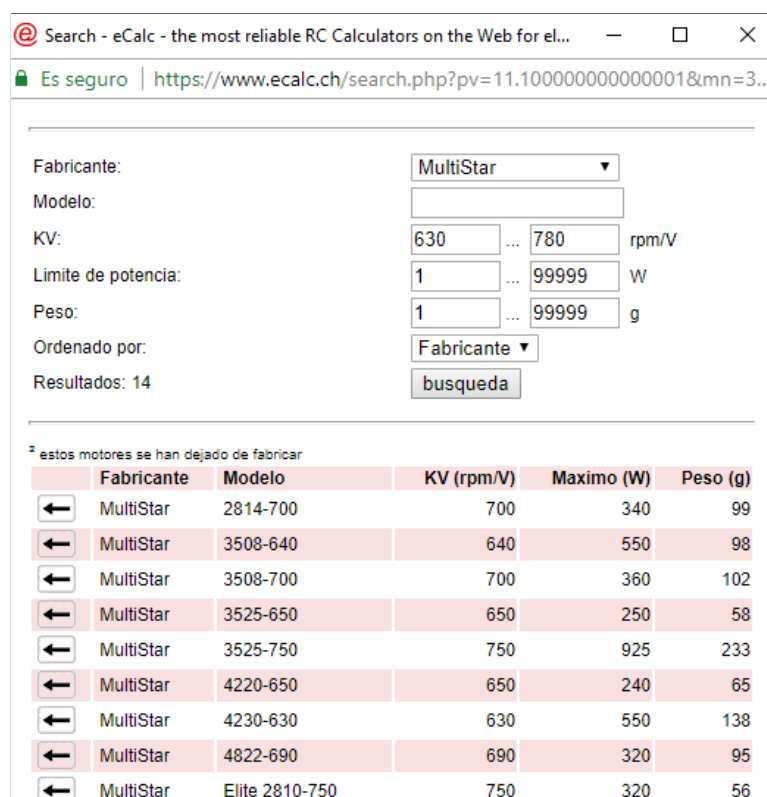
motor de al menos: 245 ... 425 W+

variador de al menos: 25 ... 45 A+

copyright by Solution for All Markus Müller
www.ecalc.ch email: info@ecalc.ch

Figura 3.14: Asistente KV hélice (Mueller, 2017)

Al abrir el asistente se muestran los datos previamente capturados y se presiona el botón “Calcular”. En la parte inferior se muestran los rangos de los parámetros deseados para el modelo. Al presionar el botón “encontrar...” se mostrará la lista de todos los motores de las marcas de los fabricantes que están en la base de datos de **eCalc**. Se seleccionó la marca **MultiSTAR** Modelo 4822-690. En la ventana principal se presiona el botón “Calcular” que se encuentra en el extremo derecho de la sección **Hélice**. Se ejecuta el algoritmo iterativo de **eCalc** para estimar los datos que se muestra en la Figura 3.15.



Search - eCalc - the most reliable RC Calculators on the Web for el...
 Es seguro | <https://www.ecalc.ch/search.php?pv=11.100000000000001&mn=3...>

Fabricante: MultiStar
 Modelo:
 KV: 630 ... 780 rpm/V
 Limite de potencia: 1 ... 99999 W
 Peso: 1 ... 99999 g
 Ordenado por: Fabricante
 Resultados: 14
 busqueda

² estos motores se han dejado de fabricar

	Fabricante	Modelo	KV (rpm/V)	Maximo (W)	Peso (g)
←	MultiStar	2814-700	700	340	99
←	MultiStar	3508-640	640	550	98
←	MultiStar	3508-700	700	360	102
←	MultiStar	3525-650	650	250	58
←	MultiStar	3525-750	750	925	233
←	MultiStar	4220-650	650	240	65
←	MultiStar	4230-630	630	550	138
←	MultiStar	4822-690	690	320	95
←	MultiStar	Elite 2810-750	750	320	56

Figura 3.15: Lista de modelos compatibles según las especificaciones (Kelly, 2014)

En (eCalc, 2014; Kelly, 2014) se encuentra información más detallada para el uso de la calculadora. De la Figura 3.16, se puede decir que:

- el índice de descarga de la batería es continua (C-rate) y aceptable con un valor de 13.31 veces el valor C, recordar que la batería tiene una 100C.
- el tiempo de vuelo estacionario es de 19.4 minutos, valor aceptable.
- la potencia eléctrica con el motor al máximo es de 277.8 Watts, este parámetro indica una condición crítica.
- la temperatura estimada del motor al máximo es de 49°C el cual se encuentra en un rango normal de temperatura.
- factor Impulso-Peso es de 2.8:1.

- el Impulso específico que genera la hélice es un indicador de la eficiencia total del sistema en vuelo estacionario.

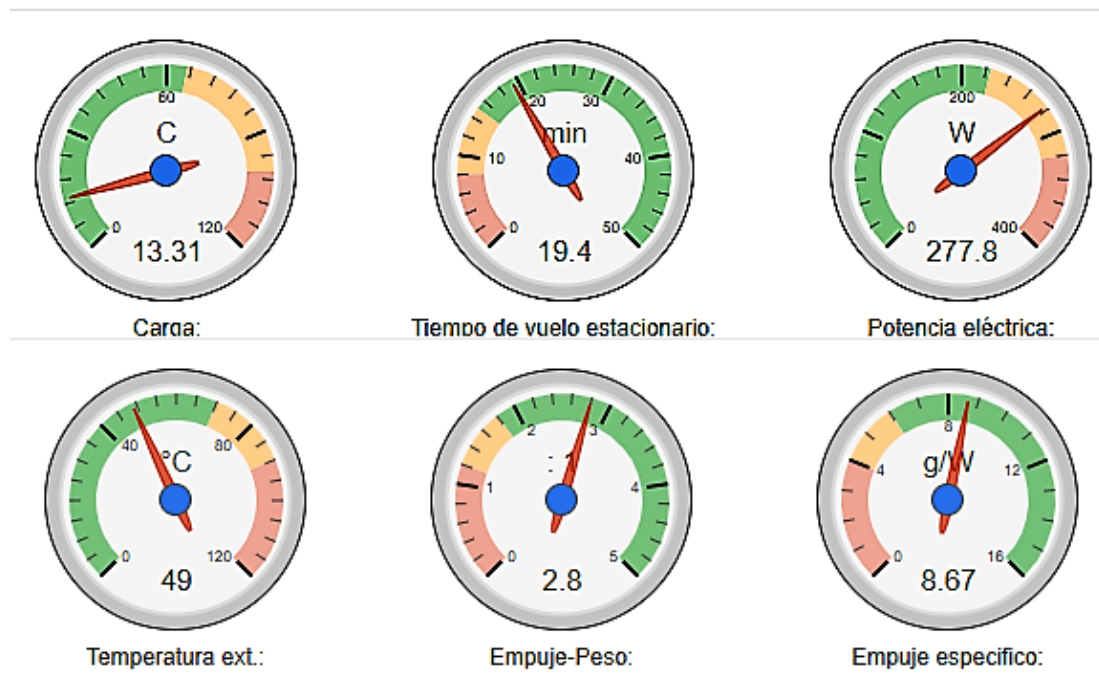


Figura 3.16: Estimaciones de los parámetros principales en eCalc (Kelly, 2014)

También se muestran otros datos complementarios en las Figuras 3.17 y 3.18. Para la interpretación de estos datos se puede consultar (eCalc, 2014).

3. MATERIALES Y MÉTODOS

3.2 Sistema Eléctrico de Propulsión

Observaciones:

Batería		Motor a eficiencia óptima		Motor al Máximo	
Carga:	13.31 C	Corriente:	12.26 A	Corriente:	26.61 A
Voltaje:	10.65 V	Voltaje:	10.80 V	Voltaje:	10.44 V
Tensión nominal:	11.10 V	Revoluciones*:	6971 rpm	Revoluciones*:	6167 rpm
Energía:	88.8 Wh	Potencia eléctrica:	132.4 W	Potencia eléctrica:	277.8 W
Capacidad total:	8000 mAh	Potencia mecánica:	115.9 W	Potencia mecánica:	231.4 W
Capacidad usada:	6800 mAh	Eficiencia:	87.6 %	Potencia-Peso:	555.7 W/kg
Tiempo min de vuelo:	3.8 min				252.1 W/lb
tiempo medio de vuelo:	13.5 min			Eficiencia:	83.3 %
Tiempo de vuelo estacionario:	19.4 min			Temperatura ext.:	49 °C
Peso:	474 g				120 °F
	16.7 oz				
				Medidas de potencia	
				Intensidad:	106.44 A
				Voltage:	10.65 V
				Potencia:	1133.6 W

Figura 3.17: Resultados del cálculo según especificaciones primera página de eCalc (Kelly, 2014)

Motor @ Hover		Motorización Total		Multicóptero	
Corriente:	5.26 A	Peso de la Motorización:	1115 g	Peso total:	2000 g
Voltaje:	10.97 V		39.3 oz		70.5 oz
Revoluciones*:	3264 rpm	Empuje-Peso:	2.8 : 1	máximo peso adicional:	2890 g
Acelerador (log):	33 %	Corriente en estacionario:	21.04 A		101.9 oz
Acelerador (lineal):	49 %	Pot(entrada) en estacionario:	233.5 W	inclinación máxima:	66 °
Potencia eléctrica:	57.7 W	Pot(salida) en estacionario:	193.6 W	velocidad máxima:	38 km/h
Potencia mecánica:	48.4 W	Eficiencia en estacionario:	82.9 %		23.6 mph
Potencia-Peso:	116.8 W/kg	Corriente al máximo:	106.46 A	Trepada estimada :	7.5 m/s
	53 W/lb	Potencia(entrada) al máximo:	1181.7 W		1476 ft/min
Eficiencia:	83.9 %	Potencia(salida) al máximo:	925.4 W	Area total del disco:	39.73 dm²
est. Temperatura:	30 °C	Eficiencia al máximo:	78.3 %		615.81 in²
	86 °F			Fallo del motor:	
Empuje específico:	8.67 g/W				
	0.31 oz/W				

Figura 3.18: Resultados del cálculo según especificaciones segunda página de eCalc (Kelly, 2014)

3.2.5 Motores



Figura 3.19: Motor DCBL Outrunner modelo Turnigy MultiSTAR 4822-690 (hobbyking.com, 2018)

De acuerdo a 3.2.4, el modelo seleccionado fue el Turnigy MultiSTAR 4822-690, motor sin escobillas *outrunner* su principal virtud es que tiene mucha potencia en muy poco peso. Especialmente diseñados para aplicaciones multirrotores. Estos motores son potentes, no sufren de sobre-calentamiento y los resultados del simulador indican que el tipo de motor es adecuado para la configuración del dron (hobbyking.com, 2018), sus especificaciones son:

- KV (RPM / V): 690
- Potencia máxima: 320W
- Corriente de trabajo: 16 amperes
- Corriente máxima (10s): 22 amperes
- Corriente sin carga: 0,85 amperes
- Resistencia interna: 0.052 ohm
- Número de polos: 22
- Peso: 95 gramos

3.3 Sensores y Microcontrolador

3.3.1 Bluetooth

El módulo Bluetooth seleccionado es un HC-05 configurado en modo esclavo, solamente puede aceptar conexiones. El Smartphone Android iniciará la conexión con el módulo Bluetooth. El microcontrolador Arduino Due iniciará una

conexión serial asincrónica a través del puerto UART1, pines 18 (TX1) y 19 (RX1). Donde el módulo creará un enlace de comunicación entre el Arduino y Smartphone. Es común ver el pin RX del módulo Bluetooth que proviene del TX del microcontrolador a través de un divisor de voltaje debido a la lógica de 3.3V utilizada en el módulo y la lógica de 5V utilizada en la mayoría de los microcontroladores, sin embargo, el Arduino Due maneja la lógica de 3.3V en todos su puertos de datos por este motivo en la Figura 3.21 y 3.20 la conexión es directa.

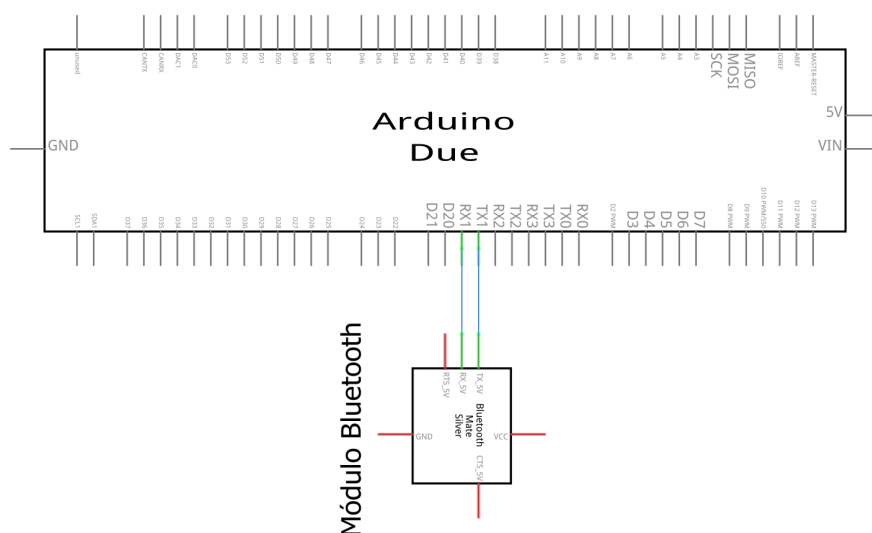


Figura 3.20: Diagrama esquemático de conexión Arduino-Bluetooth a través de UART

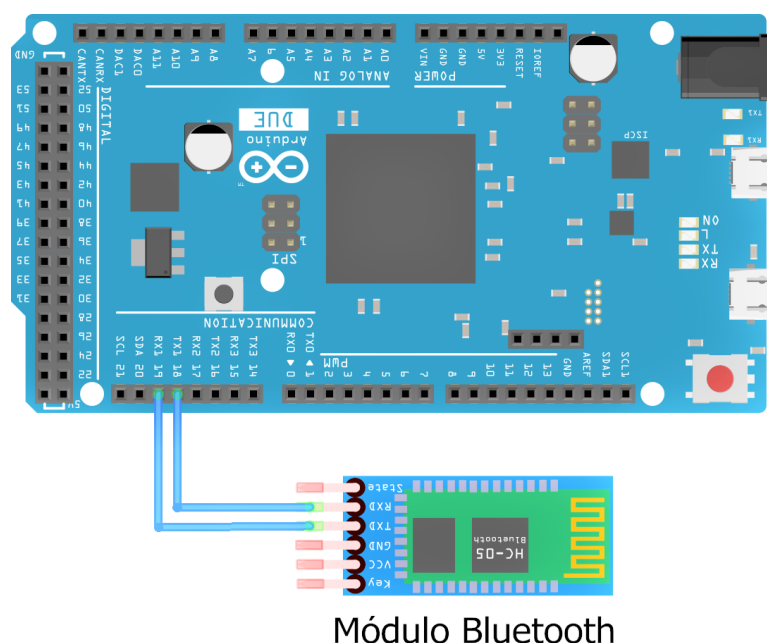


Figura 3.21: Diagrama simbólico de conexión Arduino-Bluetooth a través de UART

3.3.2 micro SD

Teniendo en cuenta las precauciones de 2.6.15, el alambrado se hizo lo más corto posible, y se evitó divisores de voltaje con resistencias para la líneas de lógica de 3.3V. Este módulo tiene integrado un cambiador de nivel de lógica, el CD74HC4050 el cual tiene un típico retardo de propagación de $\sim 10ns$. Esto le dá la característica al módulo de alimentar con 5V e internamente con lógica 3.3V (Lady-ada, 2017). El dron tiene montado un módulo micro SD para el registro de temperatura, tiempo y la posición global de la medición.

Se necesitó también un cuarto pin para el “Select-Slave” (SS). Normalmente este es el pin 10, aunque se puede usar otro. Lo importante para el SS, es que se configure como pin de salida, se use o no. Esto es requerido por la librería Wire.h para el uso de la comunicación SPI. En Arduino Due, se conecta al socket central (puerto) SPI de 3×2 pines, ver Figuras 3.22 y 3.23.

- Conectar el pin 5V al pin 5V del Arduino.
- Conectar el pin GND al pin GND del Arduino.

- Conectar el pin **CLK** al pin **CLK** del Arduino.
- Conectar el pin **DO** (data out) al pin **MISO** del Arduino.
- Conectar el pin **DI** (data in) al pin **MOSI** del Arduino.
- Conectar el pin **CS** al pin **GND** del Arduino.

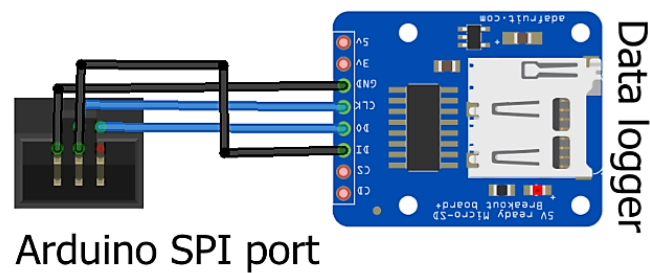


Figura 3.22: Conexión Micro SD al puerto SPI (Lady-ada, 2017)

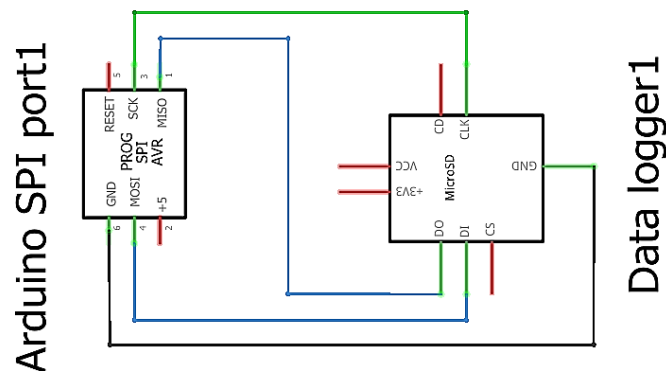


Figura 3.23: Conexión Arduino a micro SD

3.3.3 Sensor Ultrasónico

El XL-MaxSonar-EZ (Sonar) tiene una emisión ultrasónica de potencia así como auto-calibración en tiempo real para condiciones cambiantes (temperatura,

voltaje y ruido acústico o eléctrico) para asegurar que obtengamos la mejor recepción de datos por cada lectura tomada. Este sensor tiene un requerimiento de bajo consumo de energía de 3.3V - 5.5V y en su operación provee rangos de lectura muy cortos y largos, en un encapsulado pequeño. Este sensor detecta objetos desde 0 ~ 765cm y la información que provee es desde los 20-cm hasta los 765-cm con una resolución de 1-cm. Los objetos localizados entre los 0-20-cm normalmente arrojan un valor de 20-cm. Tiene una frecuencia de muestreo de 10 Hz. Todas las interfases de comunicación de este módulo están activas simultáneamente. El sensor opera a 42 KHz. Tiene un bus serial asincrónico configurado a 9600 baudios, 8 bits, 1 bit de terminar y sin bit de paridad. Tiene una señal de salida analógica con una resolución de $(V_{cc}/1024)/cm$. Las lecturas de este sensor no se afectan por el color u otras características visuales del objeto detectado. También cuenta con una salida digital donde su ancho de pulso (en alto) es proporcional a la distancia del sensor al objeto detectado (MaxBotix Inc., 2018).

Conexiones

- **Pin 1-BW** se deja una conexión abierta o en alto para utilizar salida serial en el Pin 5. Cuando el Pin 1 se mantiene en bajo, la salida del Pin 5 envía un pulso (en vez de datos seriales).
- **Pin 2-PW** este pin arroja un ancho de pulso que representa la distancia. Para el cálculo, se utiliza el factor de escala de $58\mu S$ por cm.
- **Pin 3-AN** esta salida arroja un voltaje analógico con un factor de escala de $\frac{V_{cc}}{1024}$ por cm. Si se alimenta con $V_{cc} = 5V$ serían $\sim 4.9mV/cm$ y para $V_{cc} = 3.3V$ serían $\sim 3.2mV/cm$. Su hardware limita el rango máximo en esta salida de $\sim 700cm$ a 5V y $\sim 600cm$ a 3.3V
- **Pin 4-RX** este pin tiene una resistencia pull-up interna, dado que es un dispositivo de salida, si este pin se mantiene bajo, el sensor dejara de operar.
- **Pin 5-TX** cuando el Pin 1 está abierto o en estado alto, entrega una señal serial asincrónica con el formato del RS-232, solo que los voltajes son de

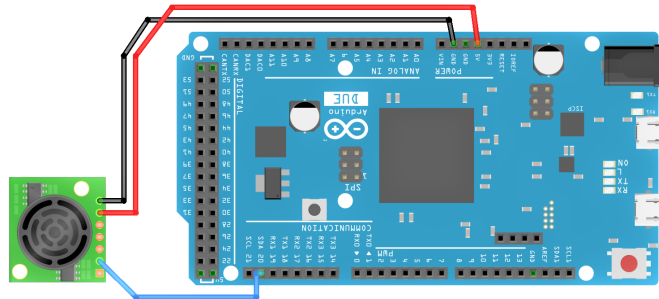


Figura 3.25: Conexión Arduino-Sonar - Diagrama simbólico

3.3.4 Naza M Lite

El controlador de vuelo en el dron es el **Naza M Lite** con GPS. Tiene dos puertos de comunicación serial, el UART para el GPS, S-BUS para otro tipo de receptores de RC y que no veremos en este escrito. Tiene una IMU integrada que consiste de un acelerómetro de 3-ejes, un giroscopio de 3-ejes y un barómetro para el sensado de la *actitud* y la *altitud*. Tiene 8 señales de salida de las cuales 6 son para enviar la señal a los ESC (M1-M6) que a su vez enviarán señales a los motores, salidas F1 y F2 para control del estabilizador (gimbal) de la cámara (opcional), ver Tabla 3.6.

