PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO

Escuela de Ingeniería Eléctrica
PONTIFICIA UNIVERSIDAD CATOLICA DE VALPARAISO

Miguel Angel Arellano Cortés

Implementación de radar para ayuda de navegación autónoma en un drone

Informe Proyecto de Título de Ingeniero Electrónico

Escuela de Ingeniería Eléctrica

Implementación de radar para ayuda de navegación autónoma de un drone

Miguel Angel Arellano Cortés

Informe Final para optar al título de Ingeniero Electrónico,

aprobada por la comisión de la

Escuela de Ingeniería Eléctrica de la

Pontificia Universidad Católica de Valparaíso

conformada por

Sr. Francisco Pizarro Torres

Profesor Guía

Sr. Gabriel Hermosilla Vigneau

Segundo Revisor

Sr. Sebastián Fingerhuth Massmann

Secretario Académico

Valparaíso, 25 de julio de 2017

Agradecimientos

Me gustaría agradecer primeramente a Dios, por la gran oportunidad de poder estudiar en una universidad y por darme las fuerzas para siempre continuar. Además a quienes me han apoyado desde que tengo memoria, que siempre me han dicho que yo puedo que desde pequeño me amaron y me cuidaron para ahora poder dar un paso enorme en mi vida y poder decir acá estoy, ya a puertas de terminar este gran desafío, a mis padres, estoy 100% seguro que no estaría donde estoy ahora si no fuera por ellos. Mamá, gracias por siempre querer ayudarme de alguna u otra forma, por tratar de trasnochar conmigo en mis noches de estudio, por entenderme en los momentos más estresantes y por siempre tener tus brazos abiertos cuando los necesité o los pedí, como te lo he dicho un par de veces, estoy más que seguro, que no hay mejor persona para tenerme y criarme que tú mama. Papá gracias por estar ahí también, darme tu apoyo y mostrarme el camino de la responsabilidad, honradez y transparencia, has inculcado grandes valores en mi vida que atesoro en mi corazón y que jamás perderé.

Y terminando por grandes amistades que han dicho: "Tú puedes" cuando ni yo mismo creía que lograría, mi amiga Grecia, que me acompañaste en todo este proceso de la tesis y me dabas palabras de aliento y alimento, eres la mejor amiga de la vida. La "osa Pauli" que me acompañó en casi todos los trasnoches que tuve que hacer y me alentaba. Y finalizando con aquellos compañeros que pasaron a ser amigos, Mauricio Rojas gracias por venir tolerándome desde hace años con mi carácter estresado y llorar juntos después de las pruebas, Andrés Vielma por siempre tener las puertas de su casa abierta para mí cuando lo necesité. Por último a los tíos de la universidad que siempre se preocupaban por mí y me daban palabras de aliento. De verdad a todos, ¡muchas gracias!

Resumen

En el trabajo se presenta una solución para evitar la colisión de un drone en su ruta de vuelo, utilizando el radar Doppler de onda continua de la empresa RFBeam modelo RSP1.

Primeramente se investigó a fondo sobre los aparatos que debieron ser utilizados para comprender su funcionamiento, sus propiedades, como trabajan y ver si es factible la unión entre el radar y el drone.

Para realizar lo anterior fue requerido un arduino Uno con el fin de procesar la señal del radar y comunicarlo con el sistema de vuelo automático HKPilot32 para que este, avise sobre el peligro de alguna colisión. En vista a lo anterior se debió resolver si es posible la comunicación para luego investigar como llevarla a cabo la acción. Para ello primero se averiguo si ambos elementos utilizan el mismo protocolo de comunicación, se descubrió que utilizan el protocolo de comunicación serial UART concluyendo que la interacción entre el radar con el arduino es posible. El siguiente punto es como llevar a cabo dicha comunicación, encontrando dos aristas principales la primera fue la conexión física de ambos elementos descubriendo que trabajan con tecnologías diferentes (TTL y CMOS) resolviendo dicho problema con un puente de resistores. La segunda arista es el código que debe poseer el arduino para leer los datos enviados por el puerto serial del radar y poder manipularlos. Para el caso del arduino al drone se siguió la misma lógica anterior comunicando a estos dispositivos mediante el protocolo UART y mediante un código en C se estableció la lectura e interpretación de los datos.

Lo siguiente fue determinar la forma de control que se tendría en el vuelo del drone, el resultado final fue la notificación al usuario del HKPilot32 cuando hay un obstáculo en su ruta de vuelo durante un tiempo determinado por el usuario en el código del Arduino.

Los resultados obtenidos fueron: el entendimiento de los elementos usados en el proyecto, estudiar, analizar e implementar los protocolos de comunicación de dichos elementos, la comunicación, el procesamiento y la manipulación de datos entre ellos. Y finalmente como objetivo final avisar en caso de que el drone esté frente a una posible colisión.

Abstract

A solution is presented to avoid the possible collision of a drone in its flight path, using the continuous wave Doppler radar of the RFBeam model RSP1.

First, was investigated thoroughly the devices that should be used to understand their operation, their properties, how they work and see if feasible the union between the radar and the drone.

To do this, an arduino One was required to process the radar signal and communicate it with the HKPilot32 automatic flight system to warn of the danger of a collision. In view of the above it should be resolved if possible the communication and then investigate how to carry out the action. To do this, it was first determined whether both elements use the same communication protocol, it was discovered that it uses the serial communication protocol UART concluding that the interaction between the radar and the arduino is possible. The next point is how to carry out such communication, finding two main edges, the first was the physical connection of both elements discovering that they work with different technologies (TTL and CMOS) solving that problem with a bridge of resistors. The second edge is the code that the arduino must have to read the data sent by the serial port of the radar and to be able to manipulate them.For the case of the arduino to the drone the same previous logic was followed communicating to these devices by means of the protocol UART and by means of code of the reading and interpretation of the data was established.

The next thing was to determine the form of control that would be in the flight of the drone, the final result was the notification to the user of the HKPilot32 when there is an obstacle in its route of flight during the course of a time determined by the user in the Arduino code.

The results obtained were: the understanding of the elements used in the project, studying, analyzing and implementing the communication protocols of such elements, communication, processing and manipulation of data between them. And finally as the final goal to warn in case the drone is facing a possible collision.

Índice general

Introducción

Desde el comienzo del desarrollo tecnológico que data desde la segunda guerra mundial [1], la humanidad ha experimentado un gran cambio en la forma de llevar su vida cotidiana, acostumbrándose al uso de ayudas provenientes de aparatos capaces de facilitarnos la toma de decisiones, la organización de los quehaceres, el diario vivir y facilitando el contacto a larga distancia con las personas, llegando a ser la tecnología algo primordial en la vida de las personas [2].

Dentro de este gran avance, existe un invento tecnológico en particular que en los últimos años ha ido repercutiendo en muchos sectores, tales como fuerzas armadas, compra y venta, entretención, entre otros [3]. Aquel invento es conocido como los vehículos aéreos no tripulados.

Dicho invento tuvo sus comienzos en los modelos construidos y volados por inventores como Cayley, Stringfellow, Du Temple y otros pioneros de la aviación, que fueron previos a sus propios intentos de desarrollar aeronaves tripuladas a lo largo de la primera mitad del siglo XIX [4].

El término vehículo aéreo no tripulado (Unmanned Aerial Vehicle, UAV) se hizo común en los años 90 para describir a las aeronaves robóticas [4]. A grandes rasgos, es una aeronave que vuela sin tripulación humana a bordo. Su utilización se basa mayoritariamente en aplicaciones militares. Los UAV también son utilizados en un pequeño pero creciente número de aplicaciones civiles, como en labores de lucha contra incendios o seguridad civil, como la vigilancia de los oleoductos [5].

Históricamente los UAV eran simplemente aviones pilotados remotamente, pero cada vez más se está empleando el control autónomo de ellos. En este sentido se han creado dos variantes: algunos son controlados desde una ubicación remota y otros vuelan de forma autónoma sobre la base de planes de vuelo pre-programados usando sistemas más complejos de automatización dinámica.

Debido al gran requerimiento de los usuarios y llevando consigo el alza en los vehículos aéreos no tripulado y por lo tanto un mayor avance en sus tecnologías, es que hoy en día se pueden manipular estos aparatos a distancias de 120 metros de altura y 500 metros [6] dentro del radio de alcance visual de su piloto.

Si bien en la actualidad existen vehículos aéreos no tripulados que su trayectoria puede ser es fijada mediante un GPS, existe la posibilidad de una colisión con un objeto que el navegante no se haya percatado.

Actualmente la empresa DJI Phantom dentro de sus últimos avances, desarrolló un Drone con la capacidad de la evasión de obstáculos en base a sonares y cámaras. Sin embargo, en lugares con poca visibilidad estos pueden dejar de ser confiables [6]. Una solución para lo dicho anteriormente es aplicando métodos mediante radio frecuencias, los que no dependen directamente de la visual del aparato y utilizan radares para su funcionamiento.

Radar es un término que procede de un acrónimo inglés: RAdio Detecting And Ranging ("Detección y localización por radio") [7]. Para entender bien la génesis tecnológica completa de este equipo es necesario remontarse al año 1864, en donde el físico inglés James Maxwell desarrolló las ecuaciones que gobiernan el comportamiento de las ondas electromagnéticas. Posterior a esto, en 1886, el físico alemán Heinrich Herz pudo demostrar a partir de las ecuaciones de Maxwell, las leyes de reflexión de las "ondas de radio". Con esto logró demostrar que hay ciertas propiedades físicas de los medios que facilitan la reflexión de estas ondas [8].

Para ejecutar las mediciones descritas, Herz diseñó un elemento que generaba (transmisor) una onda a partir de la descarga de un "capacitor" sobre una bujía generando un arco. Esta energía la canalizaba a través de un "loop de alambre" (antena) y a través de otro "loop" lograba medir la cantidad de energía traspasada. Sin embargo, Herz no pudo ver el fruto de su trabajo ya que el año 1894 falleció [8].

En 1900 los estadounidenses lograron ejecutar su primera transmisión de voz por un medio sin hilos, logrando el interés de muchos oficiales jóvenes y científicos por esta nueva forma de comunicación por el espacio libre, incorporando a los EE.UU. de América en el desarrollo de estas tecnologías [8].

En 1903 el investigador alemán Christian Hulsmeyer fue capaz de detectar ondas de radio que reflejaba en los buques, creando un radar de corto alcance de tan solo una milla [8].

Pasaron 20 años, para que en 1922 el genio italiano de Marconi, retomando los estudios teóricos hiciera notar que era posible que las ondas de radio, focalizadas en un haz pudieran ser reflejadas por un objeto como un buque y así obtener su presencia, distancia y demarcación en especial durante la noche, con neblina o en malas condiciones de tiempo [8].

Al hablar de ondas de radio en esta época se hace referencia a señales de hasta 30 (MHz), las que ahora sólo se utilizan en la frecuencia baja del espectro como ondas cortas de HF en el área de comunicaciones [8].

En 1935 Robert Watson -Watt logró detectar un avión a 15 millas. En el mismo año consiguió detectar un bombardero a 40 millas con ondas de 12 (MHz). Se le llamó RDF: Radio RDF: Radio Detection Detection Finding [9].

A continuación en 1936 Page y Young desarrollan el primer radar pulsado llegando a tener un alcance de 25 Millas. Y así sucesivamente el crecimiento de estas tecnologías fue en aumento hasta los que conocemos en la actualidad [9].

Dentro de la gama de radares existen uno en particular que utiliza el efecto Doppler, dicho efecto ocurre cuando el receptor de la onda se mueve con respecto al emisor, o viceversa. [9] Si el emisor se está moviendo, significa que cada nueva oscilación parte desde una posición ligeramente diferente. A consecuencia de esto, la distancia entre cada cresta de la onda será diferente [10]. Cuando estas ondas más juntas llegan a un receptor, le parece que la frecuencia es mayor. En cambio, las ondas emitidas en el sentido contrario de la marcha sufren el fenómeno contrario: si el emisor se aleja del receptor, la frecuencia recibida es menor. Este fenómeno ocurre también con las ondas sonoras. Es muy fácil notarlo en los coches de competición, el sonido es muy agudo cuando se acercan y de golpe se convierte en grave cuando pasan por delante y empiezan a alejarse [11].

Dicho de otra forma, la solución para evadir la dificultad de depender de la visibilidad que puedan tener las cámaras implementadas al Drone como lo plantea DJI Phantom, es de implementar un radar Doppler para la guía del vehículo aéreo no tripulado.

Objetivo general

- Implementar un radar para ayuda de navegación en un DRONE.

Objetivos específicos

- Estudiar el estado del arte de radares y su implementación en vehículos no tripulados.
- Estudiar e implementar la interface de antena y tratamiento de señal que será utilizada en el DRONE.
- Estudiar e implementar el protocolo de comunicación, lectura e interpretación entre la interfaz y el DRONE.
- Determinar e implementar el sistema de navegación autónoma con la ayuda de un RADAR.

1 Presentación del proyecto

Debido al problema con la colisión de un drone, algunas empresas o estudiantes han desarrollado opciones para poder solucionar dicha dificultad en estos vehículos aéreos no tripulados, las que se verán en el presente capítulo.

1.1 Soluciones propuestas

Las soluciones propuestas por empresas o estudiantes son las presentadas a continuación.

1.1.1 DA-Jian Innovations Science and Techonology

La empresa China conocida como DJI o DA-Jian Innovations Science and Techonology, líder en la creación de vehículos aéreos no tripulados, revolucionó el mercado de los drones con su último producto siguiendo la generación de los "Phantom". Esta empresa arrojó a la venta el Phantom 4 con la capacidad de evadir obstáculos u objetos que se le presenten en su ruta a seguir, dándole casi la total autonomía en vuelo a su producto [12].

Su última creación lleva consigo dos sensores de distancia por delante y cuatro por debajo de detección por sonar y además posee reconocimiento por nube de puntos estereoscópica [6], dicho de otra manera, representa el camino que sigue el vehículo mediante un conjunto de muestras en coordenadas (x,y,z) [13] tomadas por las cámaras del Drone y mediante el software Obstacule sensing System. Este fue diseñado por los mismos científicos de la empresa [14] y lograron hacer que el Drone reconociera 15.24 (m) por delante suyo y aproximadamente 10 (m) por debajo del vehículo, tomando la decisión entre bordear el obstáculo o detenerse. Además avisa la proximidad de un objeto por medio de indicadores visuales y sonoros para que el usuario tenga conocimiento sobre lo que está ocurriendo [6]. En la Figura 1-1 se muestra un ejemplo de como el Drone toma muestra de su camino.

Figura 1-1 Toma muestra de su camino (fuente: http://www.hstore.cl).

1.1.2 Pushbroom stereo

Andrew Barry, estudiante del Instituto Tecnológico de Massachusetts (MIT), creó en su trabajo de tesis un algoritmo el cual permite que un dron detecte obstáculos en tiempo real y sin necesidad de ninguna tecnología remota, incluso si vuela a velocidades cercanas a los 50 (km/h).

Hasta la fecha, los algoritmos de estos vehículos ocupan mucho tiempo de computación y no permiten, en la práctica, que los drones superen los ocho kilómetros por hora sin un hardware de procesamiento de datos especializado [15].

El creador utiliza un sistema estándar de bloques de coincidencia, el que produce estimaciones de profundidad encontrando similitudes entre dos imágenes. Dado un bloque de pixel en la imagen izquierda, por ejemplo, el sistema buscará un objeto mediante la línea epipolar (ejemplificada en la Figura 1-2) y ayudará a encontrar la mejor coincidencia. La posición de la similitud en relación a su coordenada en las imágenes por la cámara izquierda, permite al computador calcular la posición 3D del objeto en ese bloque pixel [3].

En la Figura 1-3 se muestra una tabla explicativa del sistema estándar de bloques de coincidencias.

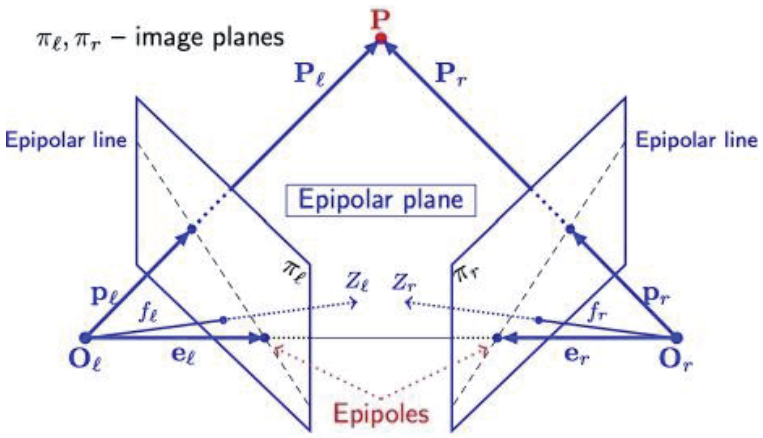


Figura 1-2 Línea Epipolar y Distancia Estéreo Base (fuente: www.cs.auckland.ac.n).

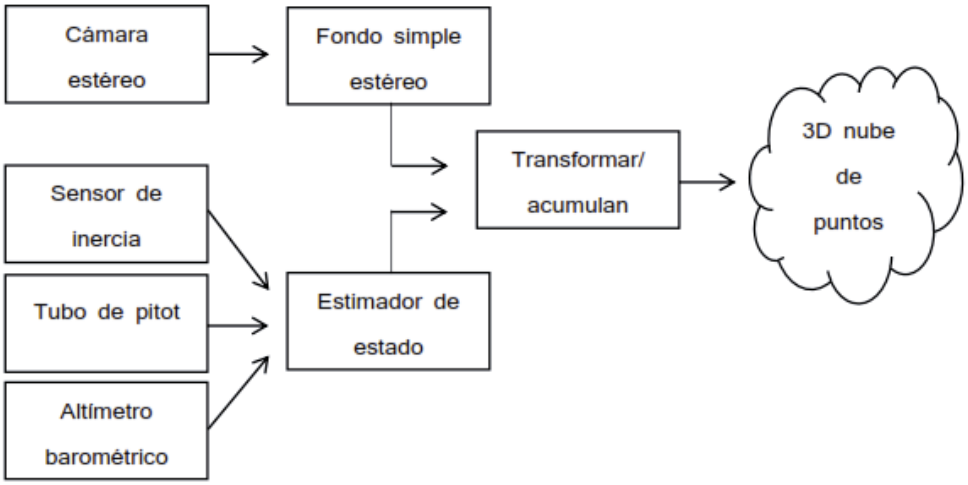


Figura 1-3 Tabla explicativa del sistema estándar de bloques de coincidencia (fuente: http://groups.csail.mit.edu).

En otras palabras, lo que este sistema hace es buscar a lo largo de la línea epipolar un grupo de pixeles que coincidan con el bloque candidato a ser analizado y con ello explorar el territorio frontal del vehículo mientras este está en vuelo desde las cámaras. Por ejemplo, dado un bloque de pixel en la imagen izquierda, se comienza a buscar mediante la imagen derecha una gran disparidad, correspondiendo a un objeto que se va acercando a las cámaras. A medida que disminuye esta disimilitud (cambiando de lugar la imagen derecha que estamos comparando), se examinan los bloques de pixeles que corresponden a objetos más lejanos hasta llegar a cero, donde la distancia estéreo base (base distance en inglés, además mostrada en la Figura 1-2, como la distancia entre los epipolos) es insignificante comparado a la distancia del bloque de pixel. Con esta última consideración se puede determinar la ubicación del objeto [3].

Teniendo en cuenta los puntos anteriores, es fácil ver que si se limita la distancia de búsqueda a una distancia "d" (en metros) fija, se puede aumentar la velocidad de procesamiento con el costo de descuidar objetos a distancias distintas a "d". Si bien, esto puede parecer limitado, pero las

cámaras al estar en constante movimiento (por estar integradas en el vehículo), como lo muestra la Figura 1-4, pueden recuperar rápidamente la información de profundidad mediante la integración de la estimación de la posición del vehículo (conocido también como odometría) y los resultados de disparidad. En otras palabras, se limita la habilidad para tomar el mejor bloque de pixeles, siendo reemplazado por un umbral de posibles coincidencias. En la Figura 1-5 muestra como toma los datos [3].

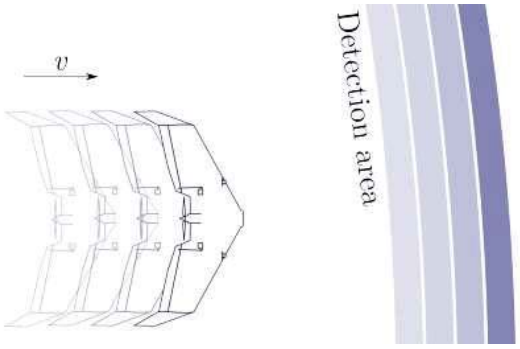


Figura 1-4 distancia "d" de las cámaras a la zona de detección (fuente: http://groups.csail.mit.edu).

Una vez que la plataforma ha cubierto "d" metros, nuevos obstáculos serán inmediatamente identificados a la máxima distancia posible (con la exclusión de obstáculos en movimiento). En caso de que la plataforma gire bruscamente o el vehículo se quede estático, la información se desecha [3].

Figura 1-5 Detección de obstáculos (fuente: http://groups.csail.mit.edu).

1.1.3 Máquina de aprendizaje con percepción visual

Considerando una pista forestal, la entrada es una imagen tomada por una cámara situada por encima de la pista. Los autores de estas máquinas de aprendizaje, trabajaron a la altura media de una persona, ya que a dicha altura se provee una buena vista de la tierra circundante y además se espera que esté libre de todo obstáculo del sendero, tales como rocas, algunos animales pequeños, entre otros [16].

Para iniciar el trabajo de estas máquinas de aprendizaje utilizaron un vector velocidad como la dirección óptica de la cámara en el plano horizontal. Por otro lado se elige al vector "t" como la

dirección del sendero, y "v" la dirección que el excursionista deberá tener al caminar, otorgándole la ruta que seguirá el robot en el futuro con el objetivo de permanecer en la pista como lo muestra la Figura 1-6.

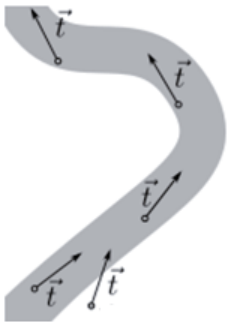


Figura 1-6 Ruta a seguir por el excursionista sin salir de la ruta [16].

Otra variable que se le otorga al plano es el ángulo alfa, el que se asigna como el ángulo entre el vector "v" y "t". Existen 3 opciones que corresponden a diferentes acciones que el portador de la cámara debe implementar con el fin de permanecer en la pista, en el supuesto que la cámara esté mirando a la dirección del movimiento.

Acciones a seguir de acuerdo al ángulo:

- Girar a la izquierda (TL): si el ángulo alfa está entre -90° y menos beta (de valor 15° para todos los casos), significa que le sendero se dirige hacia la izquierda de la imagen.
- Continuar (GS): si el ángulo alfa está entre menos beta y beta grados, el sendero continúa en línea recta.
- Girar a la derecha (TR): si el ángulo alfa está entre menos beta grados y +90°, significa que el sendero se dirige hacia la derecha de la imagen.

Esto se explica en la Figura 1-7.

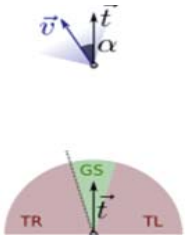


Figura 1-7 Explicación de toma de decisiones de acuerdo al ángulo alfa [16].

En caso de que el robot esté mirando en una dirección perpendicular con respecto al sendero, sólo se puede inferir que el vehículo no continuará por ese camino.

La percepción del sendero se resuelve mediante la recopilación de un conjunto de datos que sean representativos y que cubra una gran variedad del sendero y una larga distancia de éste. Para adquirir esta base de datos, se equipó a un excursionista con tres cámaras. La primera apuntaba

a treinta grados a la izquierda, la otra con dirección al frente y la tercera con un ángulo de treinta grados a la derecha, consiguiendo con esto una cobertura de aproximadamente ciento ochenta grados como muestra la Figura 1-8. Una vez que el excursionista termina de recorrer el camino, con el cuidado de mirar siempre recto a lo largo de su dirección, se crea el conjunto de datos, recopilando las imágenes de las tres cámaras, asociando cada imagen con una de las clases anteriormente mencionadas. Si el excursionista tuvo el cuidado de caminar recto en la ruta, las tomas adquiridas por la cámara central serán de clase "GS", las tomas por la cámara derecha serán de clase "TL" y la que queda serán de clase "TR" [16].

Figura 1-8 Como se toman las imágenes del sendero [16].

1.2 Cuadro comparativo

En esta sección se hará un resumen de los sistemas descritos con anterioridad enfatizando los pros y contras que poseen.

Tabla 1-1 Comparación entre sistemas.

Proyectos	Pro	Contras
DJI	Alta velocidad de reacción frente obstáculos.	Depende del campo visual de la cámara y utiliza más de un procesador.
Pushroom Stereo	Alta velocidad de viaje del vehículo.	El vehículo usado funciona como planeador, depende del campo visual de la cámara y el vehículo usado no puede estar estacionario mientras vuela.
Máquina de aprendizaje de la percepción visual de pistas forestales para robots móviles	Puede estar en zonas muy hostiles.	Un excursionista debe grabar el camino correcto previamente al recorrido del vehículo.

1.3 Solución propuesta

Según lo recolectado por la Tabla 1-1, se puede observar que todos los sistemas para evadir obstáculos y evitar colisiones dependen del campo visual tomado por las cámaras, siendo este escaso en ocasiones de menos visibilidad. Para solucionar esto, se propone el desarrollo de un

sistema con el fin de detectar obstáculos, pero teniendo en consideración los radares que dependen en menos medida del campo visual.

A continuación se presentan algunos ejemplos de aplicación de radares [17]:

- Vigilancia aérea: alarma temprana a larga distancia, adquisición para sistemas de alarmas, determinación de altitud en radares tridimensionales, vigilancia de aeropuertos y rutas aéreas.
- Vigilancia espacial y proyectiles dirigidos: alarma de proyectiles dirigidos, adquisición de proyectiles dirigidos, vigilancia de satélites artificiales.
- Radar meteorológico: observación y predicción de fenómenos naturales.
- Además se puede mencionar un gran número de aplicaciones no pertenecientes a categorías definidas, tales como alarma contra intrusos, monitorización de migración de aves, control de vehículos terrestres, entre otros.

Se puede observar los diversos usos que poseen los radares, sin embargo su clasificación va más allá de sólo sus aplicaciones, puesto que se dividen de acuerdo a lo que estos instrumentos miden y como hacen este proceso.

Podemos encontrar cuatro tipos de radares [18]:

- Radar de pulsos: radar ampliamente utilizado para la medición de distancias desde el inicio de la tecnología de radar. El principio básico de medición consiste en medir el "tiempo de vuelo", es decir, el tiempo que tarda la señal en "ir" y volver" al radar como lo muestra la Figura 1-9. La duración típica de este "tiempo de vuelo" es usualmente del orden de los milisegundos.

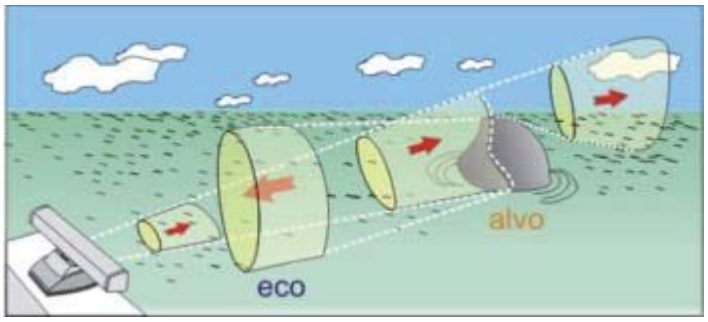


Figura 1-9 Radar de pulsos (fuente: www.escolanautica.com.br)

- Radar Doppler de pulsos: los pulsos transmitidos por un radar típico de pulsos pueden ser considerados como una pequeña ráfaga de ondas continúas como lo muestra la Figura 1-10. Este tipo de radar compara las frecuencias a las que detecta el objeto, dicho de otra manera, si la frecuencia de las ondas transmitidas se define como f_t y el objeto moviéndose hacia el radar con una velocidad v, la frecuencia de retorno del pulso estará definida como $f_t + f_{dp}$, donde f_{dp} es la frecuencia Doppler. Del mismo modo, si el objeto se

aleja del radar, la frecuencia recibida será $f_t - f_{dp}$. La ventaja de ese tipo de radar radica en el hecho de que "ignora" los objetos estacionarios.

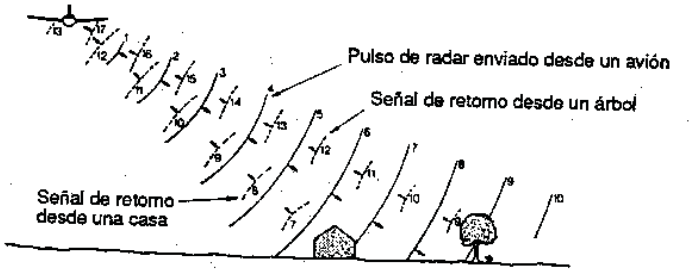


Figura 1-10 Radar Doppler de pulso (fuente: http://www.fao.org).

- Radar de onda continua: este tipo de radares envían un señal sin modulación y los ecos son recibidos desde el objetivo como lo muestra la Figura 1-11. Si el objetivo es estacionario, la frecuencia de retorno será la misma que la de la señal transmitida y la distancia del objetivo no podrá ser medida. Del mismo modo que el tipo de radar anterior, utiliza la frecuencia de retorno de un objeto en movimiento para que con el efecto Doppler logre medir la velocidad de acercamiento o alejamiento del objeto.

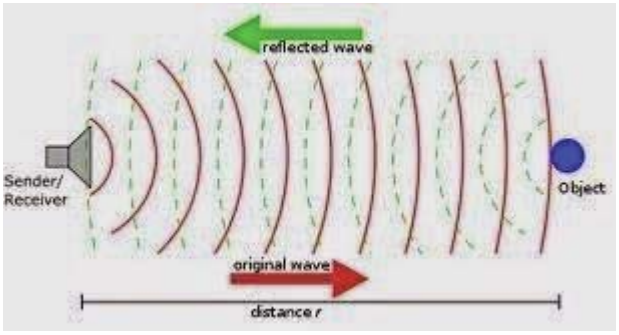


Figura 1-11 Radar de onda continua (fuente: http://golmuhendis.blogspot.cl).

- Radar de onda continua con frecuencia modulada: este tipo de radar funciona mediante la comparación de la frecuencia de la señal enviada y el corrimiento de la señal de retorno como lo muestra la Figura 1-12. Estas tienen una relación de tipo lineal a partir de la cual se puede determinar la distancia del objetivo.

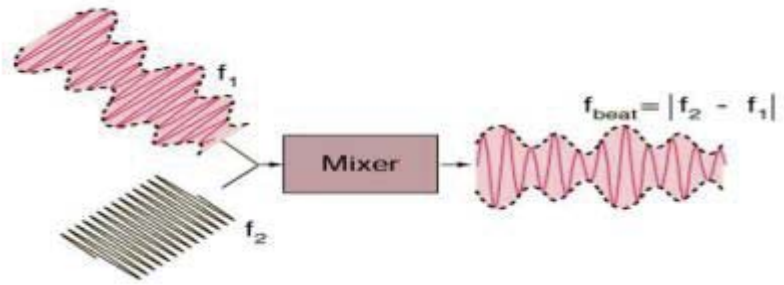


Figura 1-12 Radar de onda continua con frecuencia modulada [19].

Debido al gran alcance de aplicaciones que poseen los radares y su mayor independencia con respecto al campo visual que posee al momento de estar en vuelo (como se demostrará a continuación), se elegirá a este instrumento como medio para la percepción de obstáculos con el objetivo de disminuir riesgo de colisión en la ruta del vehículo aéreo no tripulado y finalmente dar aviso del obstáculo.