Tabla 3.6: Naza M Lite - Tabla del Pinout

Nombre del Puerto	Tipo Señal	Entrada/Salida	Conexión
A (Aileron)	PWM	Entrada	Radio/Control
E (Elevator)	PWM	Entrada	Radio/Control
T (Throttle)	PWM	Entrada	Radio/Control
R (Rudder)	PWM	Entrada	Radio/Control
U (Gear)	PWM	Entrada	Radio/Control
LED bus	Serial	E/S	Versatile Unit
EXP bus	UART	E/S	Unidad GPS
M1-M6	PWM	Salida	ESC 1-6
F1-F2	PWM	Salida	Gimbal

Para el uso de este controlador se deben tener en cuenta las siguientes consideraciones:

- Para modelos mayores a los 650mm o con carga pesada, se recomienda el uso de otro controlador de vuelo. El dron-prototipo es de exactamente los 650mm así que por el alto costo de la recomendación, nos quedamos con el Naza-M Lite.
- se aseguró que el centro de gravedad fuera en el centro de la cruz formada por el quadrirrotor.
- cuando se instaló el MC, se aseguró estar pegado en el centro de gravedad y en la posición correcta (no volteado de cabeza). También se aseguró que el MC estuviera paralelo al horizonte de la aeronave.
- se aseguró que el MC estuviera apuntando al frente (puertos ESC al frente). El frente del quadrirrotor (su nariz) depende de la configuración de la aeronave en el software de Asistencia del Naza. Para nuestro propósito, la configuración que se utilizó fue en X, y la nariz de la aeronave quedó entre los motores 1 y 2, ver Figura 3.27



Figura 3.26: Controlador o estabilizador de vuelo modelo Naza M Lite

Este controlador tiene la capacidad de estabilizar el dron en vuelo estacionario y mantenerse en su lugar a menos que Arduino le envíe un comando (PWM) mayor al mínimo, que pase por un filtro y se lo re-envíe.

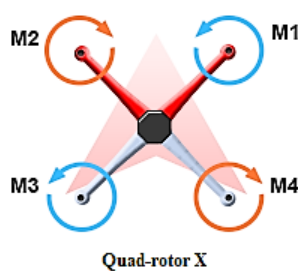


Figura 3.27: Dron en configuración "X"

3.3.5 El Microcontrolador

El microcontrolador se encarga de gestionar y procesar toda la información de otros dispositivos: Bluetooth, ESCs, motores, micro SD, GPS, magnetómetro, sonar y el controlador de vuelo Naza. Para determinar qué microprocesador

utilizar, primero se enumeraron los sensores y controladores, que se integrarían al microcontrolador para saber los requerimientos de intercomunicación. Estos deberán estar empatados con el objetivo general definido en 1.2 y los objetivos específicos definidos en 1.2. La Tabla 3.7 muestra los elementos periféricos y sus requerimientos de comunicación.

Tabla 3.7: Elementos Periféricos Requeridos

Elemento Periférico	Protocolo de Comunicación
Módulo Bluetooth	Bus UART para comunicación con Smartphone.
Módulo GPS	Bus UART solo RX para lectura del GPS.
Micro SD	Bus SPI para registro de datos
Altímetro	I2C para lecturas de altitud.
Naza M Lite	5 Pines de salida PWM para el <i>Naza M Lite</i>
Sensor Ultrasónico	1 Pin digital de entrada.
Micro SD	1 Pin digital de salida para librería del micro SD.
Software	funciones trigonométricas integradas.
Librerías	Librerías necesarias para la comunicación.

Primeramente se descartó el uso de PICs (Peripheral Interface Controller de Microchip Technology) ya que no ofrecen un buen entorno de programación. La documentación y librerías no tienen el soporte y la popularidad de un Arduino o un Raspberry Pi. En los últimos, la comunidad de usuarios ha crecido ampliamente a nivel global y dirigido a todos los niveles de educación. Otra característica de estas dos plataformas es que son de *Código Abierto* (*Open Source*), a diferencia de los PICs. Entonces, la decisión estuvo entre estos dispositivos: Arduino Due y Raspberry Pi.

Arduino Due

El Arduino Due es una placa electrónica basada en el microcontrolador Atmel SAM3X8E ARM Cortex-M3. Es la primera placa Arduino basado en un microcontrolador con núcleo ARM de 32-bits. Tiene 54 entradas/salidas digitales,

de las cuales 12 se pueden utilizar como salidas PWM, 12 entradas analógicas, 4 Bus UARTs (puertos seriales de hardware), un reloj de 84 MHz, una conexión USB OTG, 2 salidas analógicas reales, 2 Bus TWI, un puerto JTAG, 1 puerto SPI, un botón de RESET y otro de ERASE (arduino.cc, 2018).

Como nota, las siglas TWI significan *Two Wire Interface*, Interfase de dos hilos, que es idéntico a I2C. Este es el término utilizado por Atmel y otras compañías para evitar conflictos de marca registrada relacionada con I2C.

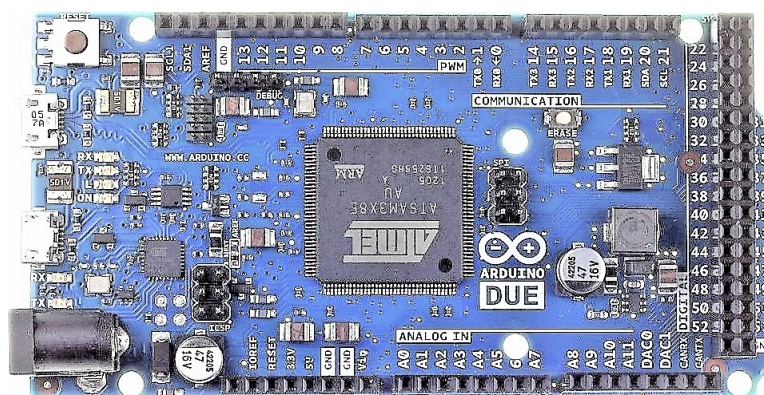


Figura 3.28: Arduino Due (arduino.cc, 2018)

Raspberry Pi 3

El Raspberry Pi 3 es la tercera generación de Raspberry Pi. Reemplazó a Raspberry Pi 2 Modelo B en febrero del 2016. Tiene un CPU ARMv8 de cuatro núcleos de 64-bits a 1.2 GHz, WIFI inalámbrica, Bluetooth 4.1, Bluetooth Low Energy (BLE), 4 puertos USB, 40 pines GPIO, puerto Full HD (1080p), puerto Ethernet, un Jack audio-vídeo de 3.5mm, interfase para cámara CSI (Camera Serial Interface) y para pantalla (DSI - Display Serial Interface), ranura micro SD, núcleo VideoCore IV 3D. Se basa en el uso de un sistema operativo llamado Raspbian derivado de Debian, o se pueden usar otros sistemas operativos. Los 40 pines GPIO tienen doble función:

- 24x - pines GPIO (General Purpose Input/Output)
- 1x - Serial UART
- 2x - SPI buses

- 1x - I2C bus
- 2x - 5V
- 2x - 3.3V
- 8x - Tierras

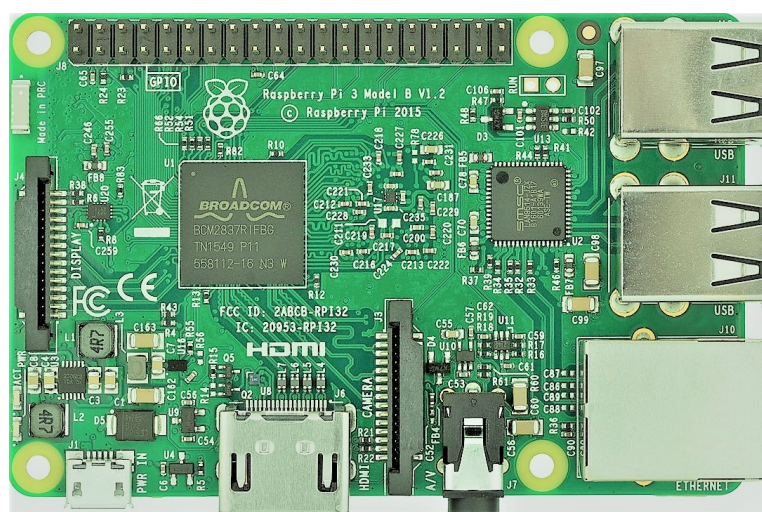


Figura 3.29: Raspberry Pi 3 ([Raspberry.org](https://www.raspberrypi.org/), 2018)

A pesar de las virtudes del Raspberry Pi 3, se eligió el Arduino Due, en base a los resultados de la Tabla comparativa 3.8

Tabla 3.8: Arduino Due vs. Raspberry Pi

Plataforma	UART	SPI	I2C	Salidas	Entrada	Lenguaje
Arduino Due	✓	✓	✓	✓	✓	✓
Raspberry Pi	✗	✓	✓	✓	✓	✗

Tabla Comparativa de Puertos

Arduino Due tiene todas las conexiones requeridas: número de UARTs, SPI, I2C, salidas y entradas. En cambio, Raspberry Pi solamente tiene un puerto UART. El uso de dos UARTs en el prototipo no se pudo reducir debido a que el

GPS y módulo Bluetooth solamente transmiten en serie asincrónica. Un factor que también tuvo peso es el lenguaje de programación, en el Raspberry Pi se utiliza *Python*, un lenguaje que se hubiera tenido que aprender desde las bases de la abstracción y los objetos. En cambio el lenguaje de programación para el Arduino Due es en esencia C++ un lenguaje que se conoce ampliamente sin necesidad de una curva lenta de aprendizaje.

Con todo lo descrito anteriormente se pudo realizar la integración de los componentes en el siguiente orden:

1. Armado de Chasis.
2. Armado e instalación de Motores y ESCs.
3. Conexiones e instalación de tarjeta de distribución de voltaje (PDB).
4. Instalación de Arduino Due.
5. Instalación de tarjeta prototipo y los dispositivos (breadboard).
6. Instalación de tarjeta Arduino Due.
7. Instalación de antena GPS.
8. Instalación módulo de control de energía del controlador Naza.
9. Instalación del sensor Ultrasónico.
10. Cableado.

A medida que se armaba el chasis se fue integrando la instalación de los motores, ESC y el PDB debido a que:

- las líneas de los motores corren por dentro de los brazos del dron.
- la PDB se instaló entre la placa inferior y superior del núcleo del chasis.
- los ESC van soldados al PDB así como el cable de alimentación de la batería.

La Figura 3.30 muestra el diagrama electrónico esquemático del dron y en la Figura 3.31 se muestra el diagrama simbólico para el fácil cableado entre los dispositivos.

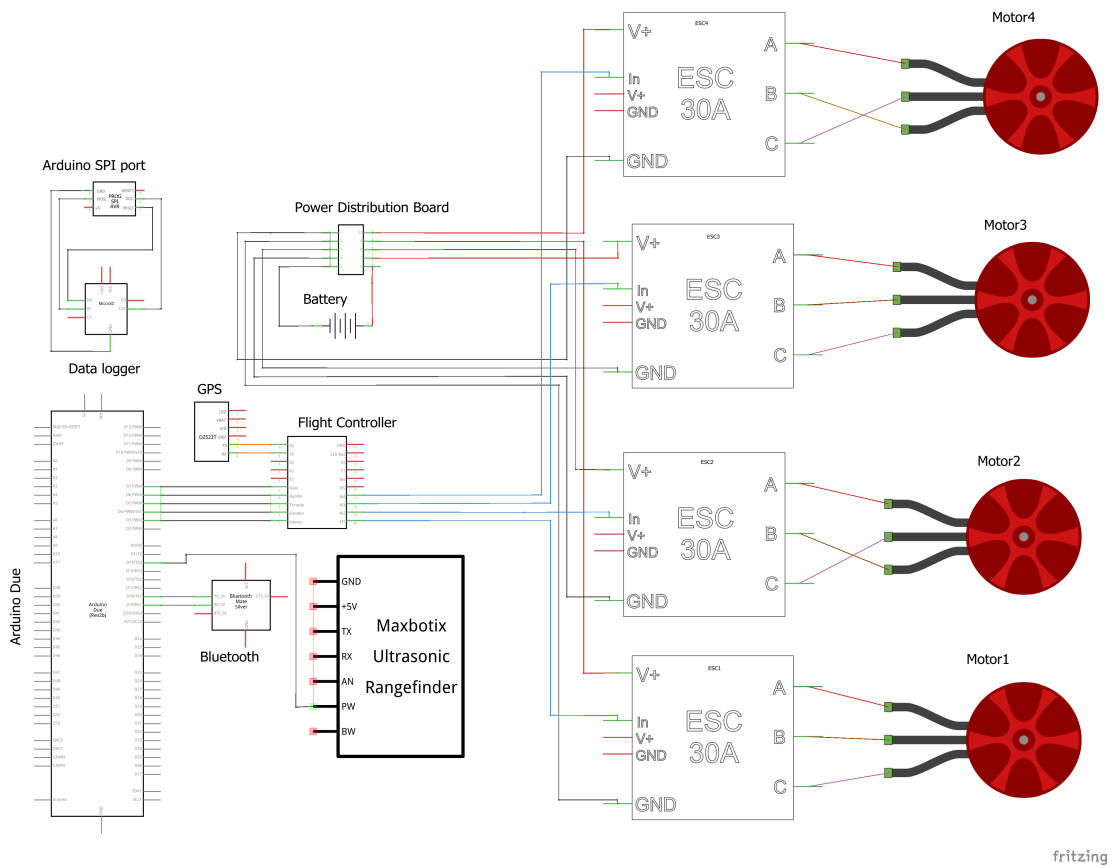


Figura 3.30: Diagrama esquemático del dron prototipo

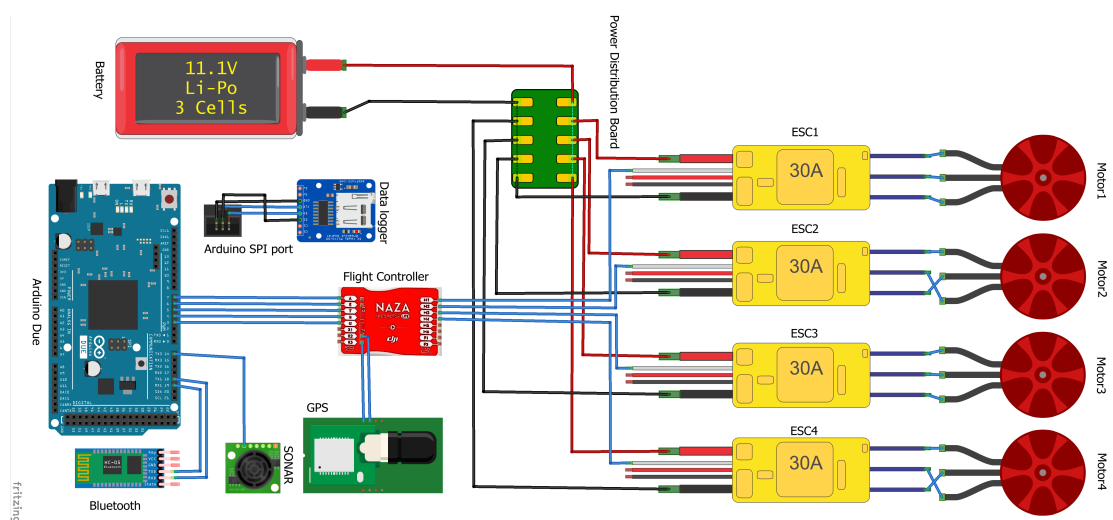


Figura 3.31: Diagrama simbólico del dron prototipo

3.4 Software

3.4.1 Introducción

A partir de este momento nos dirigiremos al Arduino Due como Arduino para ser prácticos. Al elegir Arduino como microcontrolador tenemos una ventaja más al codificar con Arduino Software IDE (Integrated Development Environment). Este software IDE es una aplicación que los desarrolladores usan para crear programas para Arduino. Incluye un editor de código fuente, un depurador y un compilador.

Arduino es una plataforma electrónica de Código-Abierto basado en un hardware y software de fácil uso. La tarjeta Arduino es capaz de leer señales de entrada, luz de un sensor, un botón presionado, o un mensaje en Twitter, activar un motor, encender un LED, publicar algo en-linea. Se le puede decir qué hacer a través de enviarle un conjunto de instrucciones al microcontrolador en la tarjeta. Para hacer esto, se usa el Lenguaje de Programación de Arduino (basado en programación *Wiring*) y el IDE de Arduino, basado en *Processing* (Arduino.cc, 2018). La GUI es similar a la de *Processing* e incorpora todas las librerías que tiene *Wiring*. El lenguaje de programación es C++.

Hay muchos otros microcontroladores y plataformas disponibles para la Informática física. Parallax Basic Stamp, Netmedia's BX-24, Phidgets, Handy-board y muchos otros que ofrecen una funcionalidad similar. Todas estas herramientas hacen la tarea sucia de la programación de microcontroladores y lo convierten en un paquete fácil de usar. Arduino también simplifica el proceso de trabajo con los microcontroladores, ofreciendo buena ventaja para los profesores, estudiantes y aficionados:

- **Bajo Costo** Las tarjetas Arduino son relativamente de bajo costo comparados a otras plataformas con microcontrolador. La versión más barata de un módulo Arduino puede ensamblarse a mano. Los módulos pre-ensamblados tienen un precio menor a los US\$50. Al momento de esta redacción la tarjeta Arduino-Due tenía un precio de US\$48.25 en (Arduino.cc, 2018).

- **Multi-plataforma** El Software de Arduino (IDE) corre en Windows, Macintosh OSX, los sistemas operativos Linux. La mayoría de los microcontroladores están limitados al uso en Windows.
- **Ambiente de programación sencillo y claro** El IDE de Arduino es de fácil uso para principiantes y también es competente para usuarios avanzados.
- **Software de Código-Abierto** El software de Arduino es publicado como herramienta de código-abierto, disponible para mejorarlo por programadores experimentados. El lenguaje puede extenderse por las librerías de C++. Las personas que quieren entender los detalles técnicos pueden hacer el salto de programación Arduino a programación AVR-C en el cual se basa el microcontrolador.
- **Hardware de Código-Abierto** Los planos de las tarjetas Arduino son publicados bajo una licencia *Creative Commons*, de tal forma que diseñadores de circuitos experimentados puedan crear su propia versión del módulo.

Arduino tiene una enorme variedad de librerías, unas de propósito general y otras para dispositivos específicos. Generalmente algún nuevo circuito impreso que quiera hacer presencia en el mundo del Arduino, llevará incluida librería y código ejemplo para su uso o comunicación. Una librería se utiliza para hacer la programación mucho más sencilla. Leer o escribir puertos o específicamente uno o algunos pines, ya sean analógicos o digitales, también librerías para los protocolos de comunicación SPI, I2C, UART, servicio de rutinas de interrupción, generación de ancho de pulsos modulados.

El Arduino IDE es muy sencillo y fácil de usar, haciendo un recorrido por el menú principal encontramos las opciones **File y Edit** que tienen las funciones estándares de Windows. El menú Programa contiene las opciones para compilar, subir y ejecutar un programa en el microcontrolador. Incluir los encabezados de librerías al código. En el menú **Herramientas** se encuentran

las configuraciones de uso del modelo específico de Arduino, bajar actualizaciones, configuración de puertos en la PC hacia el Arduino, etc. La ventana del Arduino IDE se muestra en la Figura 3.32.

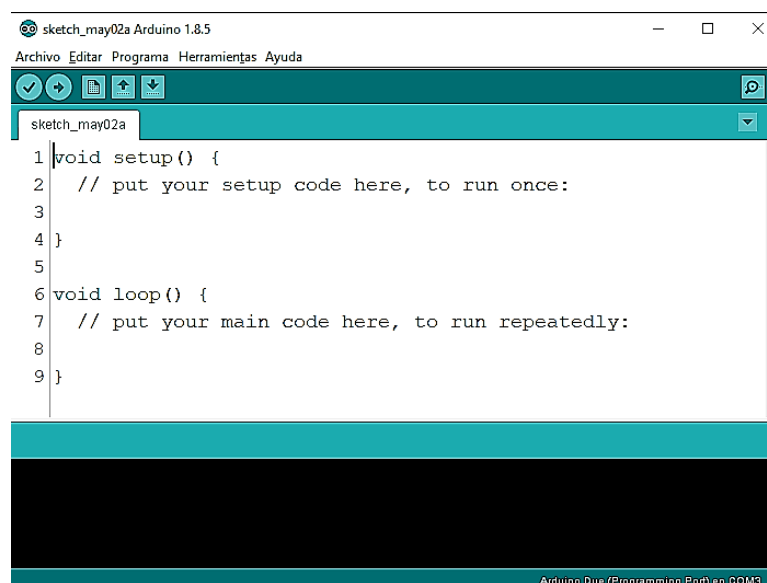


Figura 3.32: Arduino - Entorno Integrado de Programación ??