Según estudios entregados por la ITU (International Telecommunication Union), la atenuación de la propagación de ondas de un radar está expresada por:

$$\Gamma c = K_1 M \, [\frac{Db}{Km}] \tag{1-1}$$

Con:

- r_c: atenuación específica en la nube (dB/km)
- K_1: coeficiente de la atenuación específica ((dB/km)/(g/m^3))
- M: densidad de agua líquida en la nube o la niebla (g/m^3)

Con el fin de ejemplificar lo anterior, se eligió la densidad de la niebla con el valor de 2.34 (gr/m^3), que implica una visibilidad de unos 30 (m), entregando una atenuación específica para una frecuencia de 25 (GHz) de 0.5 (db/Km). Valores obtenidos por la línea punteada amarilla en la Figura 1-13 [20].

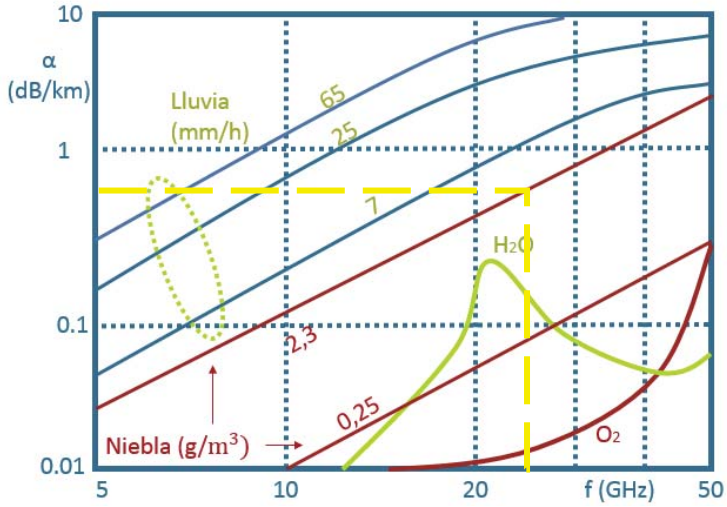


Figura 1-13 Atenuación específica por neblina [21].

Debido a este resultado, se puede concluir que la atenuación total en una neblina espesa es de aproximadamente 1.12 (W/Km) siendo muy baja respecto a los 30 (m) de visibilidad que tendría la cámara. Si bien, esto depende de la potencia que el radar pueda estar emitiendo, esta puede ser controlada por el usuario y basta con emitir aproximadamente una potencia de 2(W) para que este sistema supere los 30 (m) de visibilidad de las cámaras, debido a que a esa distancia sólo se tendría una atenuación de 0.0336 (W).

Debido a la disponibilidad de materiales en la universidad, se usará el radar Doppler de onda continua. A continuación se presentan algunos sistemas en donde ya se está empleando:

- Reconocimiento del movimiento humano (caminar) [22]: el funcionamiento de esta aplicación de radar es que, cuando la señal enviada por el radar impacta en una persona que está en movimiento, la señal reflejada desde las distintas zonas del cuerpo tendrán un desplazamiento Doppler, que será proporcional a la velocidad de estas. La contribución de la velocidad se verá afectada de acuerdo al lugar se haga referencia.
  La componente primaria de la reflexión de la señal será el torso, brazo y piernas. Para una persona caminando con una velocidad constante V_0, la señal reflejada desde el torso $S_0(t)$ tendrá una fase constante de Doppler. Sin embargo, la señal reflejada por la velocidad de brazos y piernas $S_m(t)$ será modulada a la frecuencia f_m, que es la tasa de oscilación de los brazos o piernas como muestra la Figura 1-14. En general los brazos y piernas tendrán el mismo periodo de balanceo.

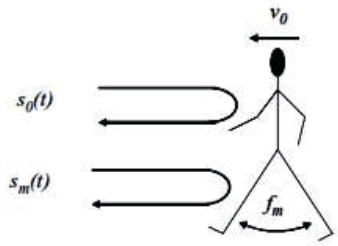


Figura 1-14 Representación de señales reflejadas The MITRE Corporation [22].

Al extraer las componentes de velocidad del objetivo por el radar, se aplica la transformada rápida de Fourier, para extraer las componentes espectrales.
La señal obtenida es guardada es un ordenador portátil o un computador de escritorio, como un archivo WAV. Esto permite un procesamiento más fácil de la señal usando Matlab.
En la Figura 1-15 se muestra un espectrograma típico de una medición Doppler, adquiriendo los detalles de una persona en movimiento.

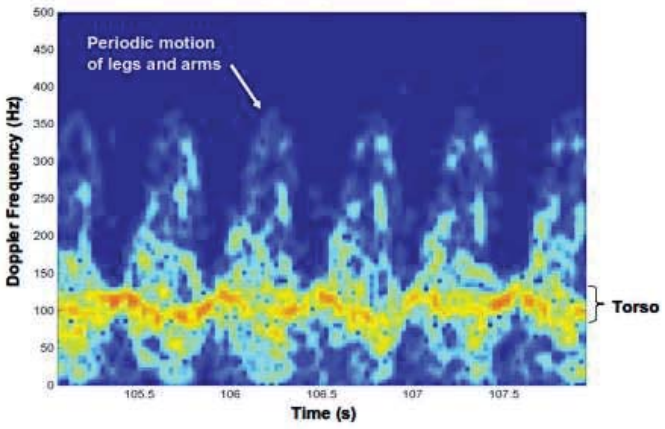


Figura 1-15 Espectrograma de persona caminando [22].

A continúa se presentan los elementos que dispuso la universidad para la realización del proyecto: el radar Doppler de la empresa RFBEAM y un módulo Drone.

- Módulo Radar Doppler RSP1: este radar será usado a lo largo del proyecto debido a la disponibilidad en el momento. Funciona como detector de movimiento, siendo capaz de detectar objetos que se desplazan a velocidad incluso de 250(Km/h) [23]. Sobre este instrumento mostrado en la Figura 1-16 se va a profundizar en el capítulo siguiente.

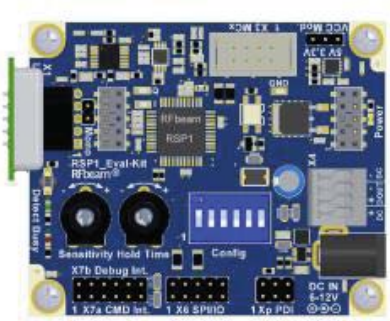


Figura 1-16 RSP1 radar Doppler (fuente: RFbeam.ch).

- Módulo Drone HKpilot 32: el HKPilot32, es uno de los sistemas de piloto automático más avanzados disponibles, con soporte para casi cualquier tipo de vehículo multirotor, incluso un submarino [24]. En la Figura 1-17 se muestra este dispositivo. Esta herramienta se encarga de todo el procesamiento de información necesario para poder ejercer el vuelo en un Drone y envía las correspondientes indicaciones a los rotores de este.

Figura 1-17 HKPilot32 (fuente: https://hobbyking.com).

1.4 Conclusiones

En este capítulo se presentaron los puntos fundamentales para presentar el proyecto, analizando algunas soluciones del problema que han propuesto otros desarrolladores. Sin embargo, debido a las condiciones climáticas que existen en la región de Valparaíso se necesita una solución más robusta en caso de neblina o condiciones meteorológicas más dificultosas. Para suplir esto, se utiliza la opción de los radares, ya que se demostró que su atenuación es más baja en situaciones meteorológicas más difíciles, siendo este un buen indicio para poder guiar el vehículo aéreo no tripulado.

Para finalizar se presentaron los elementos a usar para la continuación del proyecto, siendo el radar Doppler RSP1 y el vehículo aéreo no tripulado con su procesador HKPilot32.

2 Módulo Radar Doppler

En este capítulo se explica con mayores detalles las funcionalidades del radar Doppler genérico, mostrando sus aplicaciones, ecuaciones, entre otros. Además se profundizará en el módulo RSP1 explicando cómo funciona, revisando su estructura, datasheet, e interfaces gráficas para el manejo del módulo, destacando lo más importante para el proyecto.

2.1 Radar Doppler de onda continua

El radar Doppler de onda continua, tal como su nombre lo indica, utiliza una forma de onda continua que se puede aproximar a una sinusoidal de la forma $\cos(2\pi f_0 t)$. Los espectros de los ecos de los objetos sin movimiento estarán concentrados en la frecuencia f_0. El centro de la frecuencia de los ecos desde objetivos en movimiento se desplazará f_d, conocida como la frecuencia Doppler. Así, mediante la medición de esta diferencia de frecuencias, se puede extraer de forma muy precisa la velocidad radial de los objetos [17]. Debido a la naturaleza continua de los emisores de onda continua, la medición no es posible sin algunas modificaciones en las operaciones del radar y formas de onda, que se discutirán más adelante. Al ser la señal de eco recibida y procesada de forma permanente, se deben resolver dos problemas por motivos de lo anterior [25];

- Evitar una conexión directa de la energía transmitida en el receptor (conexión de realimentación).
- Asignar los ecos recibidos a un sistema de tiempo para ser capaz de hacer mediciones de tiempo de "ida" y "vuelta" (tiempo de ejecución) de la señal.

Una conexión directa de la energía transmitida en el receptor se puede prevenir por [25]:

- Separar de forma espacial la antena transmisora de la receptora.
- Separación dependiente de la frecuencia.

Una medición de tiempo de ejecución no es necesaria para los medidores de velocidad.

2.2 Diagrama de bloques funcional

Con el fin de evitar interrupciones en la emisión del radar, son usadas dos antenas en este: una para transmitir y la otra para recibir.

En la Figura 2-1 se muestra un diagrama de bloques de un radar de onda continua la que exhibe cómo se procesa la señal para poder obtener la frecuencia Doppler.

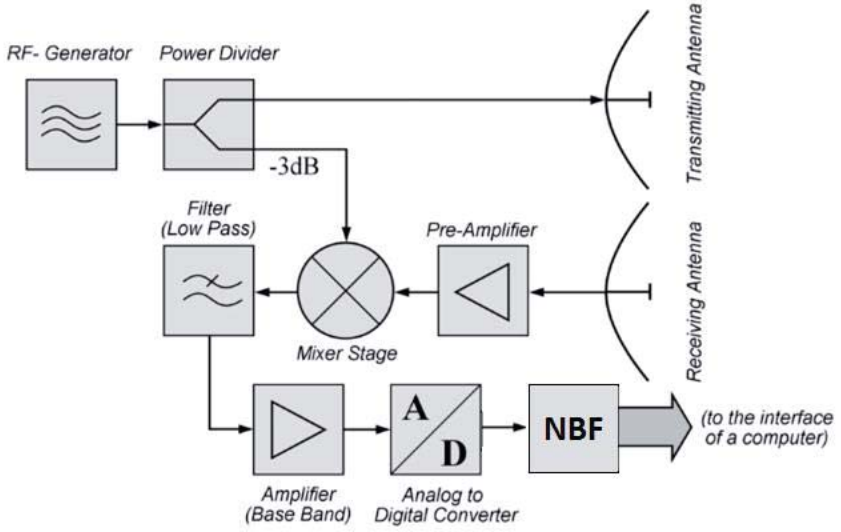


Figura 2-1 Diagrama de bloques del radar de CW (fuente: www.ratartutorial.eu).

La estructura de un radar Doppler para mediciones de velocidad es muy simple. Todo el circuito de transmisor y receptor se puede realizar con dispositivos semiconductores sobre un sustrato como un componente integrado. Este dispositivo se suele llamar transceptor, un acrónimo de las palabras "transmitter" y "receptor". En muchos casos, este transceptor ya está provisto de las antenas necesarias. Mayormente estas antenas están en una placa de circuito impreso [25].

En un receptor de conversión directa la señal de eco no se convierte a una frecuencia intermedia. Este proceso se realiza en el mezclador y junto a la frecuencia del generador, el mezclador hace la comparación y además sitúa la señal a una frecuencia banda base [25].

Con el fin de trasladar la señal de eco, los mezcladores utilizados requieren una fuente de poder, siendo esta fuente la mitad de la potencia del generador, extrayendo dicho valor mediante el divisor de potencia. A continuación hay un filtro pasa bajos, con la intención de bloquear las componentes del alta frecuencia, para terminar en el amplificador que entregará la ganancia correspondiente para alimentar a la siguiente etapa del circuito o lo que pueda continuar luego de terminado este proceso [25].

2.3 Ecuaciones de radar de onda continua

Ya que el bloque NBF implementa la transformada de Fourier y se está trabajando bajo instrumentos no ideales, posee una finita capacidad de datos para poder procesar en un tiempo determinado. La duración de estos bloques es conocida como "tiempo de permanencia" o "intervalo de permanencia". La duración del intervalo determina la resolución de frecuencia o el ancho de banda del NBF. [26];

$$\Delta f = \frac{1}{T_{Dwell}} \tag{2-1}$$

T_{Dwell} es el intervalo de permanencia. Una vez que se elige la frecuencia máxima resoluble por el banco NBF, el tamaño del banco de NBF se calcula como [26];

$$N_{FFT} = \frac{2B}{\Delta f} \tag{2-2}$$

B es la máxima frecuencia resoluble por la transformada de Fourier. EL factor 2 es necesario para una cuenta de ambos desplazamientos Doppler, positivos y negativos. Dando como resultado [26]:

$$T_{Dwell} = \frac{N_{FFT}}{2B} \tag{2-3}$$

La ecuación del radar general que expresa la relación señal ruido de las antenas es:

$$SNR = \frac{P_{av} T_i G \lambda^2 \sigma}{(4\pi)^3 R^4 k T_e F L} \tag{2-4}$$

En el caso de los radares de onda continua, P_{av} es reemplazada por la potencia media transmitida en el intervalo de permanencia P_{CW} y T_i es reemplazada por T_{Dwell}. Por lo tanto la ecuación queda de la siguiente forma [26]:

$$SNR = \frac{P_{cw} T_{Dwell} G_t G_r \lambda^2 \sigma^{\ 2}}{(4\pi)^3 R^4 k T_e F L L_{win}} \tag{2-5}$$

Donde G_t y G_r son la ganancia de la antena transmisora y receptora. El factor L_{win} son las pérdidas térmicas asociadas con el tipo de procesamiento usado en la computación de la transformada de Fourier. Los otros términos en la ecuación 2-5 serán definidos a continuación:

- λ, es la longitud de onda de la señal.
- σ, definida como la sección transversal del radar.
- R, es la distancia a la que se encuentra el objetivo del radar.
- K, es la constante de Boltzman que posee un valor de $1.38 \cdot 10^{-23}$(J/°K).
- T_e, temperatura presente en grados Kelvin
- F, es el factor de fidelidad del receptor de radar, el que se denomina factor de ruido F.
- L, son las pérdidas internas del radar.

Dependiendo de la forma en que se procesa la señal, los radares se pueden clasificar de distintas formas, debido a la finalidad del proyecto y los elementos que ya se poseen, es que, se tratarán los detectores I&Q [26].

2.4 Señal procesada por radar de onda continua I&Q

Una vez obtenida la señal digital, se puede extraer la amplitud mediante una transformación análogo/digital, sin embargo, aún falta la información de fase del obstáculo. Para esto existe el

método de detector sincrónico [26]. Para poder procesar una señal continua, se requiere de una conversión análoga/digital. No obstante, mediante este conversor se obtendrá el comportamiento de las amplitudes de la señal, pero discriminando la fase. Para resolver este problema, existen los radares "I&Q" o en fase y cuadratura. El objetivo de este tipo de radares es la de poder tener la representación digital de la señal, pero sin perder información. Para esto, se utiliza un par de conversores análogos digitales junto a un par de mezcladores.

El funcionamiento de este sistema I&Q aprovecha que estas señales se pueden trabajar como vectores complejos. Con un conversor análogo/digital se obtiene la parte real de la señal. Para obtener la parte imaginaria al inicio del sistema y usando otra ruta se restan 90° al vector inicial, dejando la parte imaginaria en el eje real como muestra al Figura 2-2 y con otro conversor análogo digital se obtiene lo requerido.

Con este sistema se obtienen las dos componentes que describen al vector complejo, siendo la parte real "I" y la parte imaginaria "Q". Con estos componentes, se puede calcular la amplitud y fase de la señal correspondiente, mediante trigonometría [25] [27] .

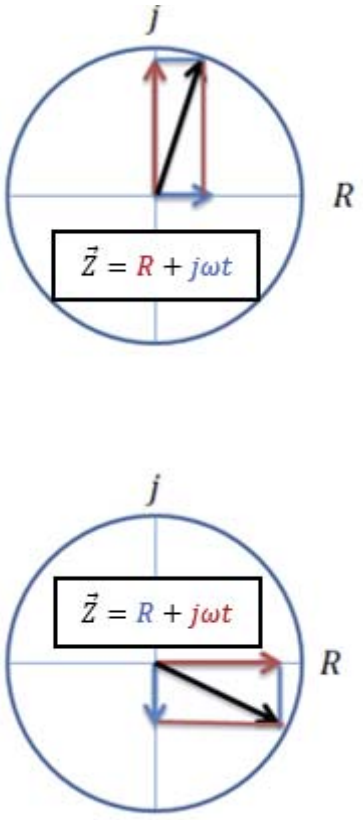


Figura 2-2 Vector complejo original y vector desfasado.

La relación que poseen los componentes del vector complejo son las siguientes:

$$I = A\cos\theta \tag{2-6}$$

$$Q = Asin\theta \tag{2-7}$$

Y mediante trigonometría se obtiene:

$$A^2 = I^2 + Q^2 \tag{2-8}$$

$$\theta = tan^{-1}\left(\frac{Q}{I}\right) \tag{2-9}$$

Y en la Figura 2-3 se muestra el diagrama de bloques funcional de este tipo de radar I&Q.

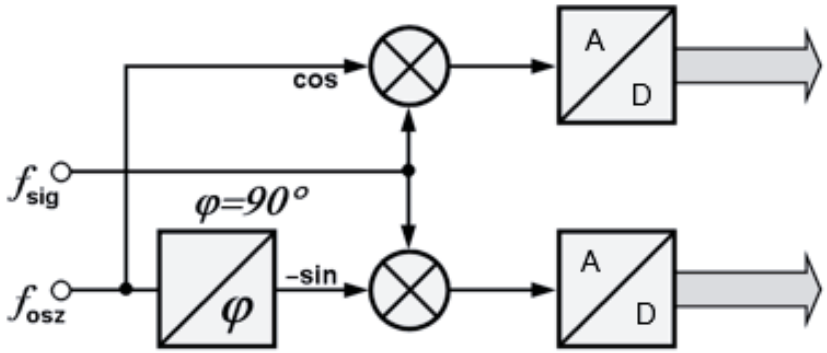


Figura 2-3 Diagrama de bloques funcional del radar I&Q (fuente: www.radartutoutori.com).

2.5 RSP1 RFBEAM

RSP1 es el radar de onda continua I&Q comercial usado en el proyecto, además es el primer radar de RFBEAM que funciona como procesador de señal.

RSP1 contiene todo el procesamiento de señal para radares Doppler. Puede detectar objetos que se estén desplazando a bajas velocidades, como objetos que van a 250 (km/h). Este procesador tiene la cualidad de cancelar el ruido y adaptarse automáticamente a diferentes transceptores Doppler. Funcionalmente se pueden modificar por opciones manuales alrededor de 30 parámetros y comandos. El kit puede ser usado en un sistema independiente o como un servidor de un equipo de computador o microcontrolador [23] [28].

2.6 Características principales

Las características principales del módulo Doppler RSP1 son las siguientes:

- Arquitectura del hardware: La arquitectura del procesador, mostrada en la Figura 2-4 permite la adquisición y procesamiento de datos en paralelo. Sólo unos pocos componentes externos son necesarios gracias al alto nivel de integración, incluyendo la EPROM y generador de reloj de precisión [23] [28].

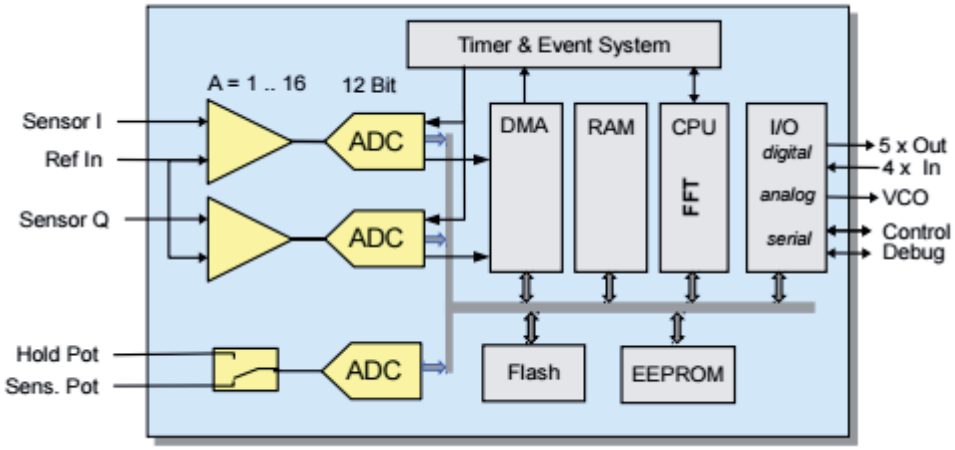


Figura 2-4 Arquitectura del procesador [23].

Los conectores e indicadores se muestran en la Figura 2-5 [23] [28].

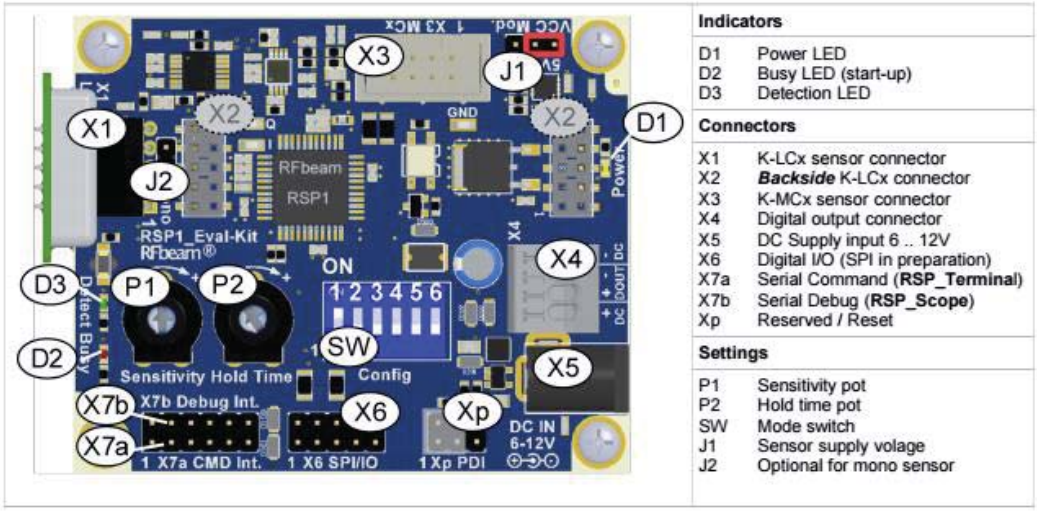


Figura 2-5 Conectores e Indicadores de RSP1 [28].

- Adquisición de datos: RSP1 trabaja con 2 convertidores análogo-digital de 12 bits cada uno. La velocidad de muestro se puede seleccionar entre 1200 (Hz) a 22.5 (KHz). Esto corresponde a las máximas velocidades entre 13 (km/h) a 250 (km/h) [23].
- Procesamiento de datos: el procesamiento está basado en una FFT y un umbral de ruido adaptativo. Muchos de los parámetros permiten ajustar y optimizar el rendimiento para muchas aplicaciones [23].
- Algoritmo de detección: el algoritmo de detección está basado en la FFT (rápida transformara de Fourier) de la señal I&Q. Esta es una salida FFT logarítmica con el fin de obtener buenas condiciones de procesamiento de señales.

FFT representa de hecho, muchos filtros de banda estrecha que reducen la amplitud de ruido. RSP1 utiliza 256 FFT, resultante en 128 filtros para cada uno de los movimientos de avance y retroceso. Este tipo de detección se traduce en una mejor sensibilidad [23].

- Detección de ruido adaptativo: la técnica de detección que usa el radar Doppler RSP1 conduce a una excepcional sensibilidad de la resolución [23].
  El ruido se mide separadamente para cada frecuencia representada por los resultados de la FFT. Dos etapas de medición son las que realizan esta tarea:
  Después del encendido, una curva de ruido inicial se construye mediante la medición del promedio de cada filtro FFT. El número de medias (tiempo de medición) puede ser seleccionado por el parámetro S04 [23].
  La media de adaptación está construida de forma continua durante el funcionamiento. La constante de tiempo de adaptación puede ser seleccionada por el parámetro S0C [23].
- Procesamiento de la señal I&Q: RSP1 soporta procesamiento I&Q usando FFT. La Señal Doppler I&Q posee un fase de +90° o -90° como lo muestra la Figura 2-6. Estas señales aparecen ya sea en el plano real (derecha) o en el plano imaginario (izquierda) de la salida FFT. La señal en el centro es la DC offset causada por el amplificador y conversor ADC y puede ser ignorado.
  Algunas ventajas de usar I&Q comparado con el sensor de un solo canal: Diferenciación entre movimiento en contra o a favor del radar, eficiente supresión de interferencia y supresión de vibración.
  Incluso si no se requiere detección bidireccional, el procesamiento de la señal I&Q significa una mejor supresión del ruido.

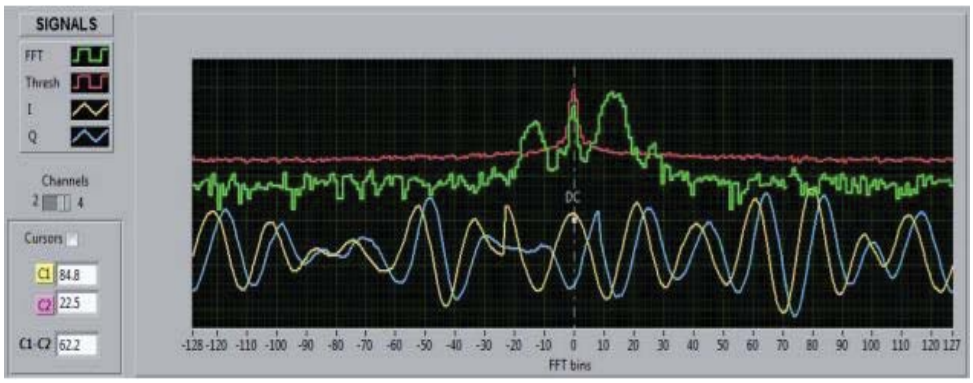


Figura 2-6 Desfase de 90° en señal I&Q [28].

- Filtro de interferencia: típicas interferencias aparecen simétricamente en el plano izquierdo y derecho de la salida de la FFT. Típicos ruidos de fuentes son de componentes electrónicos y luces fluorescentes.
  Además estas señales de interferencias pueden ser fácilmente distinguidas por la señal Doppler I/Q producida por el sensor estéreo modulado.
  RSP1 adapta el umbral (línea roja) para el ruido, pero no se adapta para las señales Doppler I&Q.
  En la Figura 2-7 se muestra como el umbral se adapta a la interferencia.

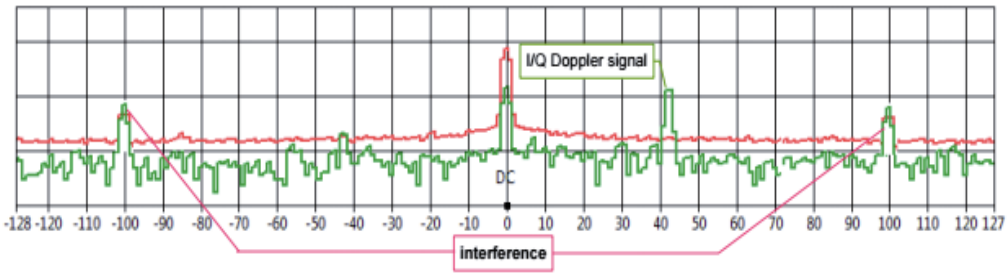


Figura 2-7 Adaptación de umbral [28].

- Filtro de ruido aleatorio: RSP1 posee diferentes mecanismos con el fin de suprimir la influencia del ruido estocástico. El ruido aleatorio produce distribuciones de amplitudes estocásticamente en la salida del FFT, sin embargo, pueden ser reducidas calculando la media a estas amplitudes. FFT promediado puede ser habilitado por el parámetro S02. Los ruidos con amplitudes que superen el trigger se cuenta y deben superar un umbral de contador ajustado por el parámetro A03 explicado en el punto 2.7.
- Interfaces: RSP1 provee diferentes interfaces para la configuración y control de la señal. RSP1 puede ser usado como un procesador individual o en conjunto con un equipo controlador. Algunas interfaces son: Interfaces de comando, interfaces de depuramiento y digital I/O.
- Puertos: con fines del proyecto, se hará mención al puerto que se usará para la realización de este. El puerto usado es el x7a Serial Command Interface mostrado en la Figura 2-8, con sus características en la Tabla 2-1.

Figura 2-8 Interfaz X7b [23].

Tabla 2-1 X7a Interfaz de comando serial

Pin	Señal	Descripción
1	GND	Alimentación GND
2	Nc	No conectado
3	+5 (V)	Alimentación de entrada
4	RxD	Data de entrada serial UART
5	TxD	Data de salida serial UART
6	Nc	No conectado

Es importante recalcar que los pines RxD y TxD trabajan con 3.3 (V) y que usan protocolo de comunicación UART.

El uso de este puerto es exclusivamente para el uso de la interfaz gráfica RSP_Terminal, ya que, como se verá más adelante, se usará dicha interfaz para la obtención de su cadena de datos.

- Interfáz gráfica RSP_Terminal: el procesador RSP1 puede ser influenciado por distintos parámetros. RSP_terminal entrega la posibilidad de ver y configurar aquellos parámetros. Dicho de otra forma, RSP_Terminal emula un ordenador o microprocesador utilizado en un hardware de usuario basado en RSP1.
  El comando importante y con el que se trabajará es "$L00", debido a que envía una cadena constante de datos que se muestra en la Figura 2-9 [28] y con la que se trabajará más adelante.

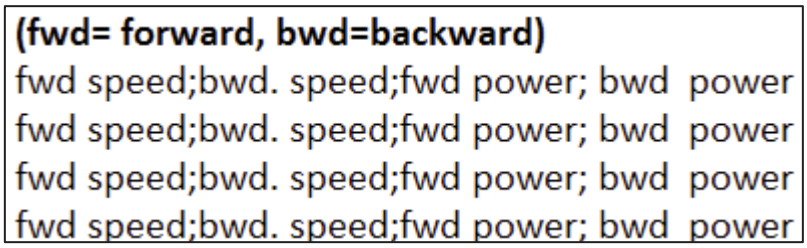


Figura 2-9 Cadena de datos enviada por comando $L00 [23].

2.7 Parámetros

En el radar Doppler RSP1, existe una extensa lista de parámetros que pueden ser modificados. Sin embargo, para el proyecto el más relevante es el de la velocidad de muestreo, debido a que el valor de la velocidad entregada por el RSP1 variará de acuerdo a la velocidad a la que este tome las muestras. Este parámetro se manipula con el comando S03. De acuerdo a la Tabla 2-2 mostrada varía la capacidad de toma de velocidades de acuerdo a la velocidad de muestreo [23].

Tabla 2-2 Parámetro S03

Parámetro S03	Frecuencia de muestreo (Hz)	Resolución (Hz)	Máxima frecuencia (Hz)	Resolución en (km/h)	Máxima velocidad (km/h)	Actualizar tiempos (ms)
01	1280	5	640	0.11	14.5	200
02	2560	10	1280	0.23	29.1	100
03	3840	15	1920	0.34	43.6	67
04	5120	20	2560	0.45	58.2	50
05	6400	25	3200	0.57	72.7	40
06	7680	30	3840	0.68	87.3	33
07	8960	35	4480	0.80	101.8	29
08	10240	40	5120	0.91	116.4	25
09	11264	44	5632	1.00	128.0	23
0A	22530	88	11265	2.00	256.0	12

2.8 Conclusiones

Lo que se puede extraer y aprender sobre este sistema basado en el efecto Doppler de onda continua, con la complementación de un sistema I&Q", es la velocidad a la cual un objeto u obstáculo se acerca o aleja desde el Radar. Todas estas propiedades las posee un módulo perteneciente a la empresa RF BEAM de nombre RSP1. Este módulo tiene la propiedad de procesar la información para luego, a través de conectores específicos, poder extraer la información necesaria para los fines de este proyecto.