En el IDE se pueden encontrar ejemplos de programas donde se entiende fácilmente el uso de las librerías.

3.5 Software de Arduino

El término en inglés para el código en IDE de Arduino se conoce como *sketch*, se utiliza este término para diferenciar al entorno de programación (Software de Arduino) del código. En un sketch de Arduino IDE, el uso de la función *void setup()* y *void loop()* es de carácter obligatorio. Estas funciones son una conveniencia en la programación estructurada.

El espacio de texto anterior a la función *setup*, está asignado a las librerías que se incluyen, directivas del pre-procesador, declarar y asignar valores a variables globales, constantes globales, declaración de los encabezados de todas las funciones. Es el preámbulo del sketch.

La función *void setup()* se incluye el código para inicializar el sistema diseñado: variables, ejecuciones de funciones de una sola vez, inicialización de los puertos y protocolos de comunicación. Puede tener también sus propias variables locales.

La función *void loop()* esta función es un bucle que contiene el código que se ejecutará de manera indefinida. Dentro de este *ciclo* se invocarán aquellas subrutinas o funciones que sean necesarias.

3.6 Software en el Celular

En la actualidad existen casi 7.5 mil millones de habitantes en el mundo y en 2016 se contabilizaron 2.6 mil millones de personas con un **teléfono celular o Inteligente** y las cifras siguen creciendo (Pi, Charly, 2017). El sistema operativo con mayor presencia en los *smartphones* y en general, es sin duda alguna *Android* (Pi, Charly, 2018). Ver Figura 3.33.

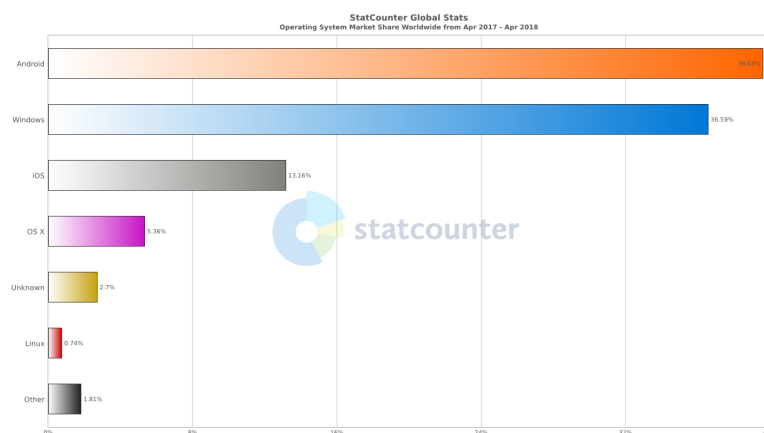


Figura 3.33: Participación en el Mercado de Sistemas Operativos a nivel mundial de Abril 2017 - Abril 2018 (Pi, Charly, 2018)

La decisión de utilizar un *smartphone-Android* se basó en la siguiente lista de puntos

- la portabilidad.
- la disponibilidad del recurso.

- cuenta con receptor GPS.
- cuenta con Bluetooth.
- se pueden desarrollar/instalar programas.

Todos los puntos son ventajas para integrarlo al proyecto como elemento de transmisión de datos vía inalámbrica. Sin embargo, programar un Android requeriría invertir un tiempo considerable por los medios tradicionales. Hoy en día existen aplicaciones para casi todo. También existe una amplia gama de aplicaciones o lenguajes de programación que son intuitivamente amigables, uno de estos es el **App Inventor** de MIT (Massachusetts Institute of Technology).

3.6.1 ¿Qué es MIT App Inventor?

El **MIT App Inventor** es un entorno de programación visual intuitivo que permite a cualquiera crear Apps completamente funcionales para el *smartphone* y *tablet*. Los principiantes pueden tener su primera App sencilla en menos de 30 minutos. También se puede crear Apps complejas con herramientas basadas en bloques (Massachusetts Institute of Technology, 2018). App Inventor provee de bloques que pueden acomodarse de tal forma que ejecutan funciones para cualquier propósito u objetivo. La herramienta en sí, es un punto de acceso a la mayoría de las características del dispositivo móvil.

MIT App Inventor es usado por estudiantes, maestros, desarrolladores, aficionados y emprendedores para desarrollar Apps para la colaboración, productividad, uso personal, recreación, aprendizaje, bienestar social y comunitaria. Este software es solo para dispositivos Android y es de Código-Abierto. Cualquiera puede descargar el código fuente para ejecutar en sus propios servidores, modificar, personalizar y mejorar sus propias versiones (Pokress & Veiga, 2013).

Con esto en mente, se desarrolló una “App” que pudiera enviar datos al dron, vía Bluetooth. El entorno de programación consiste de dos vistas: **Diseñador** y **Bloques**, ver las Figuras 3.34 y 3.35 respectivamente.

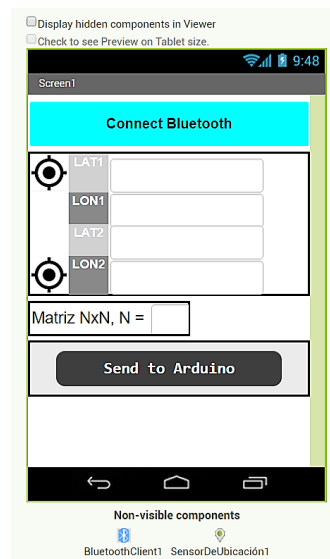


Figura 3.34: Vista de Diseño (Interfaz con el usuario)

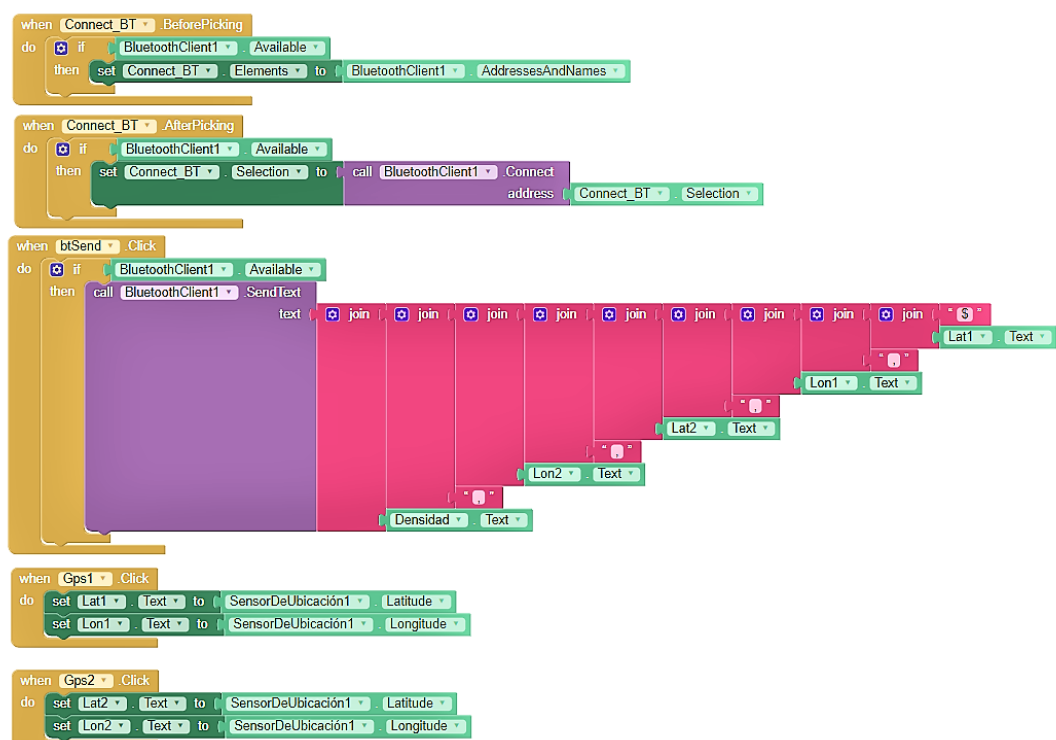


Figura 3.35: Vista de Bloques (código)

3.7 La App

Antes de ejecutar la App (Application for smartphone), se debe asegurar que Bluetooth del smartphone esté emparejado con el módulo Bluetooth del dron y que el GPS del smartphone esté encendido. En pantalla, se pulsa el Botón “Connect Bluetooth” para que la conexión se active. Después, establecer las coordenadas GPS inicial y final. Para la coordenada inicial es **Lat1** y **Lon1**. Para la coordenada final es **Lat2** y **Lon2**. Se define el valor de N , que será el tamaño de una matriz cuadrada $N \times N$. El recorrido del dron es en forma de espiral cuadrada. Cada elemento de la matriz es un punto de navegación. Esto se detallará más adelante. Al presionar el botón “Send to Arduino” el smartphone enviará por medio de Bluetooth, una cadena de caracteres con el siguiente formato:

$$\text{\$Lat1,Lon1,Lat2,Lon2,N}$$

Ejemplo:

$$\text{\$32.631571,-115.441325,32.634125,-115.437403,4}$$

Esta cadena de caracteres ASCII es enviada empezando por el signo “\$” y se genera mediante los bloques de la Figura 3.36, mientras que el dron se encuentra en un ciclo de espera para recibir la cadena completa y extraer sus elementos.

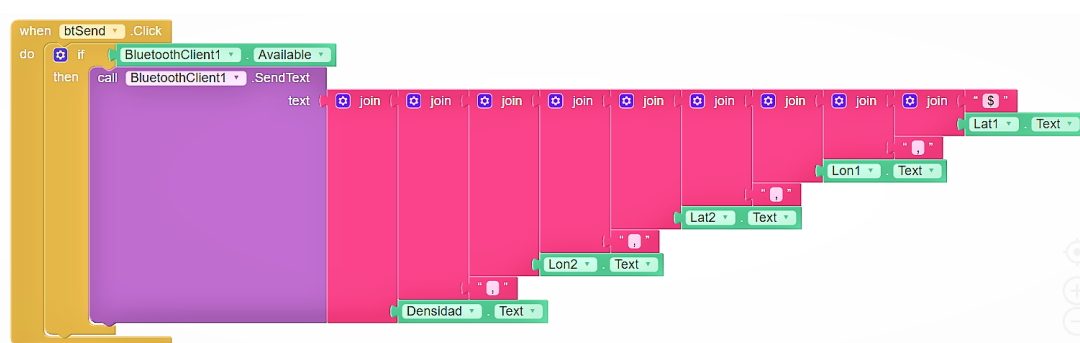


Figura 3.36: Bloques para Enviar Cadena

La interpretación de este código de bloques es: cuando haga clic al botón de enviar, y si hay conexión Bluetooth, entonces envía el texto concatenado.

3.8 Módulo de Motores

Este módulo se encarga de ejecutar las señales PWM que recibe del módulo *Control de Vuelo*. Los ESCs son responsables de hacer girar los motores a una velocidad requerida por el controlador de vuelo. La mayoría de los ESC necesitan calibrarse y esta no es la excepción. Los ESC necesitan saber el rango de operación de las señales PWM que el controlador de vuelo estará enviando (milehighrc.com, 2018).

3.8.1 Calibración de ESCs

Se calibra un ESC a la vez. Se utilizó un sketch escrito especialmente para este propósito.

El algoritmo es como sigue:

1. ESC y Arduino apagados.
2. Se energiza Arduino para ejecutar el sketch.
3. Arduino envía una señal PWM de 2000 μs al ESC (apagado todavía).
4. Se energiza ESC e inmediatamente detecta la frecuencia del PWM.
5. Después de 2 segundos de encendido el ESC, Arduino envía otra señal PWM de 1000 μs .

De esta manera el ESC reconoce un valor mínimo de 1000 μs y uno máximo de 2000 μs . Esto es la calibración de ESCs. En la web hay quienes recomiendan utilizar rangos de 0 a 180 μs , dado que la utilería <Servo.h> está diseñada para los servo-motores. Al principio se intentó así, teniendo fallas de calibración. Hasta que se encontró la recomendación de 1000 a 2000. Este método se tiene que ejecutar para cada uno de los ESCs.

El sketch se encuentra en la Sección ([Calibración del ESC](#)) del [Anexo B](#).

3.8.2 Armar los motores

“Armar los motores” significa que el Controlador de Vuelo aplicará energía a los motores, y empezarán a girar. El sketch de prueba para Armar los motores realiza los siguientes pasos, en forma abreviada:

1. se incluyen las librerías `<Servo.h>`, `<SPI.h>`, `<NazaDecoderLib.h>`
2. Arduino inicializa las estructuras de datos para las maniobras *roll*, *pitch*, *throttle*, *rudder* y *modo de vuelo*. Cada estructura tiene valores mínimos, máximos, medios y bajos en μs .
3. crea las clases para cada estructura y asigna pines para envío de la señal PWM al Controlador de Vuelo.
4. Arduino envía señales al controlador de vuelo para estar en modo neutral y cero impulso.
5. Arduino envía una ráfaga de señales PWM con todos los controles neutros excepto el control de *roll*.
6. empiezan a girar los motores a velocidad de *ralentí*.
7. después de un tiempo determinado, se desarman los motores.

El sketch se encuentra en la Sección ([Armar los Motores](#)) del [Anexo B](#).

3.8.3 Despegue Vertical

La forma tradicional para determinar la eficiencia del vuelo vertical y sostenido, es considerando la fuerza de carga del vehículo. Esto es una relación entre el peso del vehículo y la fuerza de los motores. Un vehículo más eficiente requiere motores menos potentes para vuelo sostenido con cierto peso ([Blackwood & Kingb, 2018](#)). Para que un dron logre un vuelo estacionario, la fuerza de impulso de las hélices debe ser igual a la resistencia y a su vez, la fuerza de sustentación debe ser igual a la del peso del dron. Si se incrementara la fuerza de impulso, seria menor a la resistencia y el dron, se elevaría. Se debe encontrar la fuerza necesaria en los motores (impulso) que logre superar la resistencia del peso y lograr la sustentación.

3.9 Módulo Bluetooth

Este módulo es la interfase entre el smartphone y la tarjeta Arduino montada en el dron. Sus funciones son:

- Recibir inalámbricamente los datos.
- Transmitir los datos al Arduino por el bus **UART1** configurado a 9600 bps.
- En este módulo, se considera el *parser* parte del sketch en Arduino, que separa datos y asigna variables.

Para empezar la lectura de la cadena de caracteres, Arduino se encuentra en ciclo de espera mientras llega el caracter “\$”.

Código en Arduino:

```
// ciclo de espera
Serial.print( "Waiting for Android input..\n" );
while ( Serial1.read() != '$' ) {}

// Código del parser:
cad = Serial1.readStringUntil( '%' ); // parse lat1
Lat1 = ( double ) cad.toFloat();

int idx = cad.indexOf( ',' ); // parse lon1
aux = cad.substring( idx+1 );
Lon1 = ( double ) aux.toFloat();

idx = cad.indexOf( ',', idx+1 ); // parse lat2
aux = cad.substring( idx+1 );
Lat2 = ( double ) aux.toFloat();

idx = cad.indexOf( ',', idx+1 ); // parse lon2
aux = cad.substring( idx+1 );
Lon2 = ( double ) aux.toFloat();

idx = cad.indexOf( ',', idx+1 ); // parse N
aux = cad.substring( idx+1 );
N = aux.toInt();
```


3.10 Módulo micro SD

Este módulo utiliza la librería *<SPI.h>* que ahorra el trabajo de interpretar los bits, transiciones de bajada y de subida de señales, así como la configuración del protocolo. Y la librería *<SD.h>* sirve para leer y escribir datos en las tarjetas de memoria SD o microSD.

Este Módulo se encarga de:

- crear los archivos “navigate.txt” y “datalog.txt” en el micro SD.
- calcular los puntos de navegación de los datos adquiridos por el módulo Bluetooth.
- escribir los puntos de navegación en forma de registro en el archivo “navigate.txt”.
- leer cada registro y entregarlo al módulo Control de Navegación.
- escribir registros conteniendo: fecha, hora, datos de monitoreo y de navegación en el archivo “datalog.txt”

En el archivo “navigate.txt” se genera una cadena de caracteres por cada posición en una matriz simulada de $N \times N$, ver Figura 3.42. El formato es el siguiente:

$\$K,R,C,Lat, Lon$

donde

- **K** es el número del punto de navegación actual, $K \in \{1..N^2\}$
- **R** es el número de renglón de la matriz $N \times N$. $R \in \{1..N\}$
- **C** es el número de columna de la matriz $N \times N$. $C \in \{1..N\}$
- **Lat** es la latitud para la posición R,C en la matriz.
- **Lon** es la longitud para la posición R,C en la matriz.

3.11 Módulo Ultrasónico

Como se explicó en 3.3.3, el sensor entrega un ancho de pulso a la entrada D20 del Arduino. Es sencillo el acoplamiento Sensor-Arduino, debido a que el microcontrolador cuenta con la función *pulseIn(pin, value)* que lee un pulso en un pin. Si *value* = HIGH, el pulso que espera es de subida. Si es LOW, el pulso que espera es de bajada. Cuando la función detecta el pulso arriba en el pin D20, empieza a contar el tiempo hasta que el pulso baja. El valor que entrega la función *pulseIn* es en micro-segundos (μs) con un tipo de dato *unsigned long* y es proporcional a la distancia del objeto detectado. $1\text{cm} = 58\mu s$. El rango confiable de operación está entre los $10\mu s$ hasta 3 minutos de largo.

```
duration = pulseIn(sonarpin, HIGH);  
range= (unsigned long) (duration/58);
```

donde

- *duration* es el tiempo que tarda en viajar la onda ultrasónica del sensor hasta el objeto, en μs .
- *range* es la distancia en centímetros calculada de dividir los μs entre 58, $1\text{cm} = 58\mu s$.
- *sonarpin* es el pin digital D20 del Arduino.
- *HIGH* es la constante definida en el pre-procesado, con valor 1 (transición de subida).

Este sensor está instalado en la parte inferior del centro del dron, para medir la distancia al suelo. Con este dato se pretende limitar la altura de vuelo del dron (escenario controlado) y evitar que vuele sin regreso.

3.12 GPS + Brújula

“GPS+Brújula” es un módulo que integra un receptor GPS de ultima generación, el **Ublox NEO-M8N**, viene acompañado del magnetómetro digital **HMC5883L**. Un módulo instalado “lejos” de los otros elementos electrónicos

y eléctricos en el dron hace que estos sensores tengan menos interferencias. El receptor GPS es de alta sensibilidad y tiempos de adquisición reducidos mientras mantiene una baja potencia en el sistema. Acepta todos los sistemas de Aumentación Satelital (Satellite Augmentation Systems), Es de la categoría de precisión estándar GNSS y acepta, GPS / QZSS, GLONASS, Galileo y BeiDou (son sistemas de posicionamiento satelitales) puede trabajar con 3 sistemas a la vez. Maneja lógica TTL de 3.3V, tiene puertos UART, USB, SPI y DDC (compatible con I2C), tiene un flash programable y registro de datos y es de grado profesional (Ublox, 2017). Los límites de operación del dispositivo son: Altitud: 50,000 m, Velocidad: 500 m/s, precisión de velocidad: 0.05 m/s, precisión de dirección: 0.3° y precisión horizontal: 2.5 m.

Este módulo se comunica con Arduino vía UART a 115200 bps, le envía mensajes binarios que son decodificados por la librería *<NazaDecoderLib.h>*. También contiene ejemplos para Arduino utilizando las funciones de la librería desarrollados por (pawelsky, 2013). Ejemplo de sketch usando esta librería y sus funciones:

```
#include "NazaDecoderLib.h"

void setup() {
  Serial2.begin(115200);
}

void loop() {
  decodedMessage = NazaDecoder.decode(Serial2.read());
  switch (decodedMessage) {
    case NAZA_MESSAGE_GPS:
      Serial.print("Lat:");
      Serial.print(NazaDecoder.getLat(), 7);
      Serial.print(",");
      Serial.print(NazaDecoder.getLon(), 7);
      Serial.print(",");
      Serial.print(NazaDecoder.getGpsAlt(), 7);
      Serial.print(",");
      Serial.print(NazaDecoder.getFixType());
      Serial.print(",");
      Serial.print(NazaDecoder.getNumSat());
      break;
    case NAZA_MESSAGE_COMPASS:
      Serial.print("Heading:");
      Serial.print(NazaDecoder.getHeadingNc(), 2);
  }
}
```

```
        break;  
    }  
}
```

donde:

- **NazaDecoder.decode(`Serial2.read()`)** entrega un valor que indica si el mensaje proviene del receptor GPS o de la brújula,
- **`Serial2.read()`** lee del puerto UART2, pin RX2, la trama de datos binarios. Nota: UART2 en Arduino es para GPS.
- **`NazaDecoder.getLat()`** entrega la Latitud de la posición del módulo.
- **`NazaDecoder.getLon()`** entrega la Longitud de la posición del módulo.
- **`NazaDecoder.getAlt()`** entrega la Altitud de la posición del módulo.
- **`NazaDecoder.getFixType()`** entrega el tipo de fijación que tiene el módulo con los satélites.
- **`NazaDecoder.getNumSat()`** entrega el número de satélites a los que tiene conexión.