A través de los puertos seriales UART del módulo, se extraerá la información de esta placa y se manipulará con algún tipo de procesador, para terminar enviando esa cadena de datos al Drone y este pueda tener conocimiento sobre los obstáculos que podrían estar frente a él.

Antes de estudiar lo nombrado anteriormente, se analizará el procesador HKPilot32, junto a sus códigos correspondientes y software a usar para poder manipularlo.

3 Módulo Controlador de Drone

El HKPilot32 es uno de los sistemas de piloto automático más avanzadas disponibles, con soporte para casi cualquier tipo de vehículo multirotor incluso un submarino. Diseñado por el proyecto hardware abierto PX-4, que se apoya en una serie de comunidades de desarrollo haciendo de esta una de las plataformas más flexibles y fiables para el control del vehículo [24].

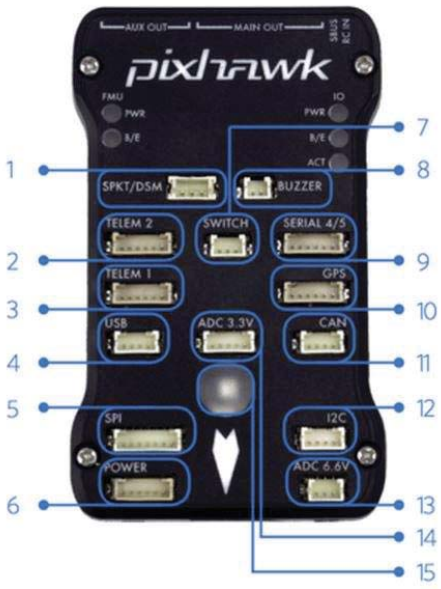
3.1 Arquitectura del Hardware

En la Figura 3-1 se muestran los puertos que HKPilot32 posee como entradas, las que generalmente se usan para que se pueda obtener la información necesaria para poder llevar un buen vuelo [24]. Sin embargo, todos esos puertos no son necesarios para el vuelo del Drone, por ejemplo, los puertos seriales 4/5 o TELEM2 se pueden usar de forma opcional y justamente el primero que se nombró será el usado para poder comunicarse con el radar.

Para fines del proyecto se hará referencia al puerto que se usará para la comunicación con el radar y se muestra en la Tabla 3-1.

Tabla 3-1 Pines de puerto Serial 4/5

Pin	Señal	Volt
1	Vcc	+5(V)
2	TX(#4)	+3.3(V)
3	RX(#4)	+3.3(V)
4	TX(#5)	+3.3(V)
5	RX(#5)	+3.3(V)
6	GND	GND



1 Spektrum DSM receiver
2 Telemetry (radio telemetry)
3 Telemetry (on-screen display)
4 USB
5 SPI (serial peripheral interface) bus
6 Power module
7 Safety switch button
8 Buzzer
9 Serial
10 GPS module
11 CAN (controller area network) bus
12 I²C splitter or compass module
13 Analog to digital converter 6.6 V
14 Analog to digital converter 3.3 V
15 LED indicator

Figura 3-1 Puertos de HKPilot32 [23].

Debido a que el puerto serial 9 (mostrado en la Figura 3-1) trae consigo 2 puertos seriales (los puertos 4/5) implica que posee 2 transmisores y 2 receptores, sin embargo se necesita solo un conector para el transmisor y otro para el receptor escogiendo el puerto serial 4, teniendo como característica que puede usar el protocolo de comunicación UART [24]. Por lo tanto el puerto serial 4 será la estructura física del módulo del Drone usado a lo largo del proyecto, faltando explicar el código de dicho módulo.

3.2 Código HKPilot32

El código de HKPilot32 consta de 2 partes principales: un código base y el código propio del vehículo.

3.2.1 Código base: ArduPilot

El código es muy extenso (sobre 70.000 líneas de código obtenidas del núcleo git) [29]. En este apartado se hará una explicación para que se pueda tener un conocimiento necesario para una comprensión global del código, debido a que explicar por completo la programación se escapa de las finalidades del proyecto.

La estructura básica del código se muestra en la Figura 3-2 que de forma ilustrativa exhibe la organización a nivel global del sistema entre hardware y las interfaces de usuario.

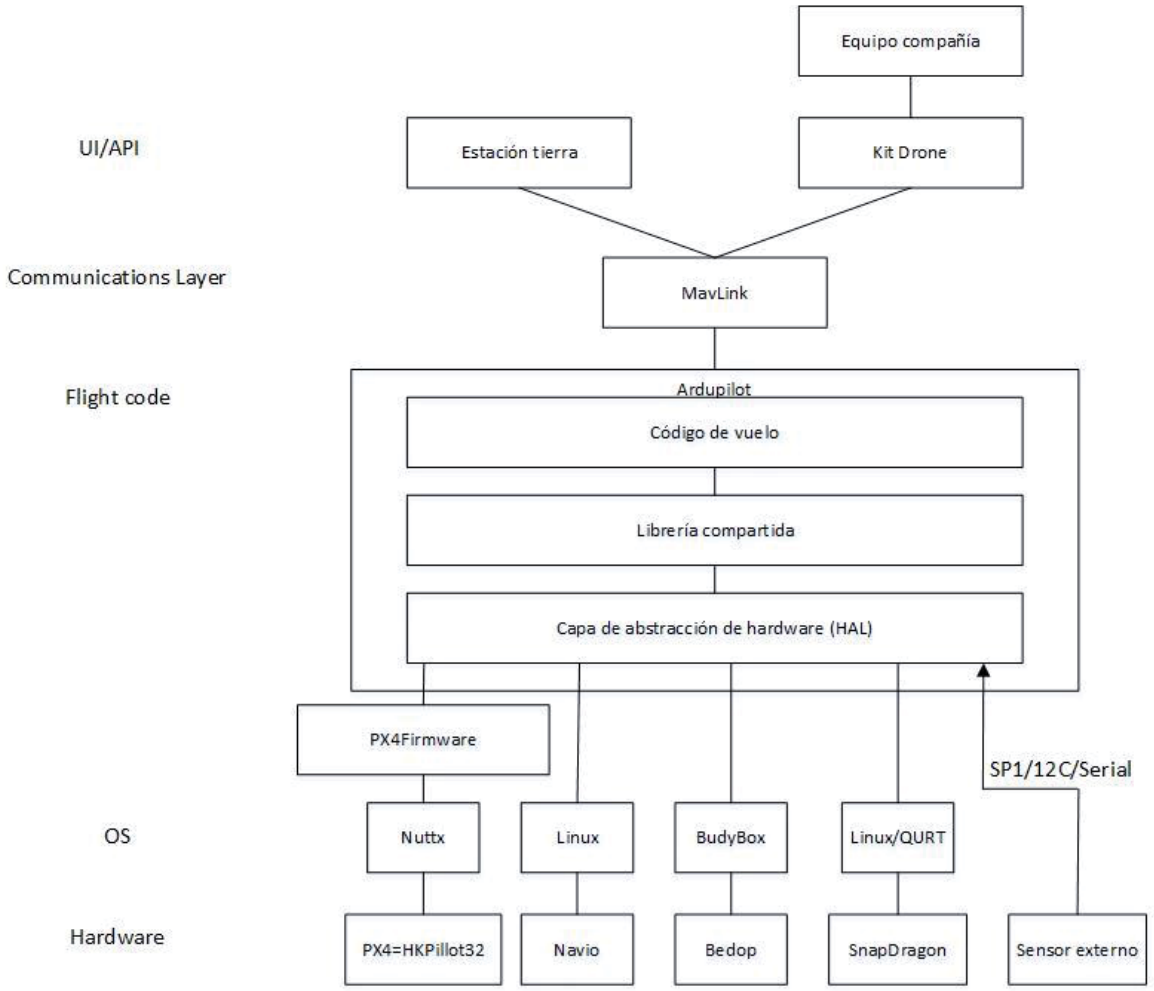


Figura 3-2 Estructura básica ArduPilot (fuente: http://ardupilot.org)

Como lo muestra la figura anterior, la estructura básica se encuentra divida en 5 partes principales [29]:

- Directorios de vehículos: son los directorios de nivel superior que definen el firmware del tipo de vehículo habitual que se usará: planeador, helicóptero, APMrover2 o perseguidor de antena.

AP_HAL hardware Abtraction Layer (capa de abstracción de hardware): en esta sección se hace portátil Ardupilot a muchas plataformas. Existe un AP_HAL de nivel superior en las librerías, que define la interface al resto del código para las características de la placa específica y además hay un subdirector AP_HAL_XXX para cada tipo de tabla. Por ejemplo; AP_HAL_AVR para tablas basadas en AVR, AP_HAL_PX4 para placas PX4 y AP_HAL_Linux para tablas basadas en Linux.

Librerías: Las bibliotecas se comparten entre los cuatro tipos de vehículos Copter, Plane, Rover y AntennaTracker. Estas bibliotecas incluyen controladores de sensores, estimación de posición y posición (también conocido como EKF) y código de control (es decir, controladores PID).

Directorio de herramientas: son varios directorios de soporte. Por ejemplo: Tools/autotest provee la infraestructura de autotest detrás del sitio autotest.ardupilot.org y tolos/replay proporciona la utilidad de reproducción de registros.

Código de soporte externo: en algunas plataformas necesitamos código de soporte externo para proporcionar funciones adicionales o soporte de placa.

La comunicación entre la consola y los puertos seriales UARTs es fundamental, debido a que muchos componentes dependen de esta comunicación. Entre ellos se encuentra cargar salidas, telemetría, módulos GPS y más. La comprensión de la comunicación con los UARTs a través de HAL puede ayudar a entender en gran medida el código de Ardupilot.

El Ardupilot Hal generalmente define 5 UARTs. El HAL por sí solo, no define algún rol particular de los UARTs, sin embargo, en otras secciones del código de ArduPilot se le asignan funciones particulares.

- UART "A": la consola (usualmente USB, ejecuta la telemetría MAVLink).
- UART "B": el primer GPS.
- UART "C": telemetría primaria (Telemetría 1 en HKPilot, radio en APM2).
- UART "D": telemetría Secundaria (Telemetría 2 en Pixhawk).
- UART "E": segundo GPS.

Si se escribe el "sketch" (siguiendo la lógica de Arduino) usando ArduPilot HAL, entonces se puede usar UARTs para cualquier propósito que se quiera [29].

Algunas funciones pueden ser:

- Writte: escribe una cadena de bytes.
- Read: lee algunos Bytes.
- Available: compruebe si hay byte en espera.

3.2.2 Código de vehículo

Existen 4 vehículos comunes [29]:

- Copter: para multi-copteros y helicópteros.
- Plane: para aviones de ala fija.
- APMover2: para vehículos terrestres y botes.
- Antenna Tracker: para buscadores de antena.

Si bien hay una gran cantidad de elementos comunes entre los diferentes tipos de vehículos, no dejan de ser diferentes. Con fines del proyecto, se concentrará en el tipo "copter".

Para iniciar con los detalles de este código, se comienza con un resumen para obtener primeramente una visión global, para luego entrar en mayores detalles. Este código está formado

por el "Código Copter principal" que reside en su propio directorio y las bibliotecas que se comparten con el vehículo y el controlador.

En la Figura 3-3 se explica el código para el vehículo cuando está en modo manual, mientras Figura 3-4 lo explica para el caso automático, ambas de forma ilustrativa.

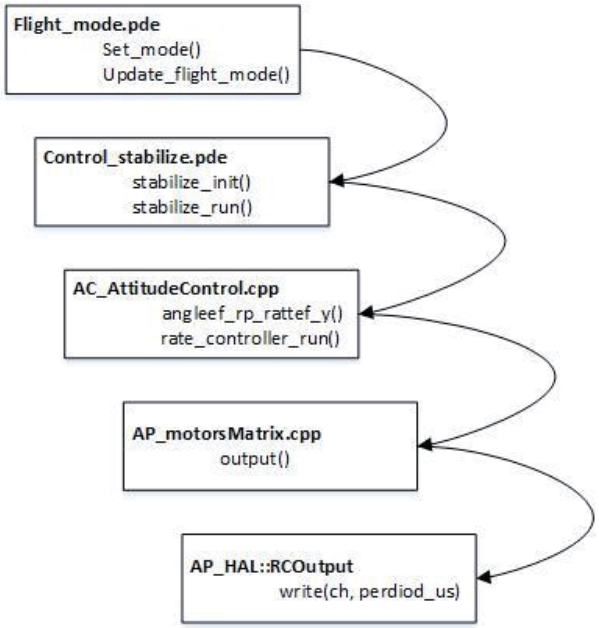


Figura 3-3 Ilustración de código en modo manual (fuente: http://ardupilot.org).

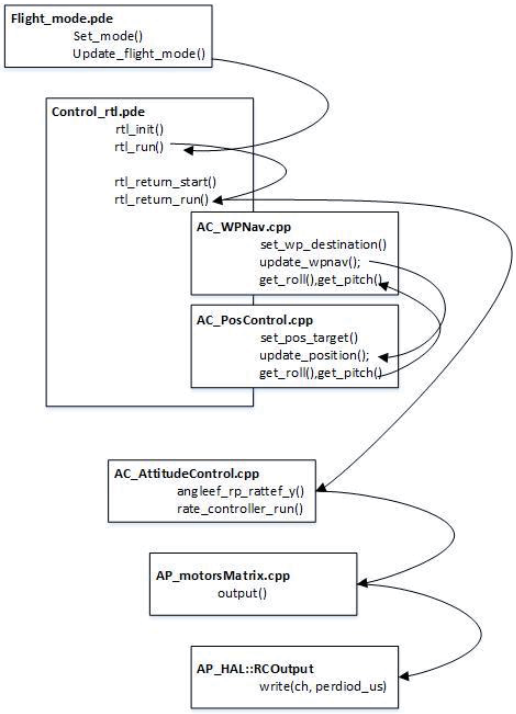


Figura 3-4 Ilustración de código en modo automático (fuente: http://ardupilot.org).

Para el caso manual los comandos más relevantes son [29]:

- Update_flight_mode: comprueba la variable "control_mode" y llama a la función modo de vuelo "specific_run()".
- Stabilize_run: interpreta la entrada del piloto y establece los ángulos de balanceo, inclinación y giro (o velocidad).
- AttitudeControl::angleef_rp:rateef_y: calcula el error "attitude" y lo convierte en respuesta para los motores.
- AP_MotorsMatrix::output: convierte esa respuesta en salida para los motores.
- AP_HAL::RCOutput::write: envía mensaje PWM a ESCs(controles electrónicos de estabilidad).

Y para el caso automático son los siguientes [29]:

- Update_flight_mode: comprueba la variable "control_mode" y llama a la función modo de vuelo "specific_run()".
- Rtl_run: usa la variable rtl_state para decidir cuál de la subfunción a llamar.
- Rtl_return_run: llama al controlador de navegación de "waypoint" para obtener el último valor deseado de "roll", "pitch" y "throttle".
- Update_wpnav: calcula el error de posición y velocidad y actualiza los objetivos de "poscontrol".
- Update_position: calcula los ángulos deseados de inclinación y aceleración. Los ángulos de inclinación son retornados al llamar mediante get_roll(), get_pitch().
- AttitudeControl::angleef_rp:rateef_rpy: calcula el error "attitude" y lo convierte en respuesta para los motores.
- AP_MotorsMatrix::output: convierte esa respuesta en salida para los motores.
- AP_HAL::RCOutput::write: envía mensaje PWM a ESCs..

3.3 Softwares de apoyo

De apoyo se usaron tres softwares para poder visualizar, editar y cargar los cambios que se realicen en el código:

3.3.1 QTCreator

Este software se usó para poder visualizar el código del HKPilot32 y también la edición del mismo.

En la Figura 3-5 se observa como el código está dividido en sub-códigos y en Figura 3-6 como se visualiza el código del vehículo usado [24].

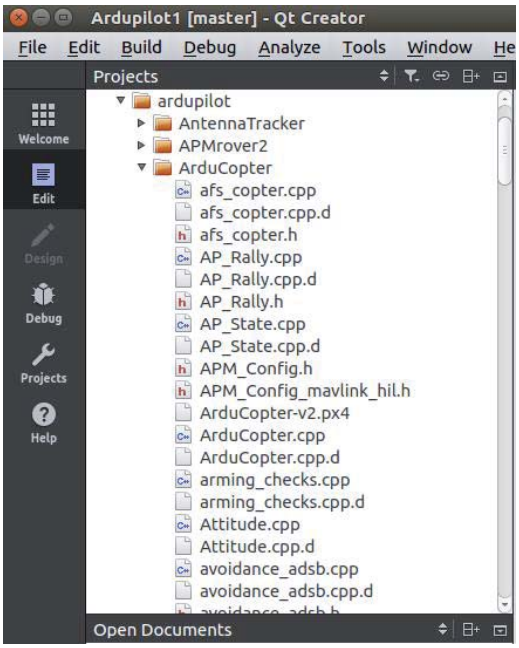


Figura 3-5 Muestra de sub-códigos.

arming_checks.cpp | Copter::barometer_checks(bool): bool

```
75
76      return barometer_checks(display_failure)
77          & rc_calibration_checks(display_failure)
78          & compass_checks(display_failure)
79          & gps_checks(display_failure)
80          & fence_checks(display_failure)
81          & ins_checks(display_failure)
82          & board_voltage_checks(display_failure)
83          & parameter_checks(display_failure)
84          & motor_checks(display_failure)
85          & pilot_throttle_checks(display_failure);
86  }
87
88  bool Copter::rc_calibration_checks(bool display_failure)
89  {
90      // pre-arm rc checks a prerequisite
91      pre_arm_rc_checks();
92      if (!ap.pre_arm_rc_check) {
93          if (display_failure) {
94              gcs_send_text(MAV_SEVERITY_CRITICAL,"PreArm: RC not calibratedasdsad");
95              hal.console->begin(38400);
96              hal.uartE->begin(38400);
97              while(hal.uartE->available())
98              {
99              hal.console->write(hal.uartE->read());
100             stabilize_run();
101             }
102         }
103         return false;
```

Figura 3-6 Visualización de código.

3.3.2 Ventana terminal de Linux

La ventana terminal de Linux, fue usada única y exclusivamente para compilar y cargar las modificaciones hechas al código en la tarjeta de HKPilot32. En la Figura 3-7 muestra el término de cargar el código en el HKPilot32 [24].

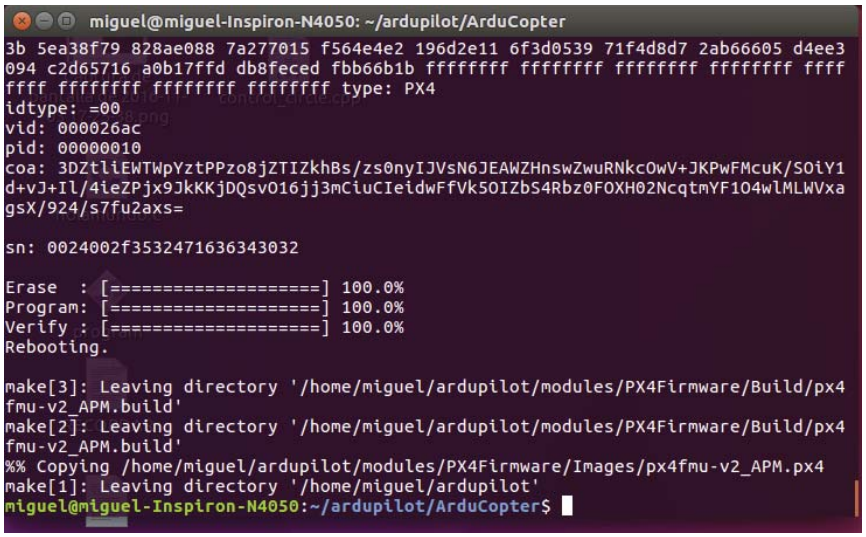


Figura 3-7 Carga de códigos en HKPilot32.

3.3.3 Arduino IDE

Este software también tuvo un uso muy definido, visualizando lo que sucedía en los puertos serial del HKPilot32 una vez se modificaba y cargaban los códigos modificados.

3.4 Conclusiones

En el capítulo se estudió de forma individual el HKPilot32, los puertos seriales, la programación y los programas utilizados para el completo uso de este.

Hasta el momento, se han estudiado los elementos fundamentales del proyecto, tales como el HKPilot32 y el radar. Sin embargo, aún queda unirlos y para ello en los capítulos siguientes se estudiará lo necesario para esto, con sus respectivos protocolos y códigos a usar para finalizar en la comunicación del sistema completo.

4 Sistema de Comunicación

Con la finalidad de obtener una navegación autónoma en el Drone, en primera instancia se estudió el uso de un radar Doppler (módulo RSP1 de RFBEAM) obteniendo muy buenos resultados, debido a que se descubrió que utiliza puerto seriales de protocolo UART que coinciden con los protocolos usados por los puertos seriales de arduino. Se usó el modelo arduino Uno para, primeramente establecer una comunicación entre el radar y el Arduino, finalizando con la obtención de los datos del radar para su posterior procesamiento.

Lo anterior será visto en este capítulo, dando un estudio sobre lo que es y cómo funciona el protocolo UART, indicando el funcionamiento de las partes más esenciales en la programación que se debió seguir en el Arduino para cumplir con la comunicación radar-drone.

Además de analizar la comunicación que deberá tener el radar con el Arduino como etapa intermedia, también se explicará la que deberá tener el HKPilot32 con el Arduino, para luego poder lograr una comunicación completa entre todo el sistema.

4.1 Comunicación RSP1 y Arduino

La primera etapa para la comunicación del radar Doppler con el drone, consta en obtener los datos entregados por el radar y procesarlos de tal manera que se puedan trabajar y utilizar de la manera correcta para el control del vuelo automático. Para esto se escogió el uso de un arduino Uno para la adquisición y procesamiento de los datos.

4.1.1 Sistema de comunicación

Ambos elementos, tanto el radar como el drone en sus puertas seriales, utilizan el sistema de comunicación UART (Universal Asynchronous Receiver/Transmitter o receptor/transmisor asincrónico universal). La función principal es convertir los datos serie a paralelo cuando se trata de datos recibidos (de entrada) y de transformar datos paralelos a serie para transmisión (de salida).

La sincronización en el traspaso de datos se lleva a cabo colocando en primer lugar un bit de comienzo (start bit), para luego seguir enviando la trama (data bits) usualmente entre 5 a 9 bits empezando siempre por el bit menos significativo y por último el bit de parada (stop bit) como lo muestra la Figura 4-1. Los UART's que trabajan con 8 bits (como es el caso del RSP1) añaden un

bit de detección de error o bit de paridad. Esto se realiza secuencialmente hasta transmitir la trama [30].

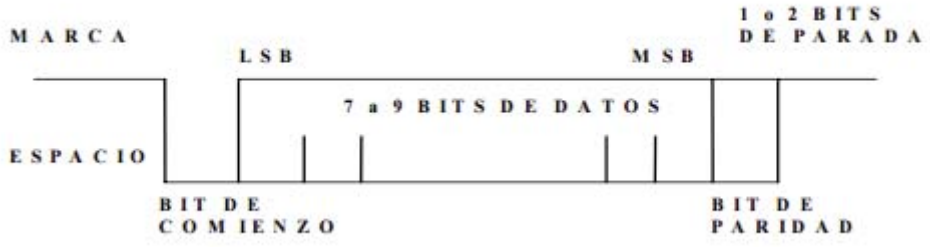


Figura 4-1 Transmisión de datos UART (fuente: www.el.uma.es)

Para hacer la comunicación entre el RSP1 y el Arduino, se debió estudiar la trama de dato en el conector del RSP1, conocido como X7a [23], tanto en la transmisión como en la recepción de este [30]:

- Recepción: Al enviar el comando $L00, por el software RSP_TERMINAL, la señal se encuentra compuesta por el envío de un byte y este se formaba de un bits de inicio, otro de término y la correspondiente trama de datos.
  La Figura 4-2 muestra lo que se vio por el osciloscopio [31]. La primera sección corresponde a la trama de datos de corrido; en la segunda se separan los datos, diferenciando entre bit de inicio y de término por trama, para luego con su respectiva referencia en ASCII, se concluye cual fue el dato recibido por el radar.

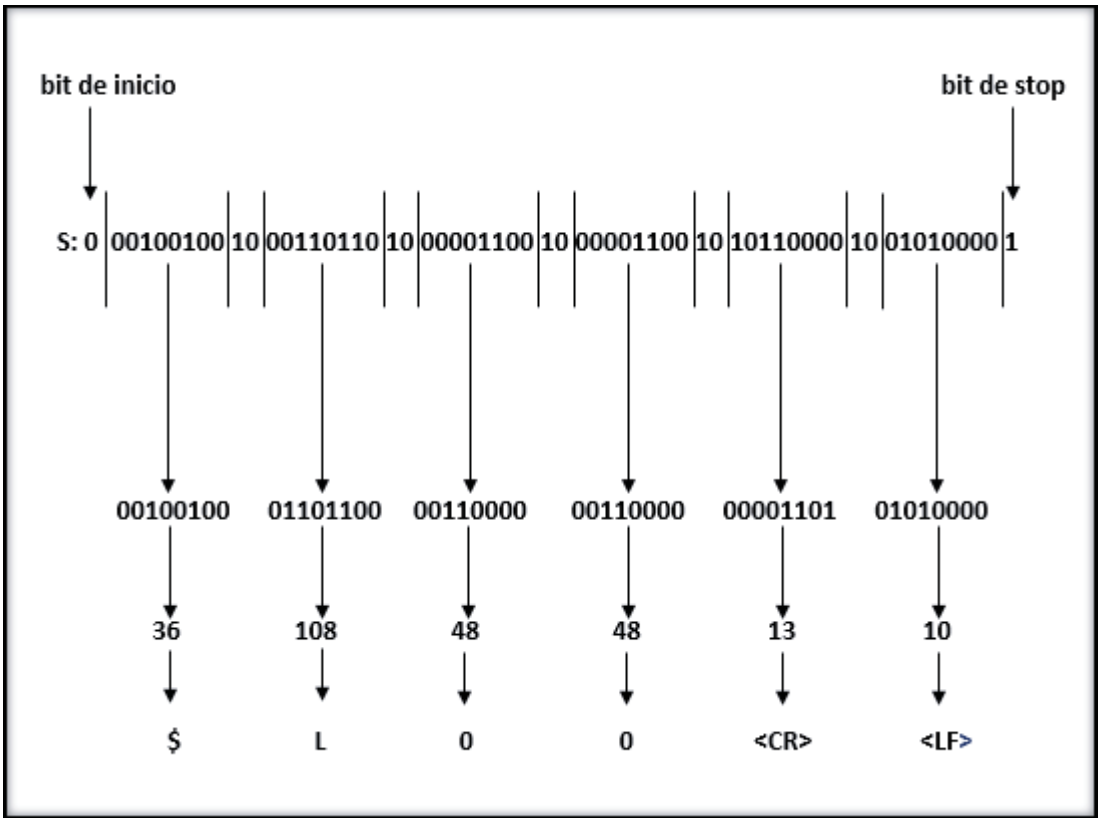


Figura 4-2 comando $L00 mostrado por osciloscopio.

- Transmisión: En el caso la transmisión del RSP1, se envía más información debido a que está transmitiendo la velocidad a la que se acerca, aleja el radar y además la potencia para los mismos casos [23].
  Si el radar se encuentra estático y sin movimiento de frente, entregará una data de: "000;000;000;000" dando un resultado de 15 bytes, más otros 2 para los caracteres "line-feed" y "corriege-return". Siendo importantes los que corresponden a la velocidad y potencia de cercanía o lejanía [28].

4.1.2 Conexión física entre RSP1 y Arduino

Para la conexión entre ambos elementos, se debieron superar los siguientes obstáculos:

- El primer problema corresponde a que los pines de los dispositivos funcionan con diferentes voltajes. Como solución a esto, se crea un divisor de voltaje en los puertos del Arduino para reducir el voltaje de 5(V) a 3.3(V) correspondiente a los que recibe el RSP1.
- El segundo problema es conectar el Rx y Tx del Arduino directamente (con el puente de resistores) al RSP1. Existe un problema que se genera puesto que los puertos del Arduino también son usados para la comunicación con el computador. Es debido a esto que mediante una librería de Arduino, se configuran los puertos digitales 10 y 11 como Tx y Rx y así usar aquellos para recepción y transmisión entre el Arduino y el RSP1, dejando

los que trae Arduino por defecto para la comunicación con el computador. La Figura 4-3 muestra la conexión física completa.

- El último inconveniente fue la alimentación del RSP1, lo que se resolvió alimentándolo con la salida de 5(V) que trae el Arduino.

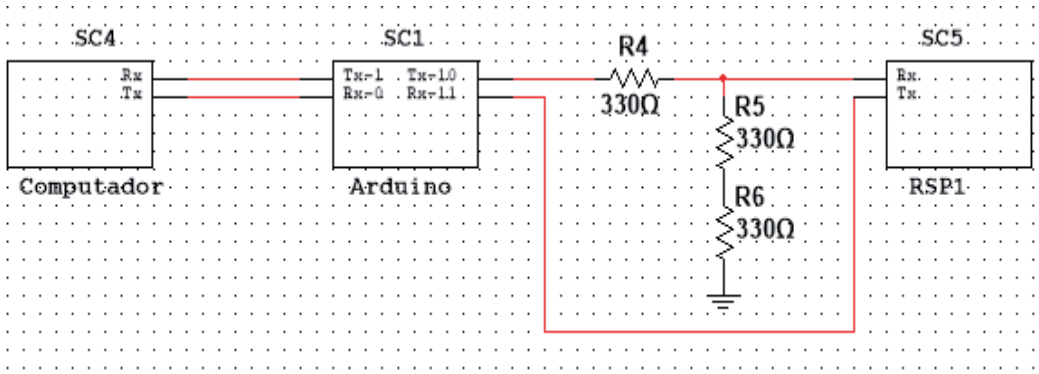


Figura 4-3 Conexión final entre RSP1 y Arduino.

Se puede notar que el receptor del arduino y el transmisor del RSP1 están conectados de forma directa. Esto ocurre ya que el receptor del arduino se encuentra "esperando" algún pulso de datos y al instante que lo recibe este sigue a la señal hasta el voltaje peak. En la Figura 4-4, se muestra que la señal del arduino no alcanza en su totalidad a la del RSP1, ya que la prueba se realizó con un puente de resistores, disminuyendo el voltaje a un punto en que el arduino la reconoce como un "cero". En el momento en que se hizo la prueba directamente sin el puente de resistores, el Arduino tomaba aquellos datos y podía discriminando entre un "uno" y un "cero".

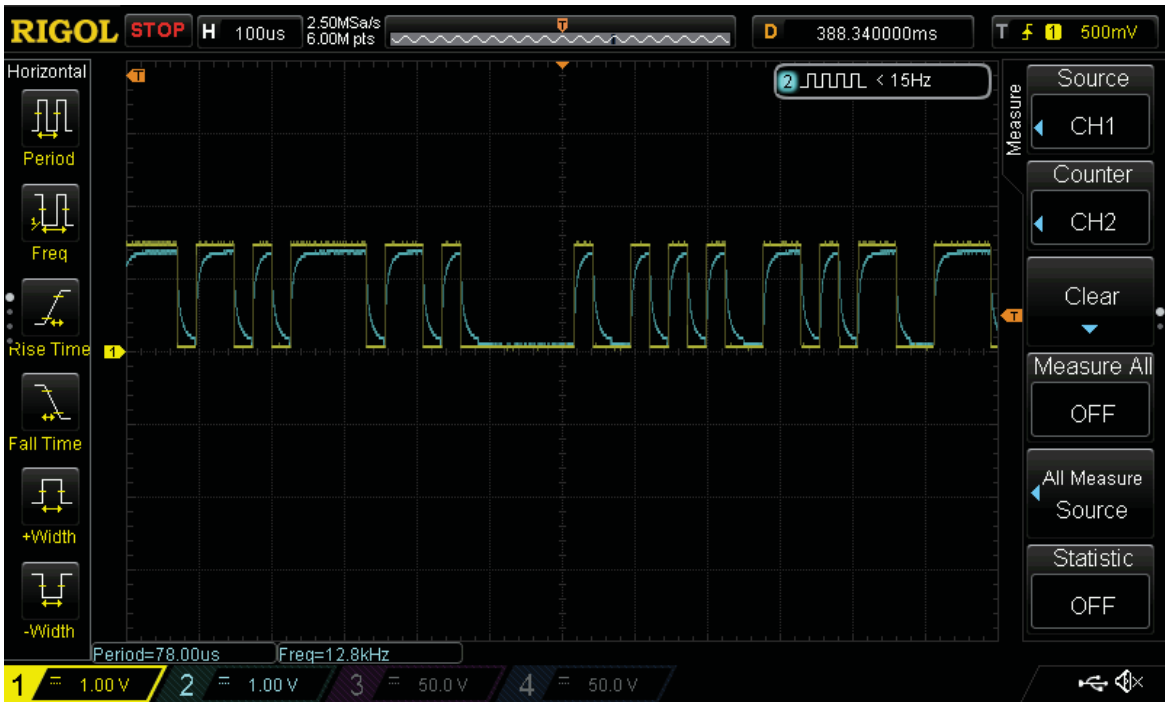


Figura 4-4 Seguimiento de Rx del Arduino de la señal de Tx de RSP1

4.1.3 Comunicación digital

La comunicación digital entre ambos dispositivos se hizo en diferentes partes:

Para habilitar la comunicación se usó un código simple explicado en el diagrama de bloques en la Figura 4-5, que consiguió la comunicación entre ambos dispositivos y además se usó el monitor serie del software de Arduino, para enviar los comandos a RSP1 y ver las respuestas como se muestra en la Figura 4-6 [31].