Al ejecutar este código muestra datos como en la Figura 3.37.

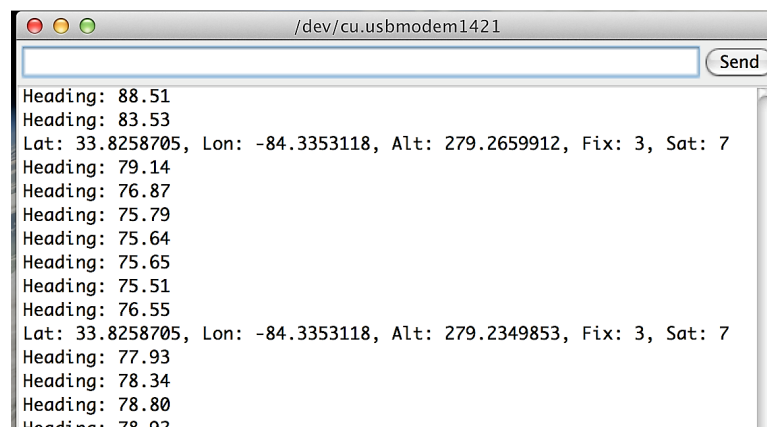


Figura 3.37: Trama de datos de GPS (Hagiopol, 2016)

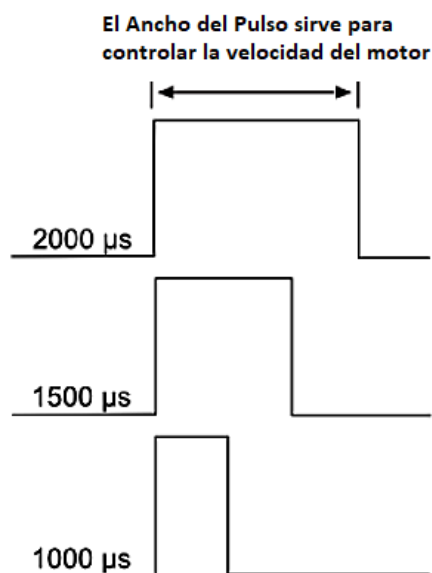


Figura 3.38: Ancho de Pulsos del receptor de RC (Parallax Inc., 2018)

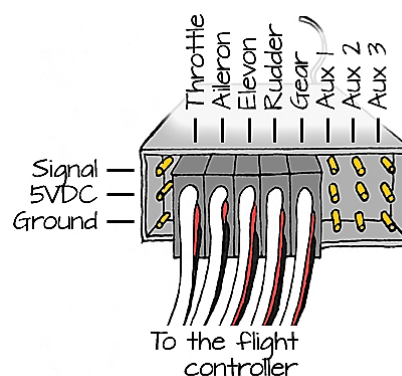


Figura 3.39: Receptor RC - Salidas al controlador de vuelo (Parallax Inc., 2018)

3.13 Control de Vuelo

En la Tabla 3.6 podemos ver las entradas y salidas del controlador de vuelo Naza, pero ¿Cómo es la señal que reciben estos puertos? La respuesta está en un receptor de Radio Control (RC) donde sus puertos de salida van hacia el controlador de vuelo en una configuración *tradicional*. El receptor RC envía señales PWM con rango en la modulación de ancho de pulso entre 1000-2000 μ s Ver Figuras 3.38 y 3.39.

Con la identificación y caracterización de las señales del controlador de vuelo, se programó el Arduino con 5 salidas PWM en este rango de ancho de pulsos. Donde un valor de 1000 μ s (límite inferior) en el puerto T (Throttle) significa que el controlador no acelerará los motores. Un valor de 1500 μ s (nivel medio) en el puerto T tendrá una aceleración al 50%. Los puertos del controlador, sus valores y el efecto que tienen en el dron, se pueden resumir en la Tabla 3.9.

Tabla 3.9: Tabla de Puertos del Controlador de Vuelo

Puerto	Ancho de Pulso	Movimiento en el Dron
Aileron	1000	Lateral hacia la izquierda.
	1500	neutral.
	2000	lateral hacia la derecha.
Elevator	1000	hacia atrás.
	1500	neutral.
	2000	hacia adelante.
Throttle	1000	hacia abajo.
	2000	hacia arriba, se elevará.
Rudder	1000	giro hacia la izquierda.
	2000	giro hacia la derecha.

El controlador de vuelo Naza M Lite tiene un software asociado a este, que es necesario para la configuración de los parámetros del Naza. El programa se llama “Naza-M Lite Assistant Software v1.00” y se descargó en (DJI, 2014) donde se encuentra también el manual de configuración.

En DJI (2014) se encuentra un procedimiento que se siguió:

- descarga e instalación del programa de asistencia del Naza.
- conexiones de controlador de vuelo a los ESCs y al receptor RC, que en nuestro caso es el Arduino.
- configuración de tipo de multi-rotor.
- configuración de montaje de la unidad GPS.
- procedimiento de calibración de los puertos mencionados en la Tabla 3.9.
- ajuste de parámetros de los motores.
- procedimiento para Armar Motores.

- configuración de la modalidad en caso de falla de comunicaciones.
- configuración del tipo de control para la orientación.
- configuración de conexión del gimbal de cámara.
- configuración de protección contra voltaje bajo.
- procedimiento de calibración de la brújula digital (ubicada en la unidad GPS).

Los puntos mencionados en la lista son los que se aplicaron de acuerdo a la configuración Arduino-Naza. Otros puntos fueron omitidos. Para más detalles sobre el uso del programa se pueden encontrar en (DJI, 2014).

Para realizar la calibración de los puertos, se codificó un sketch en Arduino para transmitir las señales PWM a cada puerto, todos a la vez. Con este sketch, los puertos reciben inicialmente la señal PWM de $1000\mu s$ y se incrementa hasta llegar a los $2000\mu s$. De esta forma, el controlador de vuelo recibe el rango de trabajo para cada puerto, similar a la calibración de los ESC. El código de este sketch se incluyó en el [Anexo B, Calibración de Puertos Naza](#).

El llegar a este punto, la programación es más sencilla. La estabilidad de vuelo no depende de Arduino sino que fue delegada al módulo Naza. Con solo el siguiente comando en Arduino, hace que el dron se eleve:

```
throttle.writeMicroseconds ( 1800 ) ;
```

3.14 Control de Navegación

En la Figura 3.40 tenemos el diagrama bloques que representa la modularidad del sketch. Este bucle está representado por el bloque de control de navegación, que revisa de manera cíclica los bloques micro SD, Ultrasónico, Control de Vuelo y la señal del GPS. El módulo Bluetooth se ejecuta en la función *void setup()* una sola vez. En el diagrama se aprecia con una línea punteada. Podemos ver la relación que hay entre los módulos y la dirección de las señales que circulan en la plataforma.

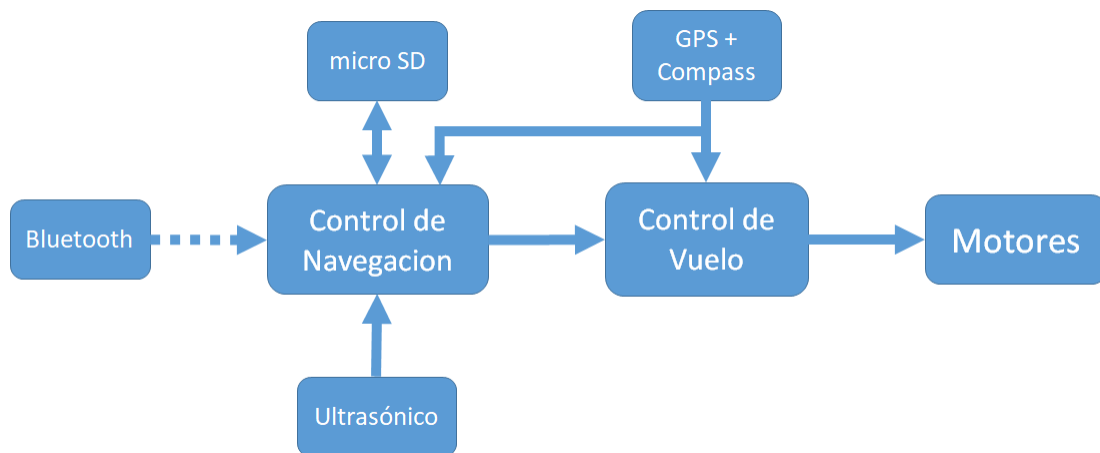


Figura 3.40: Diagrama de Bloques del Sistema

Para terminar de mencionar todos los componentes del sistema, solo falta describir el **Control de Navegación**. En la Figura 3.41 el diagrama de flujo, donde sus primeros tres bloques después del **inicio**, corresponden a la función *setup()*, son los bloques no incluidos en el *bucle*: Inicializar puertos de comunicación del Arduino, recibir datos vía Bluetooth y generar puntos de navegación. Por otro lado, el resto de los bloques corresponden a la función *loop()* del sketch: leer el GPS+Brújula y registrar datos recolectados durante el vuelo en el archivo "datalog.txt", calcular la orientación relativa y distancia al punto de navegación, enviar señales al controlador de vuelo, llegar al punto de destino y repetir el ciclo hasta encontrar el último punto de navegación almacenado en el archivo "navigate.txt".

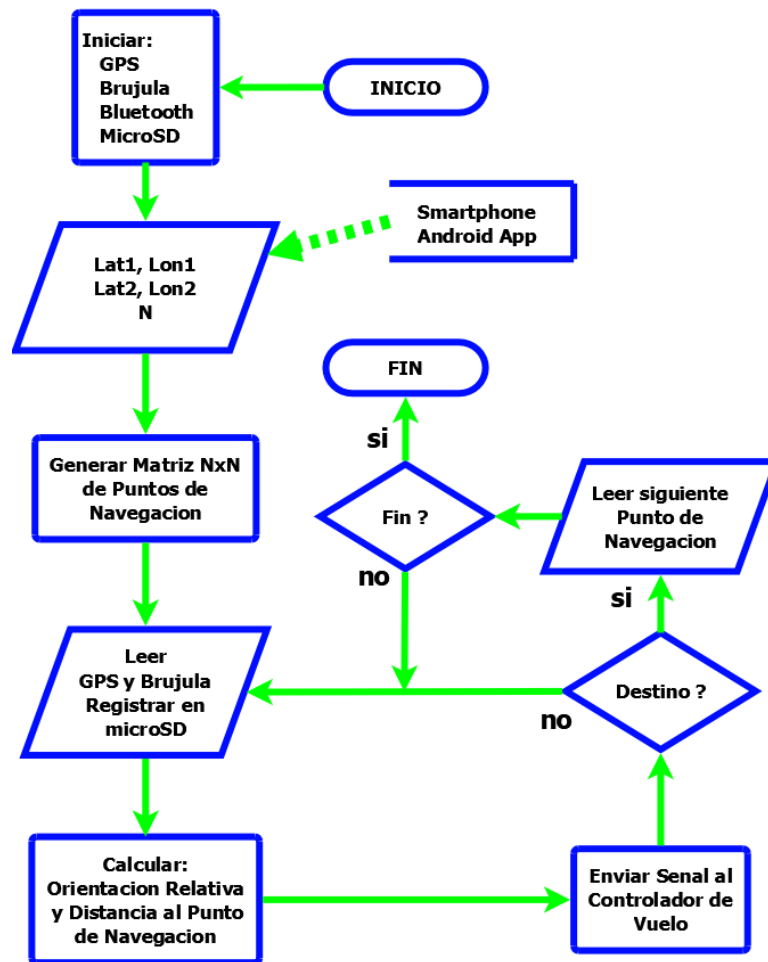


Figura 3.41: Diagrama de flujo de control de Navegación

A continuación revisaremos los siguientes puntos: Generación de puntos de navegación, recolección de datos, registro de estos datos, cálculo de orientación relativa y distancia al punto de navegación.

3.14.1 Generación de puntos de navegación

La generación de los puntos de navegación consiste en definir una matriz de $N \times N$ y generar datos de ubicación (latitud y longitud) en base a los datos enviados por Bluetooth. La primera posición (Lat1, Lon1) está situada en el renglón 1, columna 1 (1,1) y el segundo par de datos en (10,10). Los demás elementos de la matriz se calculan dividiendo la diferencia de sus coordenadas

entre N, considerando que la curvatura de la tierra no sería significativa en un área de puntos relativamente pequeña. Estos datos calculados son almacenados en un archivo de registros secuenciales siguiendo el orden de la trayectoria de la Figura 3.42 donde el primer registro del archivo contiene los datos de la celda (1,1) y el ultimo registro contiene los datos de la celda en el centro de la matriz, en la figura es la celda (5,6).

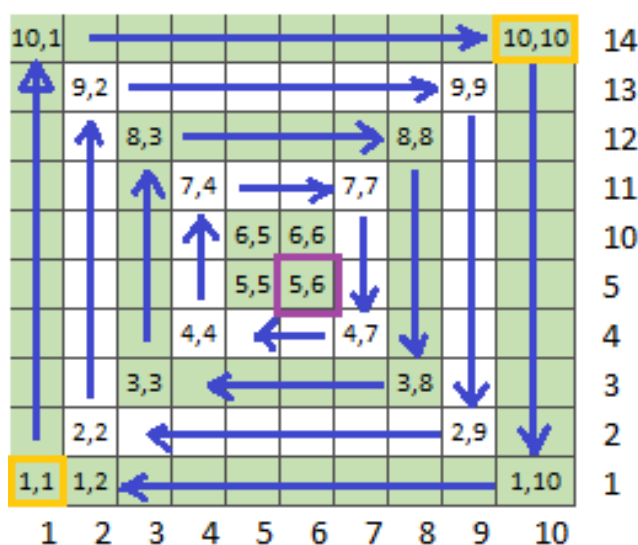


Figura 3.42: Matriz de puntos de navegación (N=10)

Durante el vuelo, se registra la información de navegación y valores de sensores en el archivo “datalog.txt”, conteniendo los siguientes datos:

\$Lat,Lon,fecha,hora,dato1,dato2

donde

- **Lat** latitud actual de vuelo.
- **Lon** longitud actual de vuelo.
- **fecha** fecha actual.
- **hora** hora actual.

- **dato1** dato del sensor1.
- **dato2** dato del sensor2.

3.14.2 Dirección-relativa y Distancia

Para que el dron vaya hacia un punto dado, necesita saber:

- su ubicación actual.
- la ubicación del punto de destino.
- su orientación (rumbo) respecto al norte magnético.
- la dirección-relativa, es el ángulo que tiene que cerrar para que el rumbo sea en dirección al destino ¿tiene sentido? ver Figura 3.43.
- la distancia entre su posición actual y el destino.

la ubicación actual y su orientación respecto al norte magnético, se obtiene de la unidad GPS. El punto de destino se obtiene del archivo “navigate.txt”. Para calcular la dirección-relativa y la distancia se utilizan estas funciones:

```
dir-relativa = courseTo( Lat-dron, Lon-dron,
                        Lat-destino, Lon-destino );
distancia = cosDistance( Lat-dron, Lon-dron,
                        Lat-destino, Lon-destino );
```

donde *dir-relativa* es la diferencia del angulo del punto destino respecto al norte magnético y el angulo de orientación del dron , *Lat-dron*, *Lon-dron* la posición actual del dron y *Lat-destino*, *Lon-destino* la posición del destino. El código de estas funciones están en [Direccion-relativa y Distancia](#) del [Anexo B](#), las fórmulas fueron consultadas en ([Veness, 2017](#)).

Con el dron en movimiento, en cada ciclo del bucle del microcontrolador se re-calcula la posiciona actual, la distancia actual, la nueva dirección-relativa y verificando si ha llegado al punto destino ($distancia \leq 2.5m$). Donde el valor 2.5 es el margen de precisión que especifica el fabricante del receptor GPS.

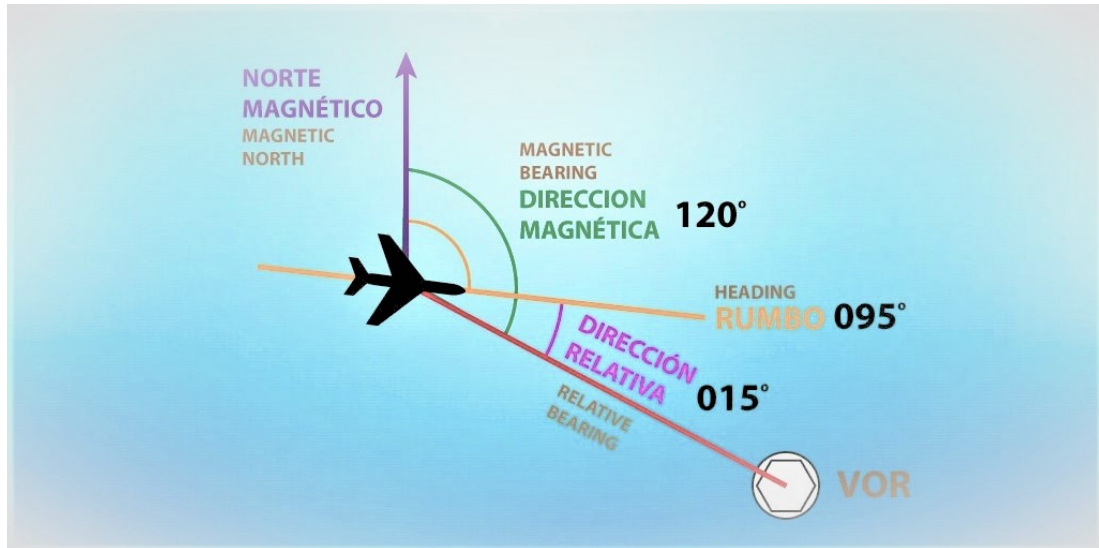


Figura 3.43: Rumbo, Dirección-relativa, Dirección Magnética y Norte Magnético (Administrador, 2017)

Capítulo 4

Resultados y Discusión

En este capítulo se presentarán los escenarios de prueba que fueron logrados mediante la aplicación de los métodos y algoritmos descritos en el capítulo anterior.

4.1 Prueba de Vuelo



Figura 4.1: Ensamble final del dron prototipo

En general, motores de bajo KV y grandes hélices tienen mayor eficiencia y pueden portar una batería más grande y por lo tanto tienen mayor tiempo de vuelo. La eficiencia en los motores sin escobillas empeora a medida que se

incrementa la velocidad. Por lo tanto, un vuelo rápido y acrobático acortará el tiempo de vuelo dado que requiere de una velocidad y potencia mayores.

La eficiencia del prototipo la podemos ver en la Figura 3.16 donde el parámetro Impulso-Peso en la página 105 nos muestra que tiene un buen nivel de maniobrabilidad. Este es un indicador del desempeño de vuelo (entre más alto el valor, más ágil es el dron) Un valor de 1.8 o más, resulta en menos del ~60% de su capacidad máxima para lograr el vuelo estacionario. De 1.2 a 1.8 resulta en una velocidad de vuelo estacionario entre 60 y 80% de su máxima capacidad (maniobrabilidad limitada). Abajo de 1.2 no puede elevarse. Entre menor sea el porcentaje, mayor margen de velocidad tendrá el motor para lograr impulsarse y controlar la estabilidad.

Esto quiere decir que es un dron grande, ligero en peso y motores potentes. Hay que observar en la Figura 3.18 que la Eficiencia en estacionario de 82.9% y la Eficiencia al máximo de 78.3% se refieren únicamente a los motores, aún así es una buena referencia.

El prototipo tiene un margen de carga adicional de 2.89 kilogramos según los resultados obtenidos con *eCalc* en la Figura 3.18.

Las pruebas se realizaron en el Instituto de Ingeniería, en un salón de clase. Se realizaron varias pruebas de despegue, alcanzando una condición aceptable de vuelo en la última prueba realizada.

El dron se sujetó con un cordón de 1.5 metros de largo pegado con “duct-tape” al piso. En la primer prueba ($Test_0$) de la tabla 4.1, la falta de referencia o práctica con el parámetro de velocidad en el controlador de vuelo y junto con un dron potente, dio como resultado una aceleración rápida haciendo que el dron rompiera su atadura.

Se logró contener el potencial riesgo, evitando que llegara al falso plafón. Sin embargo, una de las hélices se quebró y la batería se dañó por impacto. Posteriormente, se realizaron pruebas sujetándolo desde cada uno de sus lados. Cada anclaje se hizo con un aro de acero de 2 pulgadas de diámetro sujetado al piso con “duct-tape” en forma triangular para tener mayor adherencia, ver Figura 4.2.

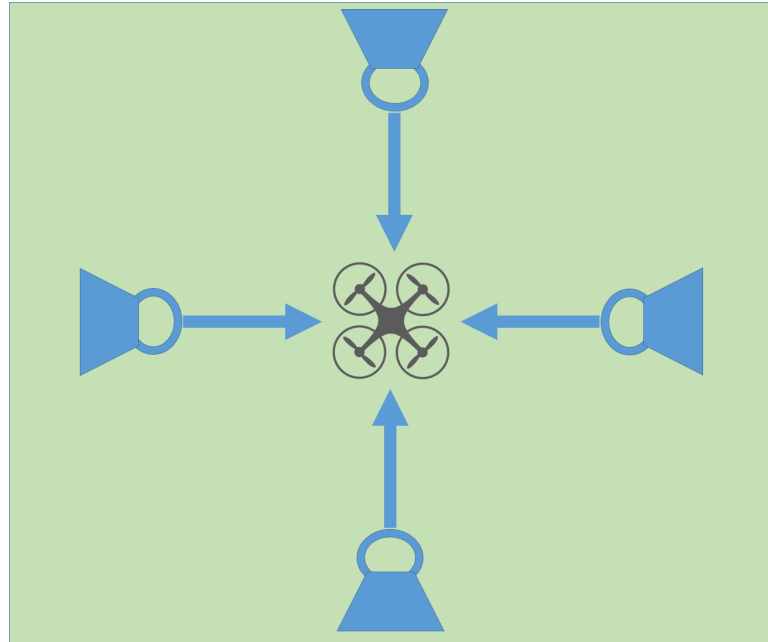


Figura 4.2: 4 sujetadores por lado

4.2 Ajuste de Parámetros

Las pruebas posteriores se realizaron de manera sistemática. Para lograr que los rotores generaran una fuerza de impulso mayor a la gravedad de la tierra, se diseñó una función que tuviera las siguientes características (ver 4.1):

- de reacción rápida de velocidad a baja altura (h) y disminución al alcanzar la altura deseada.
- los límites de amplitud configurables (T_{min} y T_{max}) ver Ecuación 4.1.

El ajuste de los parámetros se realizó a través de modificar el código y descargarlo vía USB al Arduino, ver 4.3.

Con incrementos controlados en la velocidad, cambios en el límite superior de la función, se realizaron pruebas con los datos de la Tabla 4.1.

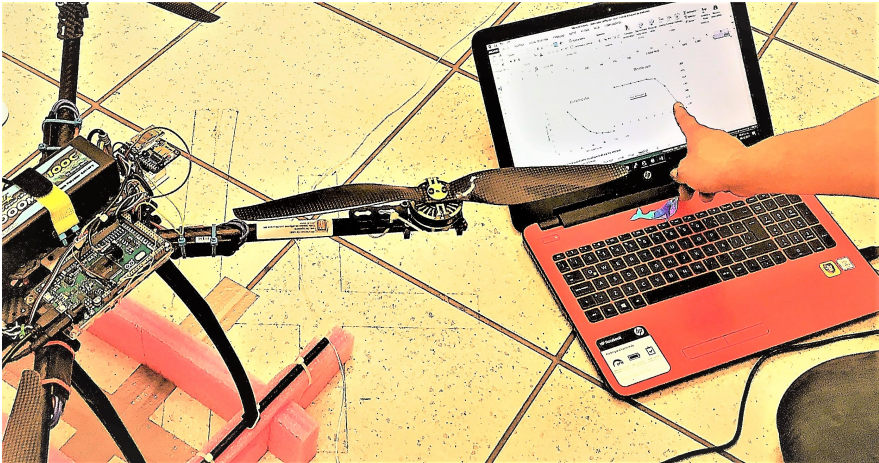


Figura 4.3: Ajustando parámetros de Despegue

Tabla 4.1: Pruebas de Despegue

	$Test_0$	$Test_1$	$Test_2$	$Test_3$
T_{max}	170	50	60	80
T_{min}	0	0	5	5

Tabla de límites de amplitud utilizados en las pruebas de despegue ver Figura ??

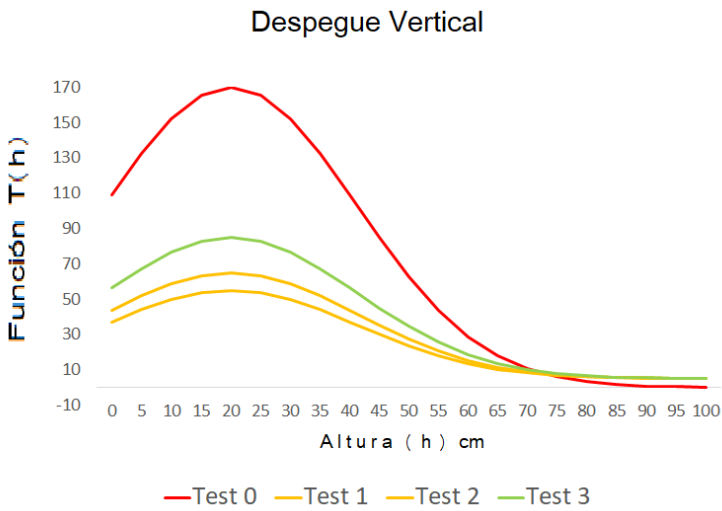


Figura 4.4: Pruebas de Velocidad utilizando Ecuación 4.1

$$T(h) = T_{min} + T_{max} e^{\left(\frac{-(\frac{h}{10}-2)^2}{9} \right)} \quad (4.1)$$

donde T_{min} y T_{max} son los valores mínimo y máximo de la función $T(h)$, y h es la altura de vuelo del dron. T influye en el impulso vertical del dron con la instrucción C++:

```
throttle.writeMicroseconds( impulso.mid + T );
```

El valor de **impulso.mid** es igual a $1500 \mu s$, así que la señal de la función $T(h)$ está flotada sobre la constante 1500 **impulso.mid**.

Según los resultados en la Figura 3.18, el porcentaje del acelerador para mantener el dron en vuelo estacionario es de 49%. Lo que significa que cualquier valor de la función $T(h)$ influye en la elevación del dron.

Utilizando la Ecuación 4.1, se empezaron las pruebas a velocidad mínima hasta alcanzar una velocidad que generó el impulso necesario para vencer la inercia y sustentar el vuelo.

En la Figura ?? se muestran las curvas de la función $(h)T$ vs altura, la curva en color rojo muestra claramente la velocidad excesiva ($Test_0$) respecto a las demás pruebas. Las curvas amarillas de las pruebas $Test_1$ y $Test_2$ muestran menor amplitud que se tradujo en falta de fuerza de elevación, aunque ambas pruebas mostraron agitación en el chasis. La prueba $Test_3$ logró elevación sostenida y una aceleración considerable con una recuperación rápida en la estabilidad, ver Figura 4.5.



Figura 4.5: Dron prototipo en Vuelo

Sin embargo, se pudo observar, en base al código de Arduino, que no logró llegar a la altura esperada, puesto que la condición de salida del bucle de despegue es llegar a los 100cm. y entonces, ejecutar el algoritmo de descenso. Al observar las curvas de velocidad, podemos deducir que el dron alcanzó una altura tal, que la velocidad aplicada a los rotores disminuyó al punto de no produjo impulso suficiente para continuar elevándose.

Considerando *impulso.mid* como el valor de vuelo estacionario, y utilizando la ecuación 4.1, quedaría:

$$T = T_{hover} + T_{max} e^{\left(\frac{-\left(\frac{h}{10}-2\right)^2}{9} \right)} \quad (4.2)$$

La parte no-lineal de la ecuación permite vencer el estado de reposo (inercia) del dron y logra la elevación. Con valores T_{max} y T_{hover} adecuados, lograríamos mantener una velocidad máxima aproximada a la última prueba con éxito y un valor mínimo más alto como se muestra en la figura 4.6. De tal manera que alcance la altura deseada (ver la curva en color azul).

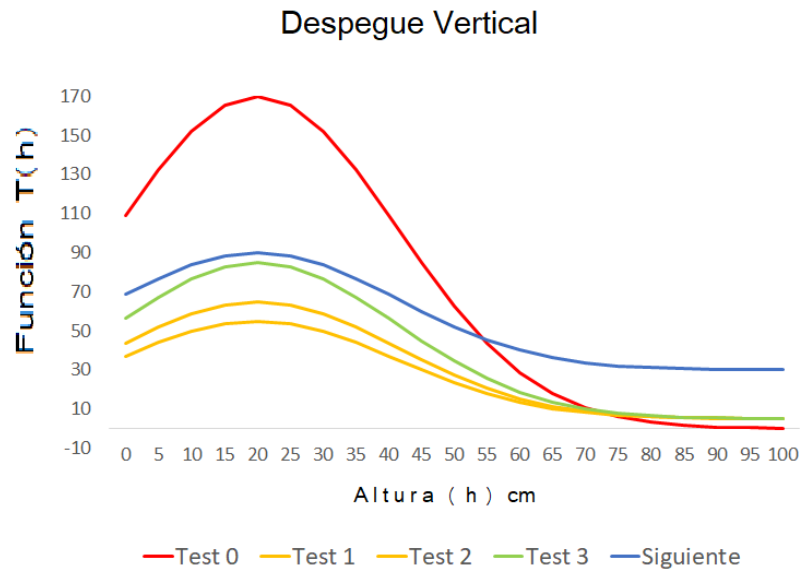


Figura 4.6: Curvas de Velocidad

Por otro lado, en el algoritmo de navegación, el cálculo de la dirección-relativa (orientación del dron respecto al punto de destino) se le insertó una condicionante para compensar el salto de 0° a 360° y viceversa. El ajuste también sirve para que el dron gire hacia el ángulo más corto al punto destino. El código quedó como sigue:

```
// angulo formado por el dron, su destino y el norte magnetico
dir-magnetica = atan2( y, x ) * 180.0 / PI;

// direccion-relativa (angulo de correccion al destino)
dir-relativa = dir-magnetica - dir-brujula();

if (abs( dir-relativa ) > 180.0) // ajuste
    dir-relativa -= signo( dir-relativa )*360.0;

return( dir-relativa );
}
```

Para el cálculo de la distancia entre dos puntos geográficos (dron y destino), se compararon los resultados de dos fórmulas, el *Haversine* y la *Ley de Cosenos* para grandes esferas, dando resultados iguales. Se optó por la ley de cosenos.

Finalmente, la mejor condición de vuelo se da cuando la velocidad del viento está en el rango de $20.7\text{--}22.4\text{ km/h}$. El tiempo de vuelo promedio es $\sim 11.5\text{ min}$ y alcance promedio $\sim 4162\text{ m}$ de acuerdo a las curvas en la Figura 4.7.

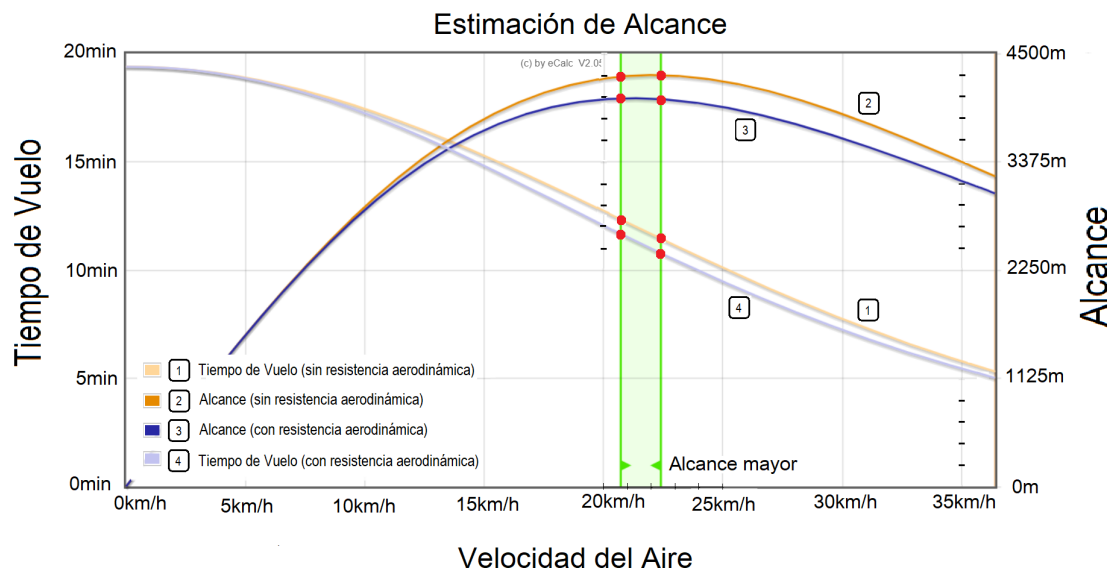


Figura 4.7: Estimación de Tiempo de Vuelo y Alcance [Diseño propio]

Capítulo 5

Conclusiones

5.1 Conclusiones

El desarrollo de este proyecto de tesis puede dividirse en dos partes: la primera contiene el diseño y caracterización de los componentes. La segunda contiene la integración de la plataforma y el desarrollo de los algoritmos. Con base en lo anterior, fue posible lograr el diseño y construcción de un dron con capacidad de vuelo.

En la primera parte, se hizo un análisis en profundidad para determinar qué tecnología utilizar y obtener un detallado diseño del quadrirrotor. Se realizó un diseño que determinó las necesidades del sistema a construir, así como las características que tendría el microcontrolador y pruebas de operación. La caracterización de componentes y pruebas de interconexión con Arduino se obtuvieron con resultados satisfactorios. Cada elemento arrojó los resultados esperados.

En la segunda parte, podemos decir que la construcción del quadrirrotor ha sido un reto y un éxito, fueron muchas piezas a considerar al mismo tiempo. Al progresar en la construcción del chasis se tuvo que retroceder en varias ocasiones para instalar piezas que no podrían ser instaladas posteriormente. En cuanto a la electrónica, fue un éxito, se consiguió integrar casi todos los componentes: microcontrolador, Bluetooth, sensores, ESCs, motores, microSD, sensor ultrasónico, placa de distribución de potencia, el controlador

de vuelo Naza, GPS+Brújula y el controlador de voltaje.

En el desarrollo de los algoritmos se codificaron los siguientes:

- Modificaciones a la librería *NazaDecoderLib.h*, más adelante se dan los detalles.
- Calibración de ESCs.
- Calibración de puertos Naza.
- Armado de motores.
- Despegue Vertical y la función de $T(h)$.

El Control manual de un dron es una experiencia desarrollada de funciones mentales, adquiridas a través de la práctica (retroalimentación) (Lupón *et al.*, 2012). En una tarea aparentemente sencilla como el despegue vertical, no es solamente mandarle la señal de aceleración al controlador de vuelo, sino que el piloto va observando el comportamiento de la unidad aérea mientras maniobra los controles. De esta manera se da la retroalimentación. La generación automática de señales de control para enviarlas al dron parece sencillo, pero se requirió un primer intento como punto de referencia y más investigación.

El trabajo aquí realizado ofrece una excelente oportunidad como base para aplicaciones que estén relacionadas con uso de drones. Se consiguió construir un dron que tiene la capacidad de velocidad de vuelo óptimo de 21 km/hr cubriendo una distancia de 4.1 Km en 11.5 min . Tiempo estimado de vuelo estacionario de 19.4 min , que puede soportar una carga extra máxima de 2.8 kg . y que tiene una arquitectura y programación de código-abierto, adaptable completamente a cualquier tipo de investigación. La tabla 5.1 muestra los precios de venta en los Estados Unidos para que la comparación fuera similar.

En la comparación los tres drones son de 650 mm de tamaño, el controlador de vuelo es de la misma marca comercial y son quadricópteros. El *DJI Matrice 200* es un dron profesional para uso de inspección en lugares difíciles de alcanzar, como las torres eléctricas de alto voltaje, los generadores eólicos,

construcciones de edificios altos, es un sistema robusto, tiene tres sensores de temperatura, dos unidades de medición inercial, tres barómetros, sonar al frente y arriba, tres sensores GPS RTK con precisión de centímetros. Es un sistema sofisticado que aunque el prototipo no cuenta, se le puede equipar con esas características de manera similar. El *DJI Matrice 100* es más comparable con el prototipo. Este modelo dirigido a los desarrolladores/investigadores, es una plataforma a la que se pueden soldar sensores y otros componentes electrónicos y el controlador es programable. Definitivamente este modelo es el homólogo del dron prototipo aquí desarrollado, otras características de vuelo y rendimiento son también similares. La diferencia en precio es considerable. Por otro lado, dron prototipo no cuenta con control remoto. Aunque un control remoto de 10 canales estándar, tiene un precio de $\sim USD60$.

Tabla 5.1: Comparación de precios de Drones

Plataforma/Modelo	Precio
DJI Matrice 100	USD 3,299
DJI Matrice 200	USD 4,999
Prototipo	USD 522

Los tres drones están basados en controladores de vuelo de la marca DJI.

5.2 Trabajo Futuro

Como trabajo futuro se plantean las siguientes líneas de investigación:

1. Comparar la respuesta RPMs vs voltaje y corriente para cada motor.
2. Diseñar un mecanismo con un dinamómetro para medir de fuerza de impulso vs RPMs y valores eléctricos para cada motor.

3. Evaluar y mejorar el algoritmo de Despegue. Compararlo con un controlador PID.
4. Comparar los escenarios de prueba de vuelo y navegación
5. Desarrollar una App más completa que incluya mapas.
6. Determinar un controlador PID para sostener una altura establecida.
7. Mejorar los niveles de ruido integrando la electrónica en un circuito impreso.
8. Mejorar la estructura del dron para hacer más seguro el vuelo.

5.3 Limitaciones

1. El altímetro **MPL3115A2** se tuvo que excluir del proyecto. Este se comunica a través del protocolo I2C, usando la librería `<Wire.h>` de ArduinoDue. Este dispositivo sería utilizado para mantener la altitud durante el vuelo. No queda claro cuál es el problema, entre la librería y el hardware ([Site Admin, Adafruit, 2018](#)), donde el microcontrolador no puede reconocer al dispositivo. Las nuevas opciones eran, por un lado, la unidad GPS del Naza, como hemos comentado, tiene el receptor GPS y un magnetómetro (brújula) y también tiene un altímetro integrado, pero no se encontraron recursos que nos dieran acceso a los datos del altímetro. Por otro lado, teníamos el sonar, que sería utilizado opcionalmente para detección de obstáculos. Los datos que pueda arrojar este sensor tendrán que ser evaluados en condiciones externas. La otra opción era utilizar la altitud que arrojan los datos del receptor GPS, pero en interiores no hay buena recepción satelital, lo que hace de los datos no confiables. Se escogió al sonar.
2. La librería `NazaDecoderLib.h` se modificó porque presentaba un error de compilación. Después de varios días, se localizó un conflicto en las Ruti-

nas de Servicio de Interrupción (ISR) del gimbal de la cámara, se deshabilitaron esas interrupciones para que permitiera funcionar la librería.

3. Faltó tiempo para realizar más pruebas de funcionalidad (giros, movimientos laterales y frontales, altitud, repetibilidad, etc.).
4. Retraso en proceso de compra del equipo. Debido a que el chasis del dron original solicitado se discontinuó (ver sección 3.1 página 94), se inició un nuevo proceso de compra atrasando el ensamble del prototipo. Tiempo de entrega de 142 días (02/Feb/17 al 7/Jun/17).
5. Adelantamiento de un semestre para defender la tesis.

Anexo A

Ensamble del Chasis

En este anexo se revisaran los pasos para el armado del chasis:

- *Acoplamiento Motor-Base*
- *Montaje Motor-Brazo y Alambrado*
- *Ensamble Brazos-Base Inferior*
- *Tipos de tubos*
- *Ensamble Tren de aterrizaje y Gimbal*
- *Ensamble Base Superior y Base de Batería*

A.1 Ensamble del Chasis

A continuación se muestran las imágenes para el armado del chasis del quadricóptero. Las Figuras [A.1](#) y [A.2](#) muestran los diagramas del ensamble general y núcleo del chasis. Con estos diagramas nos damos una buena idea de cómo empezar a armar el chasis.

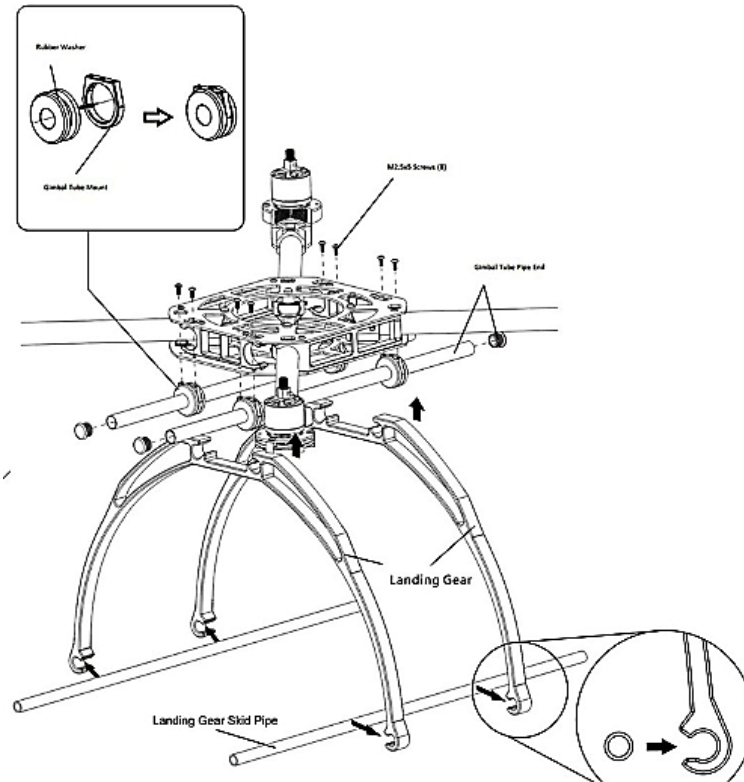


Figura A.1: Diagrama de ensamblaje general del chasis ([unmannedtech, 2012](#))

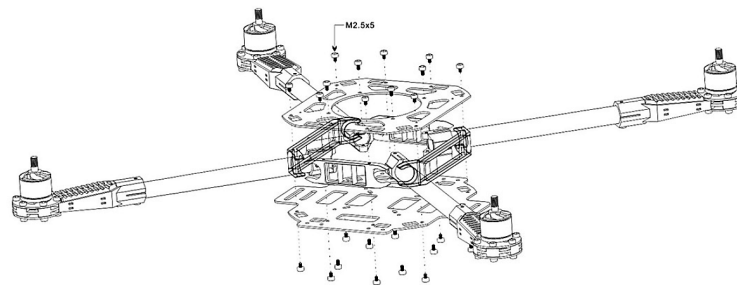


Figura A.2: Diagrama de ensamblaje del núcleo ([unmannedtech, 2012](#))

A.2 Acoplamiento Motor-Base

Empezamos con el acoplamiento de los motores en sus bases para cada uno de los cuatro motores. Ver Figuras A.3 y A.4. Se debe buscar una posición del motor adecuada para que los cables queden hacia el lado opuesto de la base.