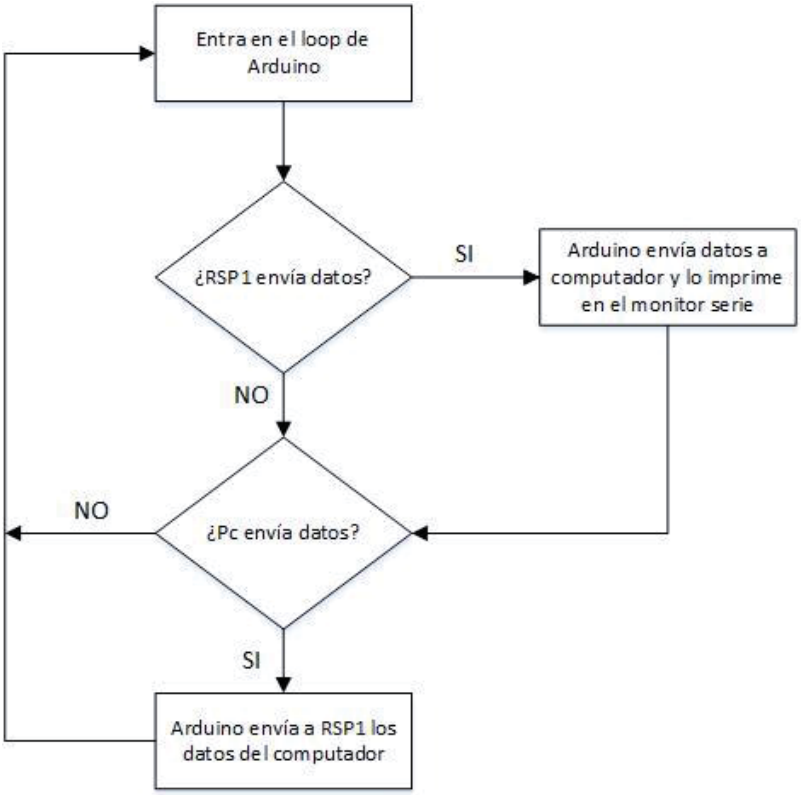


Figura 4-5 Diagrama para habilitar comunicación [31].

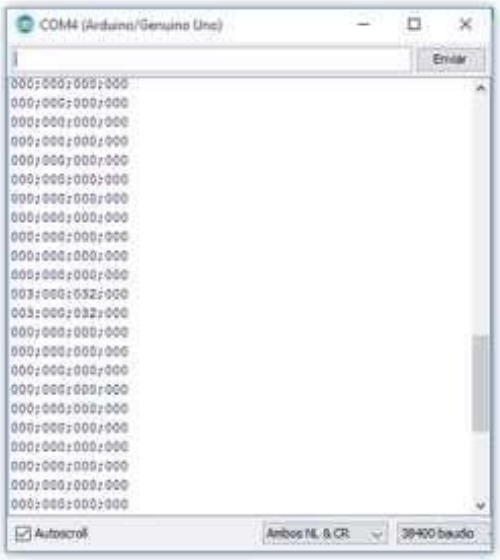


Figura 4-6 Monitor serie [31]

Para poder manipular la trama de datos que se mostró anteriormente, se necesita de la extracción de los datos desde RSP1 ya que hasta ese punto, sólo se observan por el monitor serial.

Para entender el código, es importante recordar que esto es una cadena de datos. Por ejemplo, la línea de datos: 123;456;123;456 son enviados como lo muestra la Figura 4-7 [23].

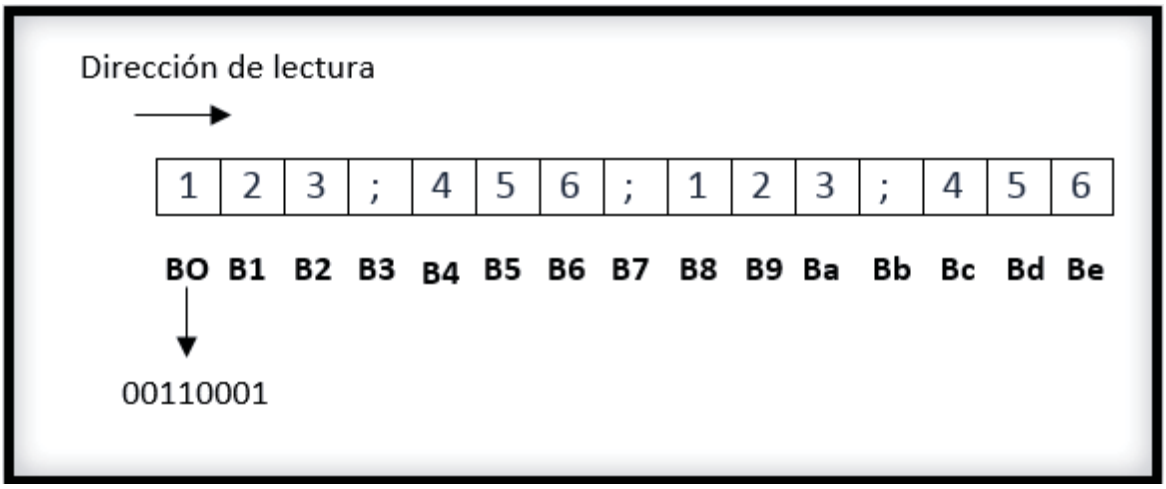


Figura 4-7 Datos entregados por RSP1 [31].

La Figura 4-8 ilustra en un diagrama de bloques la obtención de los datos de la trama y además la asignación a variables para poder manipular estos datos en el Arduino.

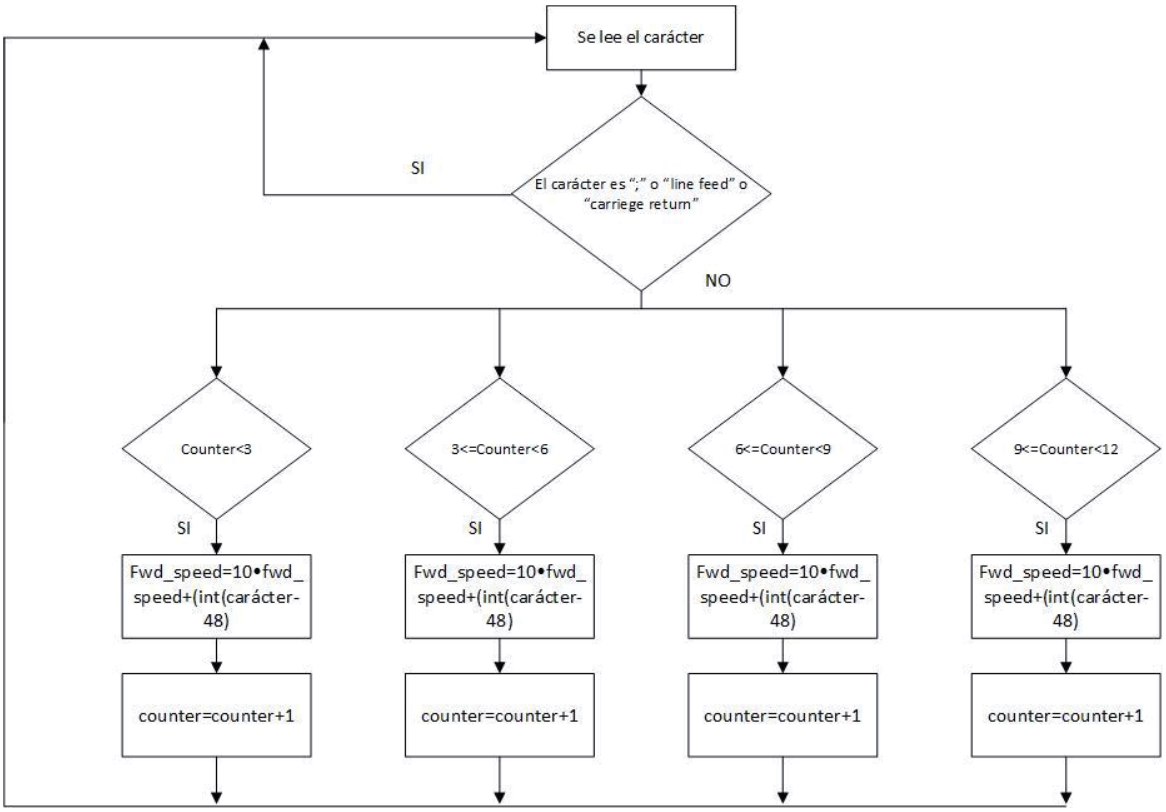


Figura 4-8 Diagrama de bloques para la toma de datos [31].

Debido a la importancia del parámetro S03, se crea un código para la obtención de este. Para obtenerlo, el Arduino debe enviar el comando $S03, por lo que el RSP1 enviará como respuesta aquel valor. La respuesta del radar es @S030X, tomando el valor X entre 0 y A. Por lo tanto, se debe leer el valor de "X", deducir el valor de la frecuencia de muestreo y su resolución [23].

En la Figura 4-9, el diagrama de bloques explica el funcionamiento de la lectura del parámetro S03.

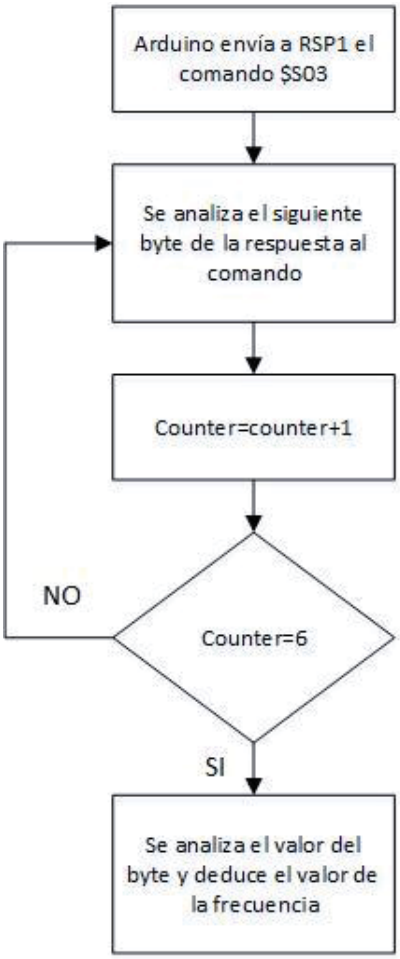


Figura 4-9 Obtención de parámetro S03 [31].

La Figura 4-10 ilustra el funcionamiento de todo el programa [31].

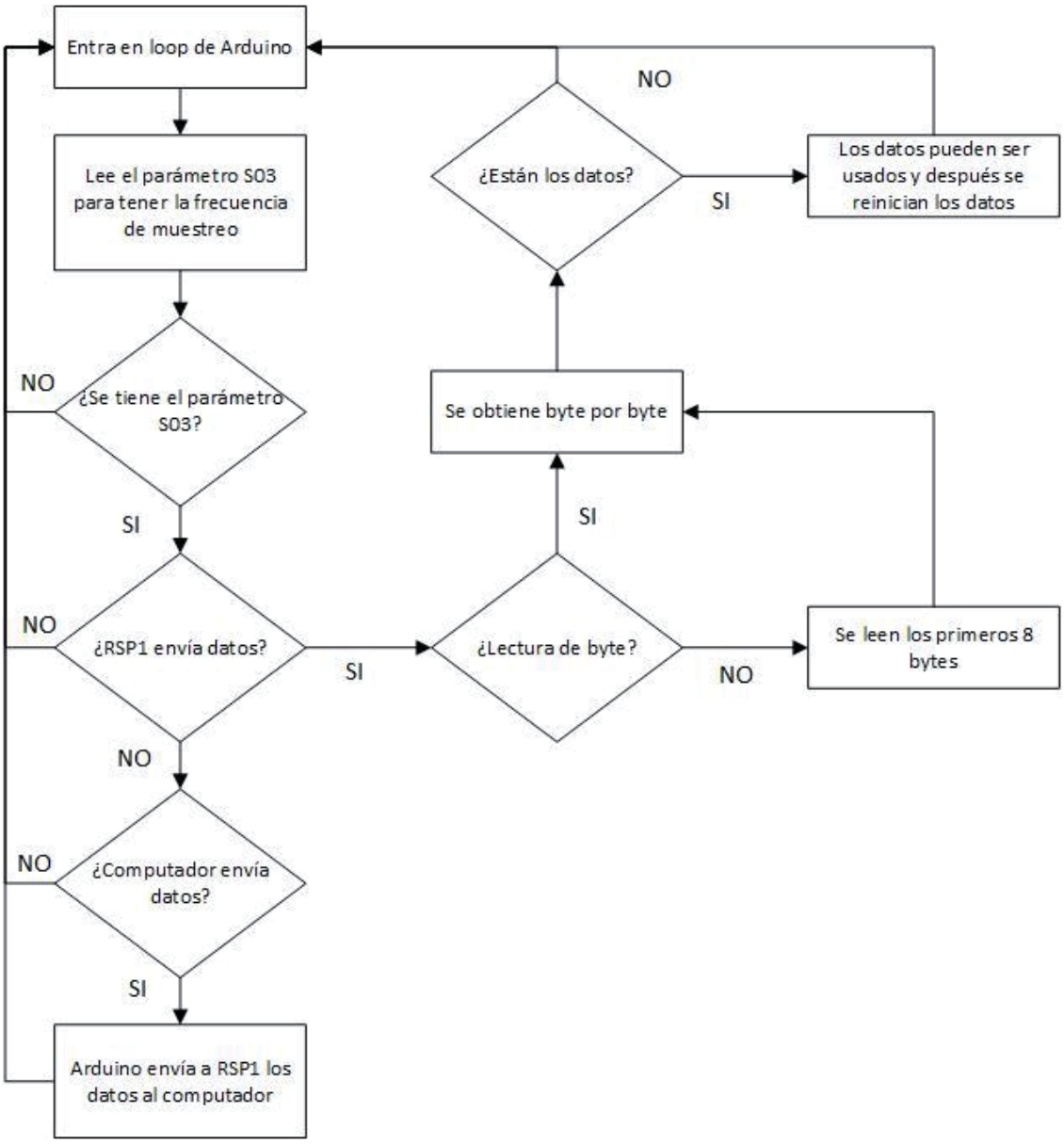


Figura 4-10 Diagrama de programa completo [31].

Se logra captar la cadena de datos entregada por el RSP1, asignar los valores de la trama a variables y con ello poder manipularlas para poder trabajarlas adecuadamente.

4.2 Comunicación Arduino y drone

En este apartado se explica el desarrollo de la comunicación entre el drone y el Arduino, con el fin de establecer una interacción entre el radar y el Drone, siendo este uno de los objetivos finales del proyecto.