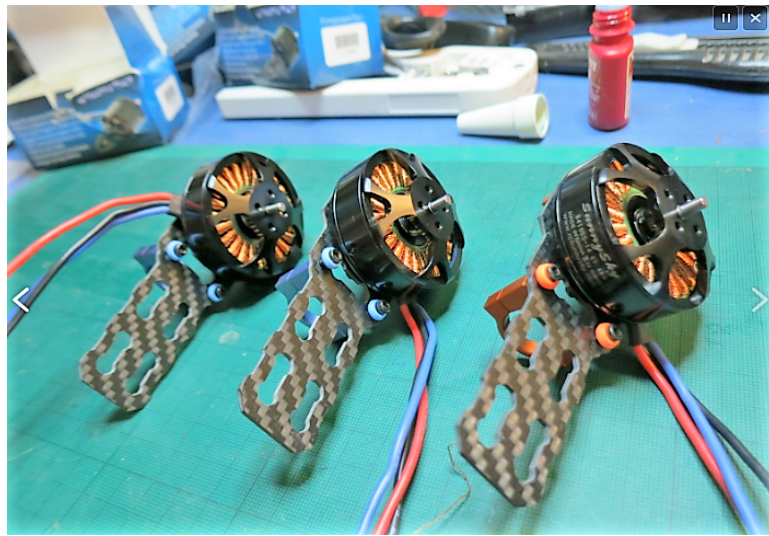


Figura A.3: Armado de Bases y Motores

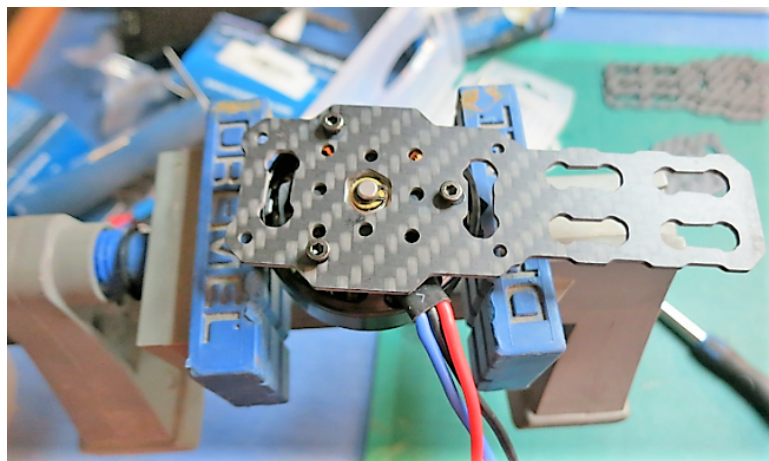


Figura A.4: Montaje de motor en Base

A.3 Montaje Motor-Brazo y Alambrado

Se pasan los cables por el interior de los brazos del chasis y se ajustan las abrazaderas al brazo, ver Figura A.5.

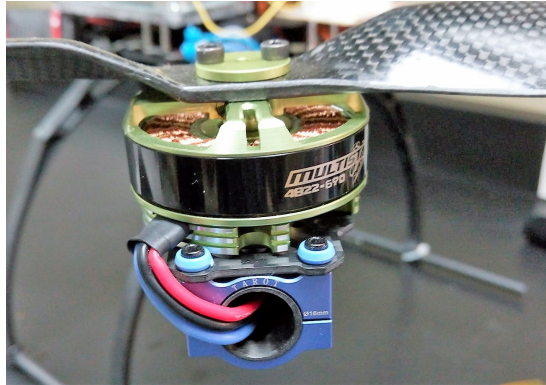


Figura A.5: Montaje del Motor orientado

A.4 Ensamble Brazos-Base Inferior

Como se muestra en el diagrama de ensamble de la base inferior de la Figura A.6, fijar los brazos a la base inferior de tal manera que queden como en la Figura A.7

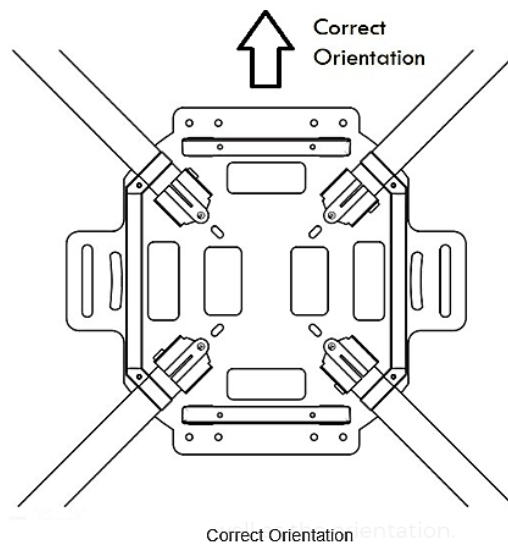


Figura A.6: Diagrama de ensamblaje de Base inferior ([unmannedtech, 2012](#))

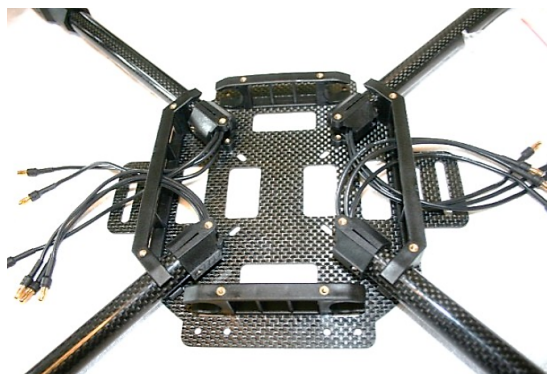


Figura A.7: Ensamblaje de la base inferior ([unmannedtech, 2012](#))

A.5 Tipos de tubos

Tener en cuenta los cuatro tubos de fibra de carbono restantes, los de menor diámetro son para el tren de aterrizaje y los de mayor diámetro son para el gimbal, ver Figura A.8.



Figura A.8: Tubos de gimbal y del tren de aterrizaje ([unmannedtech, 2012](#))

A.6 Ensamble Tren de aterrizaje y Gimbal

Instalar los tubos del tren de aterrizaje y del gimbal, según a Figura A.1 y colocar la base para el gimbal como se muestra en la Figura A.9

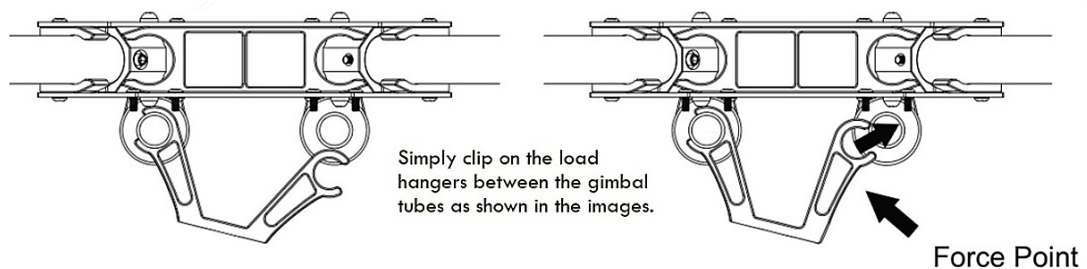


Figura A.9: Colocacion de la Base de Gimbal ([unmannedtech, 2012](#))

A continuación fijar el tren de aterrizaje del prototipo como se muestra en la Figura A.10.

A. ENSAMBLE DEL CHASIS A.7 Ensamble Base Superior y Base de Batería

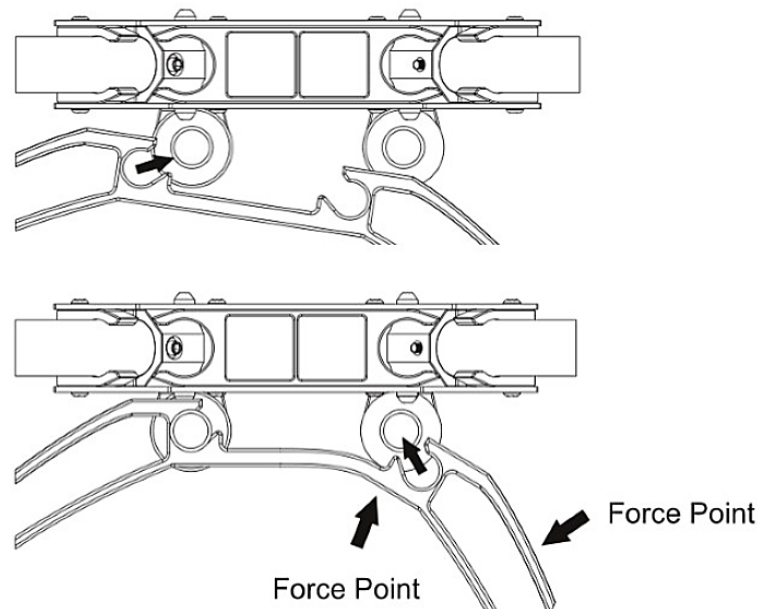


Figura A.10: Colocacion del Tren de Aterrizaje ([unmannedtech, 2012](#))

A.7 Ensamble Base Superior y Base de Batería

Antes de instalar la base superior del núcleo y la base de batería, se conectaron los ESCs a los cables del motor y a la placa de distribución de energía que va colocada en el centro de la base inferior del núcleo del chasis. Se conectan también los cables de alimentación para la unidad de monitoreo de voltaje del controlador Naza. Finalmente se puede colocar la base superior y sobre esta, instalada la unidad Naza. La Figura A.11 muestra el ensamble de la base superior y de la base de la batería.

A. ENSAMBLE DEL CHASIS A.7 Ensamble Base Superior y Base de Batería

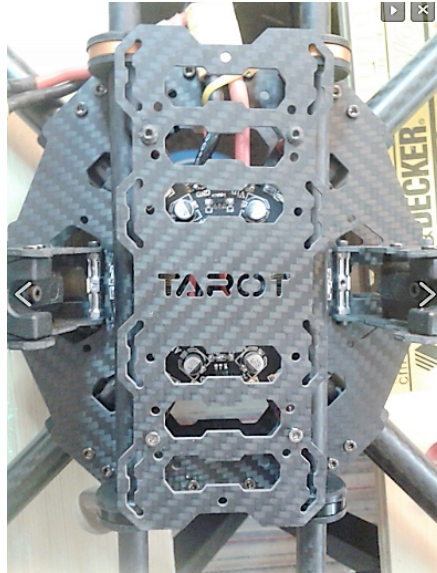


Figura A.11: Montaje de la base de la bateria ([unmannedtech](#), 2012)

El controlador Naza quedó debajo de la base de la bateria, en la siguiente Figura se ve esto, la bateria no esta colocada. En la Figura

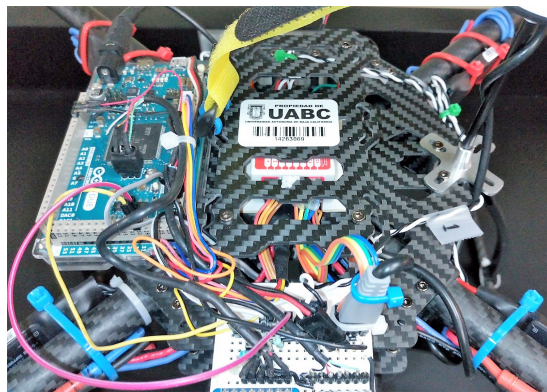


Figura A.12: Ensamble Final del núcleo del Prototipo

A. ENSAMBLE DEL CHASIS A.7 Ensamble Base Superior y Base de Batería

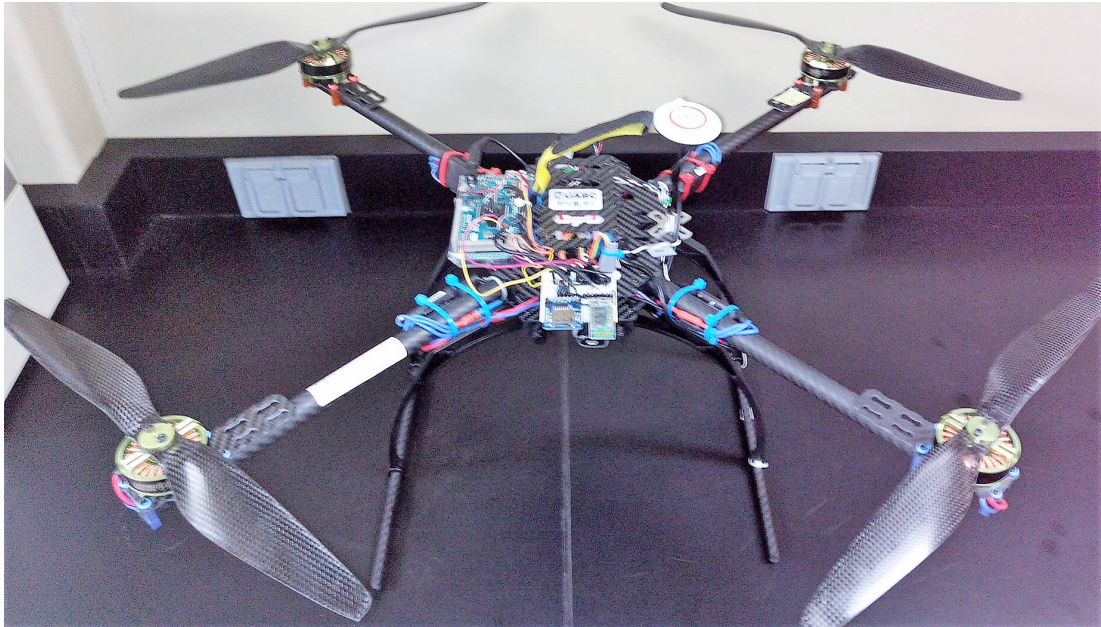


Figura A.13: Ensamble Final del Prototipo

Anexo B

Código

En este anexo se encuentra el siguiente código utilizado:

- *Calibración de ESC*
- *Armar los Motores*
- *Despegue Vertical*
- *Calibración de Puertos Naza*
- *Direccion-relativa y Distancia*

B.1 Calibración de ESC

```
#include <Servo.h>
#define MAX_SIGNAL 2000
#define MIN_SIGNAL 1000
#define motorPin 12
Servo motor;

void setup() {
```

```
Serial.begin(9600);
Serial.println("Program begin...");
Serial.println("This program will calibrate the ESC.");
Serial.println("press any key to continue...");

motor.attach(motorPin);
// while (!Serial.available());
// Serial.read();

Serial.println("Now writing maximum output.");
Serial.print("Turn on power source, ");
Serial.println("then wait 2 seconds and press any key.");
motor.writeMicroseconds(MAX_SIGNAL);

// Wait for input
while (!Serial.available());
Serial.read();
Serial.println("Sending minimum output");
motor.writeMicroseconds(MIN_SIGNAL);

}

void loop() {
Serial.println("done.");
while (1){};
}
```

B.2 Armar los motores

```
#include <Servo.h>
#include <SPI.h>
#include <NazaDecoderLib.h>

#define MINTHROTTLE 1000
```

```
#define MAXTHRATTLE 2000

int i=0;
struct vuelta {
int down = MINTHRATTLE;
int slowdown = 1350;
int mid = 1500;
int slowup = 1600;
int up = MAXTHRATTLE;
};
vuelta roll;

struct elev {
int down = MINTHRATTLE;
int slowdown = 1350;
int mid = 1500;
int slowup = 1600;
int up = MAXTHRATTLE;
};
elev pitch;

struct impul {
int down = MINTHRATTLE;
int slowdown = 1350;
int mid = 1500;
int slowup = 1600;
int up = MAXTHRATTLE;
};
impul impulso;

struct direccion {
int down = MINTHRATTLE;
int slowdown = 1350;
int mid = 1500;
int slowup = 1600;
```

```
int up = MAXTHROTTLE;
};
direccion timon;

struct nav {
int manual = 1185;
int attitude = 1525;
int gps = 1860;
};
nav modo;

// DJI Naza accepts input in the form of commands
// from the Arduino Servo Library
Servo throttle;
Servo rudder;
Servo aileron;
Servo elevator;
Servo gear;

int throttlePin = 5;
int rudderPin = 6;
int elePin = 4;
int ailPin = 3;
int gearPin = 7;
int SD_ChipSelect = 10;
boolean armed = false;
int buttonPin = 31;
int buttonState = 0;

void setup() {

  pinMode(buttonPin, INPUT);

  Serial.begin(9600); // PC monitor
```

```
Serial.println();
Serial.println("pin_assignment...");

throttle.attach(throttlePin);
rudder.attach(rudderPin);
aileron.attach(ailPin);
elevator.attach(elePin);
gear.attach(gearPin);

Serial.println("zero_thrust_and_neutral_channels...");

throttle.writeMicroseconds(impulso.down);
aileron.writeMicroseconds(roll.mid);
rudder.writeMicroseconds(timon.mid);
elevator.writeMicroseconds(pitch.mid);
gear.writeMicroseconds(modos.gps);

// check if the pushbutton is pressed. If it is,
// the buttonState is HIGH:

while (buttonState != HIGH) {
    buttonState = digitalRead(buttonPin);
}
Serial.println("press_any_key_to_arm_motors.");
delay(7000);

Serial.println("arming...");

for (int i = 0; i < 50; i++) {
    throttle.writeMicroseconds(impulso.down);
    aileron.writeMicroseconds(roll.up);
    rudder.writeMicroseconds(timon.down);
    elevator.writeMicroseconds(pitch.down);
    delay(20);
}
```

```
Serial.println("armed...");
delay(12000);

\\ acelera un poco
Serial.println("mid_controls");
throttle.writeMicroseconds(impulso.mid);
aileron.writeMicroseconds(roll.mid);
rudder.writeMicroseconds(timon.mid);
elevator.writeMicroseconds(pitch.mid);
gear.writeMicroseconds(modos.gps);
delay(15000);

\\ desacelera completamente.
throttle.writeMicroseconds(impulso.down);
}
void loop(){}

```

B.3 Despegue Vertical

```
//Libreria para enviar PWMs a los Servos.
#include <Servo.h>

#define MINTHROTTLE 1000
#define MAXTHROTTLE 2000
#define HOVERTHROTTLE 1550
#define T_max 80
#define T_min 5

//Defines so the device can do a self reset
#define SYSRESETREQ (1<<2)
#define VECTKEY (0x05fa0000UL)
#define VECTKEY_MASK (0x0000ffffUL)
#define AIRCR *(uint32_t*)0xe00ed0cUL)
#define REQUEST_EXTERNAL_RESET \

```



```
(AIRCR=(AIRCR&VECTKEY_MASK)|VECTKEY|SYSRESETREQ)

int sonarpin = 20;
unsigned long A[7],duration, range;
int pointer = 0;
int i=0;
struct vuelta {
int down = MINTHROTTLE;
int slowdown = 1350;
int mid = 1500;
int slowup = 1600;
int up = MAXTHROTTLE;
};
vuelta roll;

struct elev {
int down = MINTHROTTLE;
int slowdown = 1350;
int mid = 1500;
int slowup = 1600;
int up = MAXTHROTTLE;
};
elev pitch;

struct impul {
int down = MINTHROTTLE;
int slowdown = 1460;
int mid = HOVERTHROTTLE;
int slowup = 1600;
int up = MAXTHROTTLE;
};
impul impulso;

struct direccion {
int down = MINTHROTTLE;
```

```
int slowdown = 1350;
int mid = 1500;
int slowup = 1600;
int up = MAXTHROTTLE;
};
direccion timon;

struct nav {
int manual = 1185;
int attitude = 1525;
int gps = 1860;
};
nav modo;

// DJI Naza accepts input in the form of commands
// from the Arduino Servo Library

Servo throttle;
Servo rudder;
Servo aileron;
Servo elevator;
Servo gear;

int interruptPin = 52;
int throttlePin = 5;
int rudderPin = 6;
int elePin = 4;
int ailPin = 3;
int gearPin = 7;
int SD_ChipSelect = 10;
int w = 7;
boolean armed = false, init_flag = true;
unsigned long mean=0,
resultado = 0,
init_state = 0;
```

```
int buttonPin = 31;
int ledPin = 30;
int buttonState = 0;
int throttle_delta;

void setup() {

  pinMode(buttonPin, INPUT);
  pinMode(sonarpin, INPUT);
  pinMode(ledPin, OUTPUT );
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(interruptPin, blink, CHANGE);

  Serial.begin(9600); // monitor
  Serial.println();
  Serial.println("pin_assignment...");

  throttle.attach(throttlePin);
  rudder.attach(rudderPin);
  aileron.attach(ailPin);
  elevator.attach(elePin);
  gear.attach(gearPin);

  Serial.println("zero_thrust_and_neutral_channels...");
  throttle.writeMicroseconds(impulso.down);
  aileron.writeMicroseconds(roll.mid);
  rudder.writeMicroseconds(timon.mid);
  elevator.writeMicroseconds(pitch.mid);
  gear.writeMicroseconds(modos.attitude);

  Serial.println("turn_off_and_on_main_switch_and");
  Serial.println("press_any_key_to_arm_motors.");

  // check if the pushbutton is pressed.
```

```
// If it is, the buttonState is HIGH:
while (buttonState != HIGH)
buttonState = digitalRead(buttonPin);

// while (!Serial.available());
// Serial.read();

delay(7000);
Serial.println("arming...");
for (int i = 0; i < 50; i++) {
throttle.writeMicroseconds(impulso.down);
aileron.writeMicroseconds(roll.up);
rudder.writeMicroseconds(timon.down);
elevator.writeMicroseconds(pitch.down);
delay(20);
}
Serial.println("armed...");

Serial.println("mid_controls");
throttle.writeMicroseconds(impulso.mid);
aileron.writeMicroseconds(roll.mid);
rudder.writeMicroseconds(timon.mid);
elevator.writeMicroseconds(pitch.mid);
gear.writeMicroseconds(modo.attitude);
delay(5000);

// el control se mantendra en este ciclo mientras
// no alcance la altura de 100 cms.

while ( mean < 100 ) {

// funcion de velocidad
thrust = (int)((T_max*exp(-sq(mean/10-2)/9))+T_min);
throttle.writeMicroseconds( impulso.mid + thrust );
```

```
resultado = 0;
duration = pulseIn(sonarpin, HIGH);
range = (unsigned long)(duration/58); // 58uS/cm.

// filtro de media (w=7) para lecturas
// del sensor ultrasonico

A[ pointer++ ] = range;
if (pointer==w) pointer = 0;
for ( int i=0; i<w; i++ ) resultado += A[i];
mean = (unsigned long)(resultado/w);
if (init_flag && pointer == 6) {
    init_flag = false;
    init_state = mean;
}
}
throttle.writeMicroseconds( impulso.mid );
delay(5000);
}

void loop() {

while ( mean >= init_state ) {

thrust = (int)((T_max*exp(-sq(mean/10-2)/9))+T_min);
throttle.writeMicroseconds( impulso.mid - thrust );

// filtro de media (w=7) para lecturas
// del sensor ultrasonico

resultado = 0;
duration = pulseIn(sonarpin, HIGH);
range = (unsigned long) (duration/58); // 58uS/cm.
A[ pointer++ ] = range;
if (pointer==w) pointer = 0;
```

```
for ( int i=0; i<w; i++ ) resultado += A[i];
mean = (unsigned long)(resultado/w);
}
Serial.println( "Landed!" );
delay(5000);
throttle.writeMicroseconds(impulso.down);
while (1){};
}

void blink (void){
    REQUEST_EXTERNAL_RESET;
}
```

B.4 Calibración de Puertos Naza

```
#include <Servo.h>
#define MAXTHROTTLE 2000
#define MINTHROTTLE 1000
Servo throttle;
Servo rudder;
Servo aileron;
Servo elevator;
Servo gear;

int throttlePin = 5,
rudderPin = 6,
elePin = 4,
ailPin = 3,
gearPin = 7,
blueLedPin = 8,
SD_ChipSelect = 10,
i;

struct impul {
int down = MINTHROTTLE;
```

```
int mid = 1500;
int up = MAXTHROTTLE;
int recommended = 1176;
};
impul impulso;

struct vuelta {
int down = MINTHROTTLE;
int mid = 1500;
int up = MAXTHROTTLE;
};
vuelta roll;

struct direccion {
int down = MINTHROTTLE;
int mid = 1500;
int up = MAXTHROTTLE;
};
direccion timon;

struct elev {
int down = MINTHROTTLE;
int mid = 1500;
int up = MAXTHROTTLE;
};
elev pitch;

struct nav {
int manual = 1186;
int attitude = 1524;
int gps = 1862;
};
nav modo;

void setup(){
```

```
int ciclos = 5;
throttle.attach(throttlePin);
rudder.attach(rudderPin);
aileron.attach(ailPin);
elevator.attach(elePin);
gear.attach(gearPin);
Serial.begin(9600);

Serial.println("power_OFF_Naza_MC_and");
Serial.println("press_any_key_to_continue...");
while (!Serial.available());
Serial.read();
delay(1000);

Serial.println("setting_channels_to_center...");
aileron.writeMicroseconds(roll.mid);
elevator.writeMicroseconds(pitch.mid);
throttle.writeMicroseconds(impulso.mid);
rudder.writeMicroseconds(timon.mid);
gear.writeMicroseconds(modos.attitude);

Serial.println("power_ON_Naza_MC_and");
Serial.println("press_any_key_to_continue...");
while (!Serial.available());
Serial.read();
delay(1000);

Serial.println("setting_channels_up...");
aileron.writeMicroseconds(roll.up);
elevator.writeMicroseconds(pitch.up);
throttle.writeMicroseconds(impulso.up);
rudder.writeMicroseconds(timon.up);
gear.writeMicroseconds(modos.gps);
```



```
Serial.println("...press_Naza_[Start]_and_");
Serial.println("press_any_key_to_begin_calibration.");
while (!Serial.available());
Serial.read();
delay(1000);

aileron.writeMicroseconds(roll.up);
elevator.writeMicroseconds(pitch.up);
throttle.writeMicroseconds(impulso.up);
rudder.writeMicroseconds(timon.up);
gear.writeMicroseconds(modos.gps);

Serial.println("Aileron");
for (int j=0; j<ciclos; j++) {
  for (int i=roll.up; i>roll.down; i--) {
    aileron.writeMicroseconds(i);
    elevator.writeMicroseconds(i);
    throttle.writeMicroseconds(i);
    rudder.writeMicroseconds(i);
    Serial.print(j);
  }
  Serial.println();
  Serial.println("...press_any_key_to_finish_calibration.");
  while (!Serial.available());
  Serial.read();

  aileron.writeMicroseconds(roll.mid);
  elevator.writeMicroseconds(pitch.mid);
  throttle.writeMicroseconds(impulso.mid);
  rudder.writeMicroseconds(timon.mid);
  gear.writeMicroseconds(modos.attitude);
}

void loop() {
  Serial.println("done.");
```

```
Serial.end();  
while(1);  
}
```

B.5 Dirección-Relativa y Distancia

```
double courseTo( double Lat1, double Lon1,  
                 double Lat2, double Lon2 ) {  
  
    // declaracion de variables  
    double x, y, bear_true, rel_bear;  
  
    //convertir parametros de entrada a radianes.  
    Lat1 *= PI / 180.0;  
    Lon1 *= PI / 180.0;  
    Lat2 *= PI / 180.0;  
    Lon2 *= PI / 180.0;  
  
    // calculo de x,y segun las coordenadas geograficas  
    x = cos(Lat1)*sin(Lat2)-sin(Lat1)*  
        cos(Lat2)*cos(Lon2-Lon1);  
    y = sin(Lon2-Lon1)*cos(Lat2);  
  
    // angulo formado por el dron, su destino y el norte magnetico  
    bear_true = atan2( y, x ) * 180.0 / PI;  
  
    // angulo de correccion del dron respecto al destino  
    rel_bear = bear_true - CompassHeading();  
  
    if (abs(rel_bear)>180.0) {  
        rel_bear -= sgn(rel_bear)*360.0;  
    }  
    return(rel_bear );  
}
```

```
// distancia calculada usando la ley de cosenos
// para grandes esferas.
double cosDistance( Lat1, Lon1, Lat2, Lon2 ){

double dist;
//convertir a radianes
Lat1 *= PI / 180.0;
Lon1 *= PI / 180.0;
Lat2 *= PI / 180.0;
Lon2 *= PI / 180.0;

dist = acos( sin(Lat1)*sin(Lat2) +
             cos(Lat1)*cos(Lat2)*cos(Lon2-Lon1) ) * 6371000;

return( dist );
}
```

Abreviaciones

a posteriori, 36	DOP, 45
a priori, 36	dron, 3 , 4 , 6 , 9 , 10 , 70
ACK, 67	drones, 6
ADC, 52	DSI, 116
App, 124 , 126	DSP, 86
ARM, 115 , 116	E-frame, 23 , 27 , 31
ASCII, 43 , 56 , 111 , 126	eCalc, 98–100 , 103
Atmega328, 60	EEPROM, 46 , 49 , 87
B-frame, 23 , 25 , 27 , 31 , 33	ERAST, 1
BAC, 78	ESC, 5 , 69 , 75 , 77 , 78 , 94–96 , 98 , 100 , 101 , 112 , 118 , 127 , 163
BLE, 116	
bps, 55	
C, 72 , 90	FC, 79 , 83
código-abierto, 81	FIFO, 61
CLK, 63 , 109	FPV, 68 , 83
CLOUD-MAP, 12	
CPU, 46 , 47 , 86	GND, 58
CS, 109	GP, 43
CSI, 116	GPGGA, 43
DC, 94	GPIO, 116
DDC, 132	GPS, 4 , 15 , 41–43 , 45 , 54 , 69 , 79 , 83 , 84 , 87 , 112 , 115 , 118 , 124 , 126 , 131 , 133
DGPS, 44	GUI, 80
DOF, 19 , 23 , 24 , 31 , 79	

H-frame, 31 , 32	N, 139
HCl, 85	NACK, 67
HDOP, 44	NASA, 1
hPa, 100	Naza, 112 , 114 , 135 , 160 , 174
HTOL, 8	NBR, 9
I2C, 54 , 66 , 67 , 116 , 117 , 121	NiCd, 71
IC, 59	NiHm, 71
ICU, 2 , 3	NMEA, 42 , 43
IDE, 120 – 122	open-source, 81 , 82
IMU, 79 , 112	Outrunner, 75
Inrunner, 75	
ISM, 85	PC, 122
ISP, 46	PDB, 118
ISR, 9	PGRM, 43
	PIC, 115
Kalman, 34 , 36 , 38 , 39 , 41	PID, 15 , 80
KV, 94 , 106 , 142	Pitch, 21
Kv, 74	pitch, 27
	PMGN, 43
L, 90	PWM, 51 , 114 – 116 , 127 , 134 , 136
LC, 86	
LCD, 54	RC, 83 , 112 , 134
LED, 79 , 83	Roll, 20
LiPo, 70 – 72 , 95 , 96	roll, 27
LNA, 84	RPM, 74 , 106
LQR, 15	RTF, 69
LST, 11	RTH, 81 , 83 , 84
LULC, 11	RTK, 43 , 44
	RX, 57 , 59 , 86 , 107 , 110
mAh, 71	
MC, 83 , 113	S, 71
MISO, 63 , 109	SCK, 63
MIT, 124	SCL, 67
MOSI, 63 , 109	

ABREVIACIONES

ABREVIACIONES

SD, 87, 88, 108, 109, 114–116, 130,
136
SDA, 67
SMA, 84
SPI, 54, 61–64, 66, 84, 88, 108, 116,
117, 121, 132
SS, 65, 66, 108
T, 90
Throttle, 20, 134
TTL, 57, 58, 86
TWI, 116
TX, 57, 59, 86, 106, 110
UART, 59–61, 66, 85, 106, 107, 112,

116, 117, 121, 132
UAV, 1
URL, 93
USB, 54, 83, 85, 116, 132, 144
VANT, 1
VCC, 58, 86
VTOL, 8
WAAS, 44
WIFI, 86, 116
XBee, 54
Yaw, 22

Notaciones

$0_{3 \times 1}$ vector vertical de ceros. [27](#)

A matriz de $n \times n$. [35](#)

B matriz de $n \times l$. [35](#)

B número de hojas que tiene la hélice. [78](#)

$C_B(\nu)$ matriz Coriolis-centripeta w.r.t. *B-frame*. [25](#), [32](#)

$C_H(\zeta)$ matriz Coriolis-centripeta w.r.t. *H-frame*. [32](#)

C_x coseno de x . [17](#)

E_B matriz de movimiento. [28](#), [29](#), [33](#)

$E_H(\xi)$ matriz de movimiento w.r.t. *H-frame*. [33](#)

F_G^B vector de la fuerza gravitacional w.r.t. *B-frame*. [27](#)

F_G^E vector de la fuerza gravitacional w.r.t. *E-frame*. [27](#)

$G_B(\xi)$ vector gravitacional. [26](#)

G_H vector gravitacional w.r.t. *H-frame*. [32](#)

I matriz diagonal. *vease: matriz inercia I*, [17](#)

I matriz de inercia. [24](#)

$I_{3 \times 3}$ matriz identidad. [24](#)

I_M	momento inercial del rotor. 18
J_Θ	matriz generalizada. 24
J_{TP}	momento de inercia rotacional total. 28
K	vector genérico de tres dimensiones. 26
L	longitud de la hélice. 78
M_B	matriz de inercia del sistema w.r.t. <i>B-frame</i> . 25 , 31
M_H	matriz de inercia del sistema w.r.t. <i>H-frame</i> . 31
$O_{3 \times 3}$	sub-matriz de ceros. 24
$O_B(v)$	matriz de hélices giroscópica. 28
$O_H(\zeta)$	matriz giroscópica de las hélices w.r.t. <i>H-frame</i> . 32
P	pitch. 78
Q	matriz de covarianza del ruido de proceso. 35
R	matriz de rotación. 17
R	matriz de covarianza del ruido de medición. 35
$R_\Theta[-]$	matriz de rotación. 24 , 27 , 33
$S(\cdot)$	operador anti-simétrico. 26
$S(k)$	matriz anti-simétrica de k . 26
S_x	seno de x . 17
T	impulso. 18
T_Θ	matriz de transferencia o transformación. 24
T_x	tangente de x . 17
U_1	comando impulso (<i>throttle</i>). 20 , 29 , 33

U_2 comando vuelta o movimiento lateral (*roll*). 20, 29

U_3 comando inclinación o movimiento frente-atrás (*pitch*). 21, 29

U_4 comando giro (*yaw*). 22, 29

$U_B(\Omega)$ vector de movimiento. 28, 29

V^B vector de velocidad lineal. 24

V_B velocidades lineales. 16

W_η matriz de transferencia para velocidades angulares. 17

Δ_A incremento de velocidad de la hélice. 20, 21, 22, 23

Δ_B decremento de velocidad de la hélice. 20, 21, 22, 23

Γ^E vector de posición linear. 24

Λ vector de fuerza generalizado. 25, 26

Ω vector de velocidad de las hélices. 28, 29

Ω_1 velocidad de la hélice frontal. 28

Ω_2 velocidad de la hélice derecha. 28

Ω_3 velocidad de la hélice trasera. 28

Ω_4 velocidad de la hélice izquierda. 28

Ω_H velocidad de vuelo estacionario (Hovering). 19, 20

Ω_Σ velocidad total de las hélices. 28

Θ^E vector de posición angular. 24

$\dot{V}^B$ vector de aceleración lineal w.r.t. *B-frame*. 24

$\dot{\Omega}$ vector de aceleración de las hélices. 29

$\dot{\nu}$ vector de aceleración generalizado w.r.t. *B-frame*. 25, 30

$\dot{\omega}^B$ vector de aceleración angular w.r.t. *B-frame*. 24

$\dot{\omega}_i$ aceleración angular. 18

$\dot{\xi}$ vector de velocidad generalizado w.r.t. E-frame. 24

$\dot{\zeta}$ vector generalizado de aceleración del quadrirrotor w.r.t. *H-frame*. 31

$\dot{\zeta}$ vector de aceleración generalizado w.r.t. H-frame. 33

η vector de posición angular. 15

ν vector de velocidad generalizado w.r.t. B-frame. 24

ω^B vector de velocidad angular. 24

ω_i velocidad angular del rotor *i*. 17

ϕ ángulo para Roll. 15

ψ ángulo para Yaw. 15

τ^B vector de torques w.r.t. *B-frame*. 24

τ_B torque. 18

τ_{M_i} torque del eje del rotor *i*. 17

τ_ϕ torque de ϕ . 18

τ_ψ torque de ψ . 18

τ_θ torque de θ . 18

θ ángulo para Pitch. 15, 17

θ, ϕ, ψ los ejes angulares del sistema inercial. 15

ξ vector de posición lineal. 15

ξ vector de posición generalizado w.r.t. E-frame. 24

ζ vector generalizado de velocidad w.r.t. *H-frame*. 31

b constante de arrastre. 18, 28

b constante de impulso. 28

f_i fuerza producida por el motor i . 15, 17

g aceleración de la gravedad de la tierra. 26

k constante de ascenso. 15, 18

k tiempo actual. 35

$k - 1$ tiempo previo. 35

l distancia entre el rotor y el centro de la masa del quadrirrotor. 18, 29

m masa del cuerpo. 24

q vector de posiciones lineales y angulares. 15

v velocidades angulares. 16

v_k ruido de la medición. 35

w_k ruido del proceso. 35

w_i la velocidad angular. 15

x, y, z los ejes lineales del sistema inercial. 15

x_k estado de un proceso controlado por tiempo discreto k . 35

z medición en el tiempo discreto k . 35

Referencias

- ADMIN (2014). baterias de litio. <http://bateriasdelitio.net/?p=51>, 25-04-2018. 73, 74
- ADMINISTRADOR (2017). Navegación aérea: rumbo, ruta y derrota. <https://aviatorbluewings.com/navegacion-aerea-rumbo-ruta-derrota/>, 05-05-2018. xiii, 143
- AEROQUAD (2018). Aeroquad. http://www.aeroquadstore.com/AeroQuad_Cyclone_ARF_Kit_p/aqarf-001b.htm, 25-04-2018. xi, 94, 95
- AKBARI, H., POMERANTZ, M. & TAHA, H. (2001). Cool surfaces and shade trees to reduce energy use and improve air quality in urban areas. *Solar energy*, 70, 295–310. 12
- AKRCGUY (2014). Octacoptero skyhawk (rotary wing). <https://diydrones.com/profiles/blogs/steadidrone-releases-new-qu4d-x-with-60-min-flight-time?page=5>, 25-04-2018. ix, 14
- ALEX (UNMANNED TECH) (2013). Q650 quadcopter build idea. <https://www.dronetrest.com/t/q650-quadcopter-build-idea/88>, 26-04-2018. 96
- ARDUINO, A. (2016). Bluetooth en arduino. <https://aprendiendoarduino.wordpress.com/2016/11/13/bluetooth-en-arduino/>, 25-04-2018. 87
- ARDUINO.CC (2018). Arduino Due A000062. <https://www.amazon.com/Arduino-Org-A000062-Due/dp/B00A6C3JN2>, 01-05-2018. 122

- ARDUINO.CC (2018). Arduino due: Overview, tech specs, documentation. <https://store.arduino.cc/usa/arduino-due>, 25-04-2018. xii, 118
- ARDUINO.CC (2018). What is arduino? <https://www.arduino.cc/en/Guide/Introduction>, 01-05-2018. 122
- ARJOMANDI, M., AGOSTINO, S., MAMMONE, M., NELSON, M. & ZHOU, T. (2006). Classification of unmanned aerial vehicles. *Mech Eng*, **3016**. 7, 8
- AUSTIN, R. (2011). *Unmanned aircraft systems: UAVS design, development and deployment*, vol. 54. John Wiley & Sons. 8
- AYUNTAMIENTO DE MEXICALI (2010). Municipio. <http://www.mexicali.gob.mx/22/municipio.php>, 28-04-2018. 101
- BACHMANN, R.J., BORJA, F.J., VAIDYANATHAN, R., IFJU, P.G. & QUINN, R.D. (2009). A biologically inspired micro-vehicle capable of aerial and terrestrial locomotion. *Mechanism and Machine Theory*, **44**, 513–526. 7
- BANGGOOD.COM (2018). Hobbywing platinum 30a pro 2-6s esc opto. https://www.banggood.com/Hobbywing-Platinum-30A-Pro-2-6S-ESC-OPT0-For-Quadcopter-p-89406.html?cur_warehouse=CN, 01-05-2018. xi, xiv, 79, 99, 100
- BARRETT, S.F. & PACK, D.J. (2005). Microcontrollers fundamentals for engineers and scientists. *Synthesis Lectures on Digital Circuits and Systems*, **1**, 1–124. 45, 46
- BISHOP, G., WELCH, G. *et al.* (2001). An introduction to the kalman filter. *Proc of SIGGRAPH, Course*, **8**, 41. x, 34, 38, 40
- BLACKWOOD, Z. & KINGB, G.S. (2018). Vertical take-off unmanned aerial vehicle with forward flight transition. *WEST INDIAN*, 62. 130
- BLUETOOTH (2018). technology. <https://www.bluetooth.com/bluetooth-technology/radio-versions>, 25-04-2018. 86

- BOUABDALLAH, S., NOTH, A. & SIEGWART, R. (2004). PID vs LQ control techniques applied to an indoor micro quadrotor. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3, 2451–2456, IEEE. [15](#)
- BRESCIANI, T. (2008). Modelling, identification and control of a quadrotor helicopter. *MSc Theses*. [ix](#), [19](#), [20](#), [21](#), [22](#), [23](#)
- BROWN, R.G. & HWANG, P.Y. (1997). Introduction to random signals and applied kalman filtering: with matlab exercises and solutions. *Introduction to random signals and applied Kalman filtering: with MATLAB exercises and solutions*, by Brown, Robert Grover.; Hwang, Patrick YC New York: Wiley, c1997.. [39](#)
- CASILLAS, H. (2009). *Climatología de la Isla Urbana de Calor en Mexicali, BC y su relación con el Uso de Suelo*. Ph.D. thesis, Tesis de Maestría, Instituto de Ingeniería, Univ. Autónoma de Baja California, Mexicali, México. [3](#)
- CASILLAS-HIGUERA, A., GARCÍA-CUETO, R., CAMACHO, O. & GONZÁLEZ-NAVARRO, F.F. (2014). Detección de la isla urbana de calor mediante modelado dinámico en Mexicali, bc, México. *Información tecnológica*, **25**, 139–150. [2](#), [13](#)
- CASTILLO, P., LOZANO, R. & DZUL, A. (2005). Stabilization of a mini rotorcraft with four rotors. *IEEE control systems*, **25**, 45–55. [15](#)
- CASTILLO, P., GARCÍA, P., LOZANO, R. & ALBERTOS, P. (2007). Modelado y estabilización de un helicóptero con cuatro rotores. *Revista Iberoamericana de Automática e Informática Industrial RIAI*, **4**, 41–57. [15](#)
- COUTTS, A.M., HARRIS, R.J., PHAN, T., LIVESLEY, S.J., WILLIAMS, N.S. & TAPPER, N.J. (2016). Thermal infrared remote sensing of urban heat: Hotspots, vegetation, and an assessment of techniques for use in urban planning. *Remote Sensing of Environment*, **186**, 637–651. [11](#)
- CUETO, O.R.G. (????). Los sensores remotos y su aplicación en la meteorología: un estudio de caso. [3](#)

DEPRIEST, D. (2013). Nmea data. <http://www.gpsinformation.org/dale/nmea.htm>, 25-04-2018. 43

DÍAZ CARAVANTES, R.E., CASTRO LUQUE, A.L. & ARANDA GALLEGOS, P. (2014). Mortality by excessive natural heat in northwest mexico: Social conditions associated with this cause of death. *Frontera norte*, 26, 155–177. 3

DIEDRO (2015). Drone octa. <http://www.diedro8.com/en/drones-profesionales/598-drone-octa.html>, 25-04-2018. ix, 15

DIKMEN, İ.C., ARISOY, A. & TEMELTAS, H. (2009). Attitude control of a quadrotor. In *Recent Advances in Space Technologies, 2009. RAST'09. 4th International Conference on*, 722–727, IEEE. 15

DJI (2014). Naza-m lite user manual v 2.00. <https://www.dji.com/es/naza-m-lite/download>, 25-04-2018. 137, 138

EALC (2014). How to use this calculator. <https://www.ecalc.ch/calcinclud/help/xcoptercalchelp.htm>, 30-04-2018. 105, 106

EALC (2018). Tarot 1455 carbon propellers (two holes / cw and ccw). <http://www.helipal.com/tarot-1455-carbon-propellers-two-holes-cw-and-ccw.html>, 01-05-2018. xi, 98

ELECTRICAL ENGINEERING (2016). why ultrasonic module sends out 8 cycles ? and , why the trigger pulse is 10us? <https://electronics.stackexchange.com/questions/274249/why-ultrasonic-module-sends-out-8-cycles-and-why-the-trigger-pulse-is-10us>, 04-05-2018. xi, 90

ESCARENO, J., SALAZAR-CRUZ, S. & LOZANO, R. (2006). Embedded control of a four-rotor uav. In *American Control Conference, 2006*, 6–pp, IEEE. 15

ESDS (2013). An introduction to gps - global positioning system. <https://www.esds.co.in/blog/an-introduction-to-gps-global-positioning-system/#sthash.pigtaSli.dpbs>, 25-04-2018. 42

- FACILITY, A.R.M.A.C.R. (2017). Arms southern great plains observatory hosts unmanned aerial systems. <https://www.arm.gov/news/features/post/45751>, 25-04-2018. 12
- FLECK, M. (2016). Usability of lightweight defibrillators for UAV delivery. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, 3056–3061, ACM. 11
- GAITANI, N., BURUD, I., THUIS, T. & SANTAMOURIS, M. (2017). High-resolution spectral mapping of urban thermal properties with unmanned aerial vehicles. *Building and Environment*, 121, 215–224. 11
- GAKSTATTER, E. (2015). What exactly is gps nmea data? <https://www.gps.gov/systems/gps/space/>, 25-04-2018. 44
- GPS.GOV (2018). Constellation arrangement. <https://www.gps.gov/systems/gps/space/>, 25-04-2018. x, 42
- GUPTA, S.G., GHONGE, M.M. & JAWANDHIYA, P. (2013). Review of unmanned aircraft system (uas). *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, 2, pp-1646. 1, 7
- HAGIOPOL, A. (2016). Autonomous outdoor navigation live demo. <https://alexhagiopol.com/portfolio/arduino-quadcopter-2-0-gps-live-demo/>, 04-05-2018. xiii, 135
- HASSANALIAN, M. & ABDELKEFI, A. (2017). Design, manufacturing, and flight testing of a fixed wing micro air vehicle with zimmerman planform. *Meccanica*, 52, 1265–1282. 7
- HASSANALIAN, M., KHAKI, H. & KHOSRAVI, M. (2015). A new method for design of fixed wing micro air vehicle. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 229, 837–850. 7
- HE, R., PRENTICE, S. & ROY, N. (2008). Planning in information space for a quadrotor helicopter in a gps-denied environment. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, 1814–1820, IEEE. 15

- HELIPAL (2018). Tarot 650 sport quadcopter build kit. <http://www.helipal.com/tarot-650-sport-quadcopter-frame-set.html>, 25-04-2018. 96
- HIGUERA, A.C. & CUETO, O.R.G. (2014). Identificación de la isla urbana de calor en mexicali, bc mediante imágenes de satélite y el uso de sistemas de información geográfica. *Información tecnológica*. 13
- HOBBY WING.COM (2018). Tarot 650 sport quadcopter tl65s01 super combo. <https://www.hobby-wing.com/tarot-650-sport-quadcopter-tl65s01-super-combo.html>, 26-04-2018. 96
- HOBBYKING.COM (2018). Turnigy multistar 4822-690kv 22pole multi-rotor outrunner. https://hobbyking.com/es_es/turnigy-multistar-4822-690kv-22pole-multi-rotor-outrunner.html, 01-05-2018. xii, 108
- HOBBYSTAR (2018). 8000mah 11.1v, 3s 100c hardcase. <https://hobbystarlabs.com/8000mah-11-1v-3s-100c-hardcase.html#.WukkvogvzIU>, 01-05-2018. xiv, 97
- JACOB JAMEY, C.P. (2016). Annual report NSF EPSCoR RII Track II FEC. <http://www.cloud-map.org/sites/default/files/2016%20Annual%20Report.pdf>, 25-04-2018. 12, 13
- JEFFBLACK (2014). New quad build advice? 450mm-500mm. <https://www.rcgroups.com/forums/showthread.php?2307799-New-quad-build-advice-450mm-500mm>, 25-04-2018. 96
- JIMB0 (2018). Serial communication. <https://learn.sparkfun.com/tutorials/serial-communication>, 25-04-2018. x, 53, 55, 57, 58, 59, 60, 61
- KALMAN, R.E. (1960). A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82, 35–45. 34

- KELEMEN, M., VIRGALA, I., KELEMENOV, T., MIKOV, U., FRANKOVSKY, P., LIPTK, T. & LRINC, M. (2015). Distance measurement via using of ultrasonic sensor. [xi](#), [90](#)
- KELLY, M. (2014). A tutorial on using ecalc. <https://www.rcgroups.com/forums/showthread.php?2101447-A-tutorial-on-using-eCalc>, 30-04-2018. [xii](#), [105](#), [106](#), [107](#)
- KRIJNEN, D. & DEKKER, C. (2014). Ar drone 2.0 with subsumption architecture. In *Artificial intelligence research seminar*. [1](#), [8](#)
- LADY-ADA (2017). Micro sd card breakout board. <https://learn.adafruit.com/adafruit-micro-sd-breakout-board-card-tutorial/introduction>, 25-04-2018. [xi](#), [xii](#), [88](#), [89](#), [110](#), [111](#)
- LEARNINGRC (2017). Drone (quadcopter) frame sizes – mini, micro, nano. <http://learningrc.com/drone-quadcopter-frame-sizes/>, 01-05-2018. [xi](#), [94](#)
- LIANG, O. (2013). Fc firmware for mini quad. <https://oscarliang.com/mini-quad-fc-firmware/>, 25-04-2018. [81](#)
- LUPÓN, M., TORRENTS, A. & QUEVEDO, L. (2012). Procesos cognitivos básicos. *Apuntes de psicología en atención visual*. [153](#)
- LUUKKONEN, T. (2011). Modelling and control of quadcopter. *Independent research project in applied mathematics, Espoo*, 22. [ix](#), [13](#), [15](#), [16](#)
- MADANI, T. & BENALLEGUE, A. (2006). Backstepping control for a quadrotor helicopter. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, 3255–3260, IEEE. [15](#)
- MARTIN, P. & SALAÜN, E. (2010). The true role of accelerometer feedback in quadrotor control. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 1623–1629, IEEE. [15](#)

- MASSACHUSETTS INSTITUTE OF TECHNOLOGY (2018). Mit app inventor - anyone can build apps that impact the world. <http://explore.appinventor.mit.edu/about-us>, 03-05-2018. 126
- MAXBOTIX INC. (2018). Mit app inventor - anyone can build apps that impact the world. https://www.maxbotix.com/documents/XL-MaxSonar-EZ_Datasheet.pdf, 03-05-2018. 112
- MAYBECK, P.S. (1982). *Stochastic models, estimation, and control*, vol. 3. Academic press. 34, 35
- MIKEGRUSIN (2018). Serial peripheral interface (spi). <https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi>, 25-04-2018. x, xi, 62, 63, 65, 66, 67
- MILEHIGHRC.COM (2018). User manual of platinum pro series brushless speed controller. http://milehighrc.com/Manuals/HW-120-PL_Pro.pdf, 01-05-2018. 129
- MILLER, P.M. (2006). Mini, micro, and swarming unmanned aerial vehicles: A baseline study. In *Mini, micro, and swarming unmanned aerial vehicles: A baseline study*, LIBRARY OF CONGRESS WASHINGTON DC FEDERAL RESEARCH DIV. 9, 10
- MODELTRONIC (2018). Batería lipo vok power 1.500mah 7,4v 25c conector t-dean. <http://www.modeltronic.es/bateria-lipo-vok-power-1500-mah-74v-25c-conector-t-dean-p-20462.html>, 25-04-2018. xi, 73
- MOORE, P. & CROSSLEY, P. (1999). Gps applications in power systems. part 1: Introduction to gps. *Power Engineering Journal*, 13, 33–39. 41, 42
- MUELLER, M. (2017). ecalc tutorial. <https://www.ecalc.ch/calcinclud/help/xcoptercalcutorial.htm>, 25-04-2018. xi, xii, 78, 101, 102, 103, 104

- MUELLER, M. (2018). xcoptercalc - multicopter calculator. <https://www.ecalc.ch/xcoptercalc.php>, 27-04-2018. 100
- MULGAONKAR, Y., WHITZER, M., MORGAN, B., KRONINGER, C.M., HARRINGTON, A.M. & KUMAR, V. (2014). Power and weight considerations in small, agile quadrotors. In *Micro-and Nanotechnology Sensors, Systems, and Applications VI*, vol. 9083, 90831Q, International Society for Optics and Photonics. 71
- MURPHY, R.R., STEIMLE, E., GRIFFIN, C., CULLINS, C., HALL, M. & PRATT, K. (2008). Cooperative use of unmanned sea surface and micro aerial vehicles at hurricane Wilma. *Journal of Field Robotics*, 25, 164–180. 10
- NEMONIC (2015). Adafruit forum: Mpl3115a2 not working. <https://forums.adafruit.com/viewtopic.php?f=19&t=73118&p=418612#p418323>, 25-04-2018. 69
- NYQUIST, J.E. (1997). Unmanned aerial vehicles that even geoscience departments can afford. *Geotimes*, 42, 20–23. 2
- OKE, T. (1995). The heat island of the urban boundary layer: characteristics, causes and effects. In *Wind climate in cities*, 81–107, Springer. 12
- OKLAHOMAN, T. (2015). University researches to develop weather-forecasting drone system. TheOklahoman.com, 25-04-2018. 11
- PARALLAX INC. (2018). Drive a servo with the rc receiver - a beginner hardware hack. <https://learn.parallax.com/tutorials/robot/elev-8/add-servos-elev-8-v3/drive-servo-rc-receiver>, 04-05-2018. xiii, 136
- PARROT.SA. (2017). Parrot disco fpv. <https://www.parrot.com/us/drones/parrot-disco-fpv#-parrot-disco-fpv>, 25-04-2018. ix, 8
- PAWELSKY (2013). Dji naza gps communication protocol - nazadecoder arduino library. <https://www.rcgroups.com/forums/member.php?u=393936>, 04-05-2018. 134

- PENG, K., CAI, G., CHEN, B.M., DONG, M., LUM, K.Y. & LEE, T.H. (2009). Design and implementation of an autonomous flight control law for a uav helicopter. *Automatica*, **45**, 2333–2338. 7
- PEREZ, T. (2006). *Ship motion control: course keeping and roll stabilisation using rudder and fins*. Springer Science & Business Media. 23, 25
- PI, CHARLY (2017). ¿cuantos celulares hay en el mundo? <https://blog.ebaysocial.mx/2017/04/03/mofu-cuantos-celulares-hay-en-el-mundo/>, 03-05-2018. 125
- PI, CHARLY (2018). Operating system market share worldwide. <http://gs.statcounter.com/os-market-share#monthly-201704-201804>, 03-05-2018. xii, 125
- POKRESS, S.C. & VEIGA, J.J.D. (2013). MIT app inventor: Enabling personal mobile computing. *CoRR*, **abs/1310.2830**. 126
- POSITIVE-IMPORTS (2014). Introduction to lipo batteries. <https://www.icharger.co.nz/buying/resources-faq/introduction-to-lipo-batteries/>, 25-04-2018. 72, 73, 74
- PSIROFONIA, P., SAMARITAKIS, V., ELIOPOULOS, P. & POTAMITIS, I. (2017). Use of unmanned aerial vehicles for agricultural applications with emphasis on crop protection: three novel case-studies. *International Journal of Agricultural Science and Technology*. 10
- QUILTER, M.C. & ANDERSON, V.J. (2001). A proposed method for determining shrub utilization using (la/lis) imagery. *Journal of Range Management*, 378–381. 2
- QWINOUT (2017). Qwinout tarot 650 4-axis folding aircraft rtf kit. <https://www.amazon.com/QWinOut-Tarot-4-Axis-Folding-Aircraft/dp/B00WG6M4HS>, 25-04-2018. xi, xiv, 95
- RAFFO, G.V., ORTEGA, M.G. & RUBIO, F.R. (2010). An integral predictive/nonlinear H_{∞} control structure for a quadrotor helicopter. *Automatica*, **46**, 29–39. 15

- RASPBERRY.ORG (2018). Raspberry pi hardware guide. <https://www.raspberrypi.org/learning/hardware-guide/>, 25-04-2018. xii, 119
- RCGROUPS (2017). Ris my built for tarot ironman 650 okay? <https://www.rcgroups.com/forums/showthread.php?2959413-Is-My-Built-For-Tarot-Ironman-650-okay>, 26-04-2018. 96
- SAM (2014). A quick guide to brushless motors. <https://www.dronetrest.com/t/brushless-motors-how-they-work-and-what-the-numbers-mean/564>, 25-04-2018. xi, 76
- SAMARITAKIS, V. (2015). Tarot 650 – pixhawk – sunnysky x4108s 480kv – 15x5.5 props – quadcopter setup. <https://blog.samaritakis.gr/en/tarot-650-quadcopter-setup/>, 26-04-2018. 96
- SCHMIDT, M.D. (2011). Simulation and control of a quadrotor unmanned aerial vehicle. *University of Kentucky UKnowledge*. 71
- SEMICONDUCTORS, P. (2004). An10307 - uart to bluetooth interfacing. <https://www.nxp.com/docs/en/application-note/AN10307.pdf>, 25-04-2018. 86
- SFUPTOWNMAKER (2018). Serial peripheral interface (spi). <https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi>, 25-04-2018. xi, 67, 68
- SHENZHEN SKYHAWKRC TECHNOLOGY Co., L. (2018). Octacoptero skyhawk (rotary wing). <http://www.skyhawkrc.com/productshow.asp?id=19>, 25-04-2018. ix, 14
- SIMON, D. (2001). Kalman filtering. *Embedded systems programming*, 14, 72–79. 41
- SITE ADMIN, ADAFRUIT (2018). MPL3115A2 Not Working. <https://forums.adafruit.com/viewtopic.php?f=19&t=73118&start=30>, 07-05-2018. 155

- SMITH, S.W. (2017). Additional data via autonomous systems to supplement traditional sparse sources for weather forecasting and atmospheric science. *Collaboration Leading Operational Uas Development for Meteorology and Atmospheric Physics*. ix, 13, 14
- TEDRAKE, R. (2009). Underactuated robotics: Learning, planning, and control for efficient and agile machines: Course notes for mit 6.832. *Working draft edition*, 3. 19
- THIIS, T.K., GAITANI, N., BURUD, I. & ENGAN, J.A. (2018). Classification of urban blue green structures with aerial measurements. *International Journal of Sustainable Development and Planning*, 13, 506–515. 11
- TIMES, P.D. (2015). Israel sell Heron UAVs to India and Australia. <https://www.strategypage.com/militaryforums/512-20022.aspx#startofcomments>, 25-04-2018. 10
- TURBOSQUID2018 (2013). U.s. air force predator. <https://www.turbosquid.com/3d-models/predator-drone-3d-max/706959>, 25-04-2018. ix, 8
- TUTORIALES (2016). Tutorial básico nrf24l01 con arduino. http://www.naylampmechatronics.com/blog/16_Tutorial-b%C3%A1sico-NRF24L01-con-Arduino.html, 25-04-2018. xi, 85
- UBAYA, H. & IQBAL, M. (2015). First person view on flying robot for real time monitoring. *ICON-CSE*, 1, 41–44. 8
- UBLOX (2017). Neo-m8 u-blox m8 concurrent gnss modules data sheet. https://www.u-blox.com/sites/default/files/NEO-M8-FW3_DataSheet_%28UBX-15031086%29.pdf, 04-05-2018. 134
- UNMANNEDTECH (2012). Ut-q650 quadcopter assembly guide. <http://www.unmannedtech.co.uk/manuals/ut-q650-assembly-guide#comments>, 26-04-2018. xi, xiii, 94, 158, 161, 162, 163, 164
- US-AIR-FORCE (2014). Rq-4 global hawk. <http://www.af.mil/About-Us/Fact-Sheets/Display/Article/104516/rq-4-global-hawk/>, 25-04-2018. ix, 9, 14

- VAN BLYENBURGH, P. (1999). Uavs: an overview. *Air & Space Europe*, **1**, 43–47. 7
- VENESS, C. (2017). Calculate distance, bearing and more between latitude/-longitude points. <https://www.movable-type.co.uk/scripts/latlong.html>, 05-05-2018. 142
- WARD, P. (2017). Camera in the sky: Using drones in wedding photography and videos. <https://www.bhphotovideo.com/explora/video/tips-and-solutions/camera-sky-using-drones-wedding-photography-and-videos>, 25-04-2018. ix, 15
- WATTS, A.C., AMBROSIA, V.G. & HINKLEY, E.A. (2012). Unmanned aircraft systems in remote sensing and scientific research: Classification and considerations of use. *Remote Sensing*, **4**, 1671–1692. 1, 2, 7
- ZEMALACHE, K.M., BEJI, L. & MARREF, H. (2005). Control of an under-actuated system: application a four rotors rotorcraft. In *Robotics and Biomimetics (RO-BIO)*. 2005 IEEE International Conference on, 404–409, IEEE. 15
- ZHOU, W., QIAN, Y., LI, X., LI, W. & HAN, L. (2014). Relationships between land cover and the surface urban heat island: seasonal variability and effects of spatial and thematic resolution of land cover data on predicting land surface temperatures. *Landscape ecology*, **29**, 153–167. 11
- ZUO, Z. (2010). Trajectory tracking control design with command-filtered compensation for a quadrotor. *IET control theory & applications*, **4**, 2343–2355. 15

浪
川