4.2.1 Conexión física

La conexión física es similar a la que usó del radar al Arduino. Primero se fijaron pines para establecer una conexión serial entre ambos dispositivos. En este caso, el puerto serial 4/5 del radar, usando solo el puerto 4.

Para el Arduino, se establecieron los pines 12 y 13 como receptor y transmisor respectivamente, tal como lo muestran Figura 4-11 y Figura 4-12.

Figura 4-11 Puerto Serial 4/5 (fuente: www.ardupilot.org).

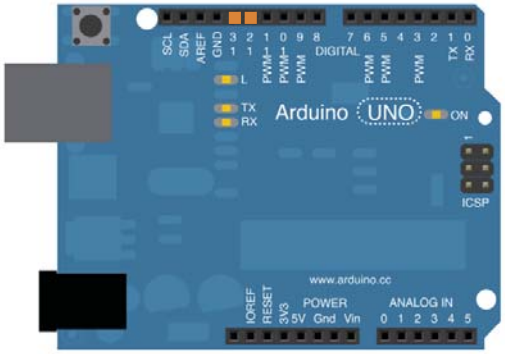


Figura 4-12 Arduino Uno (fuente: www.arduino.cc)

Posteriormente se conectan los dispositivos y para ello se requiere utilizar un puente de resistores en el pin 13 del Arduino, ya que este instrumento transmite a 5(V), pero el drone funciona con 3.3(V) en sus pines. De igual manera, como sucedió anteriormente al caso RSP1-Arduino y el receptor del Arduino, se encuentra a la espera de la señal transmitida por el HKPilot32, conectándose de forma directa. El puente de resistores sólo es utilizado desde el transmisor del Arduino al receptor del HKPilot32, quedando finalmente como lo muestra la Figura 4-13.

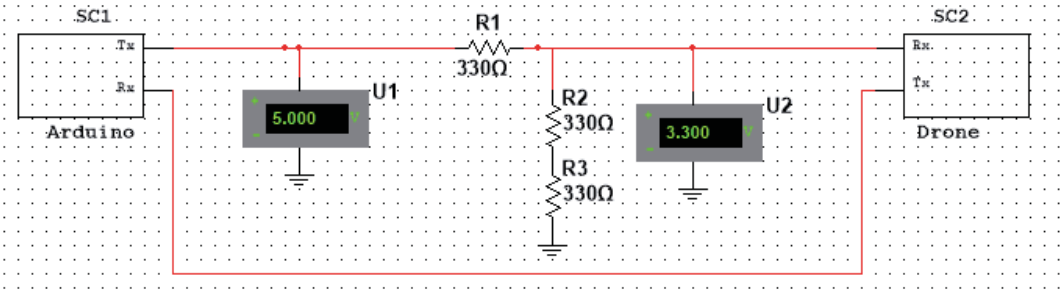


Figura 4-13 Conexión física completa.

4.2.2 Programación

Para ambos casos, la programación consta de la misma forma: la espera de algún dato del otro dispositivo para su lectura y mostrarlo a través del monitor serie tal como lo muestran las Figura 4-14 y Figura 4-15:

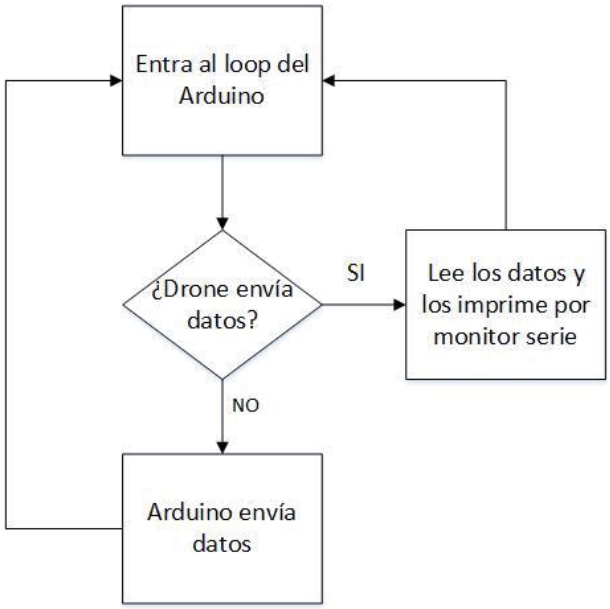


Figura 4-15 Programación de Arduino.

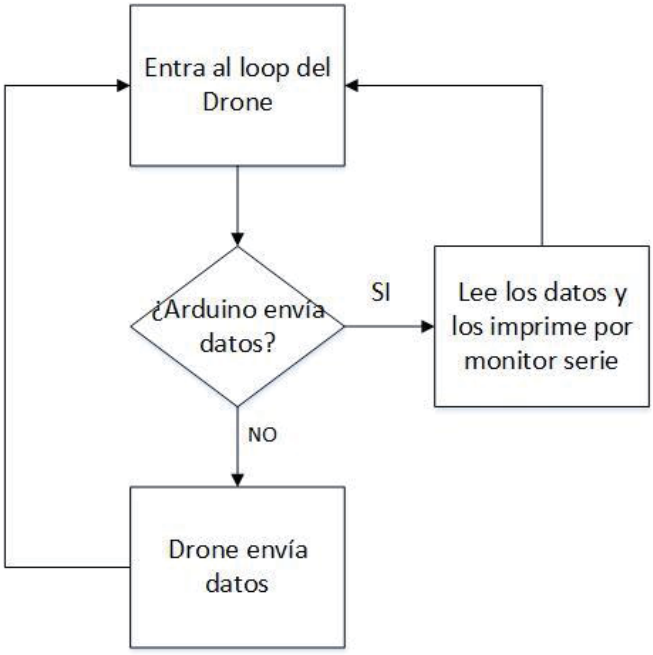


Figura 4-16 Programación en Drone.

El código mostrado en Listado 4-1 es el agregado y cargado en la placa del Drone. Este consta de la lectura de datos en caso de la recepción de alguna trama. Para del Arduino se usó uno de los "sketch" que ya vienen incorporados dentro del software, mostrando lo más importante en el Listado 4-2.

Listado 4-1 Código de lectura de datos en Drone.

```
1  While(hal.uartE->available())
2  {
3  Hal.console->write(hal.uartE->read());
4  }
```

Listado 4-2 Código de Arduino.

```
1  Void loop()
2  {
3  If(mySerial.available())
4          {
5          Serial.write(mySerial.read());
6          mySerial.printf("hola drone");
7           }
8  }
```

Los resultados obtenidos se muestran en la Figura 4-17.

Figura 4-17 Lectura de datos enviados por Arduino.

Luego de mostrar la comunicación entre el Drone y el Arduino, la siguiente etapa consta en establecer la comunicación entre el sistema completo, para posteriormente hacer que el Drone envíe una señal una vez cumplido un determinado tiempo.

5 Integración del sistema

En este capítulo se unen todos los elementos del proyecto y se habla sobre la comunicación entre estos y los resultados finales.

5.1 Sistema completo

Para poder llevar a cabo la finalidad del proyecto, el sistema debe constar de tres elementos: el radar Doppler RSP1, un arduino uno y un drone. Como etapa intermedia entre el radar y el Arduino existe un divisor de tensión hecho de un juego de resistencias. Lo mismo sucede entre el arduino y el Drone, como ya se mencionó en el capítulo anterior. La Figura 5-1 muestra cómo el sistema queda estructurado de forma completa.

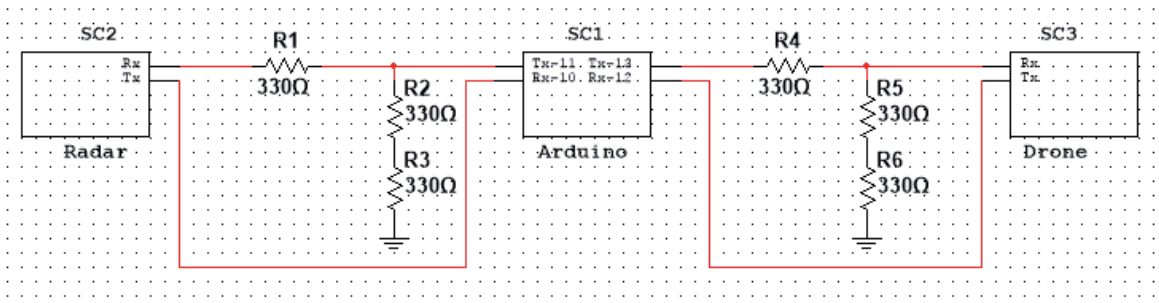


Figura 5-1 Sistema ilustrado completo.

5.2 Diagrama esquemático

En el diagrama de flujo mostrado en la Figura 5-2, se exhibe de forma segmentada los pasos que sigue el algoritmo utilizado en el sistema completo para la detección y posterior aviso de algún obstáculo próximo.

Se inicia con la detección de algún obstáculo frente al Drone. En caso que exista, comienza un contador que a los 5 segundos envía un "1" al Drone para indicarle que hay un estorbo en su ruta y en caso de que el Drone vaya en vuelo enviará un "stop" en forma de audio por lo que, quien lleve el control en ese instante, se entere de una posible colisión.

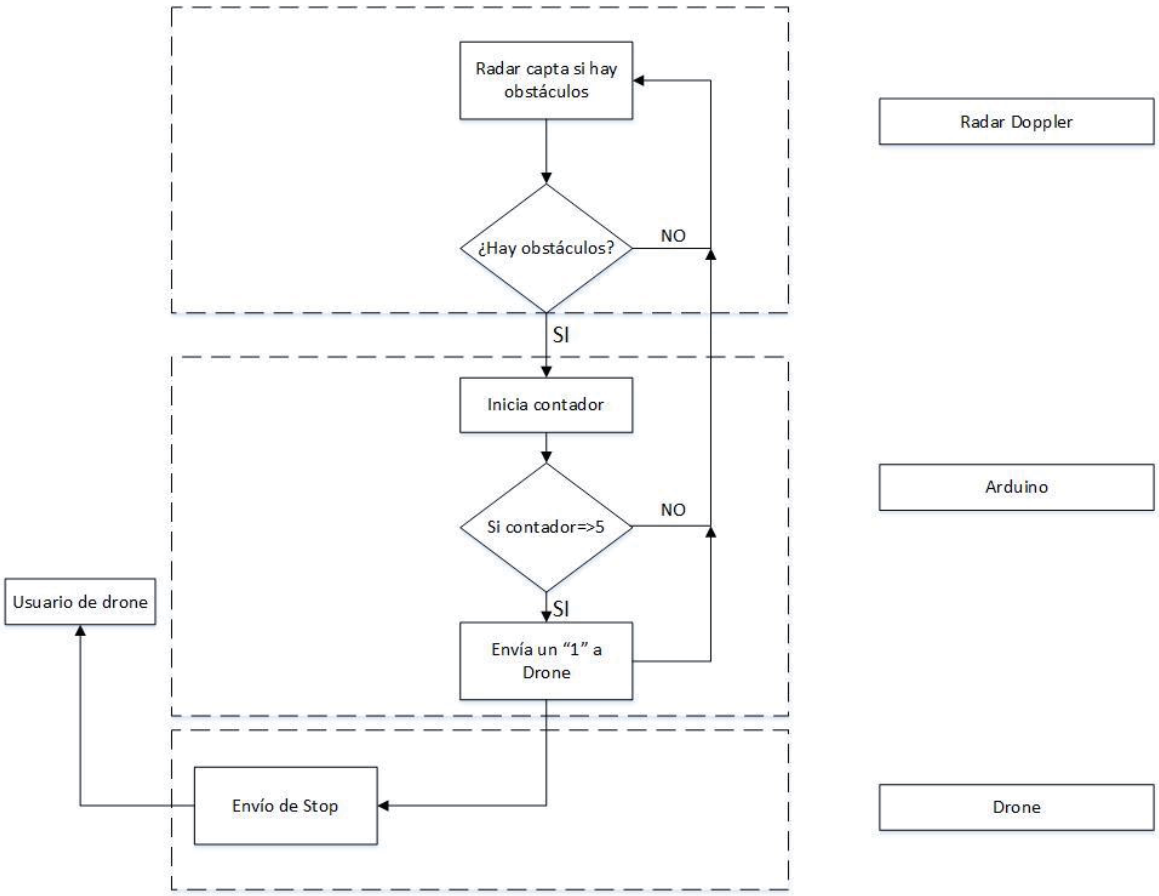


Figura 5-2 Diagrama de flujo de sistema completo.

5.3 Programación

Para la programación se hicieron algunas leves modificaciones a los códigos que antes se habían mostrado.

El código completo en el arduino se encuentra en el apéndice A.

Cabe destacar que se necesita sentenciar de forma correcta los "write" para que el envío de datos sea correcto y "read" para la lectura de los datos recibidos [32].

5.3.1 Arduino

Respecto a la programación del Arduino, se inicia asignando dos nuevas variables en el código, una para poseer el tiempo fijo que debe esperar y la otra irá guardando el tiempo de referencia. En este caso es 0 (ms) como muestra el Listado 5-1 Asignación de variables.

Listado 5-1 Asignación de variables.

```
1  Unsigned long interval=5000; //El tiempo que se necesita esperar
2  Unsigned long previousMillis=0;
```

La segunda parte y final, consta de una variable de nombre "currentMillis". Esta guarda el tiempo que va transcurriendo mientras lo compara con su referencia. Una vez que cumple la condición de ser mayor al tiempo de espera, ejecuta la acción de enviar el 1 para que el Drone lo reconozca y realice la acción de aviso correspondiente, como lo muestra el Listado 5-2.

Listado 5-2 Cuerpo de programa modificado.

```
1  Unsigned long currentMillis = millis(); //Graba el tiempo que transcurre
2  //observa si el tiempo ha pasado (5000 milisegundos)
3  If((unsigned long)(currentMillis – previousMillis)>=interval)
4  {
5  radarSerial.print("1");
6  //Guarda el tiempo transcurrido
7  previousMillis=millis();
8  }
9  Unsigned long previousMillis=0;
```

La explicación anterior se muestra en Figura 5-3.

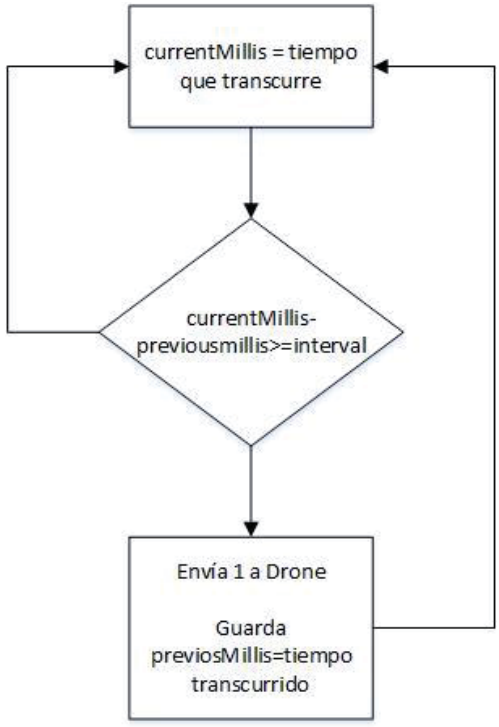


Figura 5-3 Código agregado al de Arduino.

5.3.2 Drone

El proceso de espera y conteo para un obstáculo se encuentra en el radar y el Arduino, sin embargo, el drone sólo debe estar a la espera de que el tiempo "x" transcurra. Una vez que este tiempo transcurre y el Arduino envía el "1", el drone debe tomar una decisión, puesto que la

función de este es esperar, reconocer y responder a aquella entrada notificadora. El Listado 5-3, muestra el código para realizar esta operación.

En primera instancia, se asigna un "baudrate" con el que trabaja el radar, para luego evaluar si existe algún dato de entrada y en caso de que haya, se lee e imprime a través del monitor serie (para verificar si existe comunicación). Y termina en el envío de un mensaje de audio "stop", por la interfaz gráfica.

La Figura 5-4 muestra de forma gráfica la explicación mencionada con anterioridad.

Listado 5-3 Código agregado en Drone.

```
1  Hal.console->begin(38400);
2  Hal.uartE->begin(38400);
3  While(hal.uartE->available())
4  {
5  Hal.console->write(hal.uartE->read());
6  gcs_send_text(MAV_SEVERITY_CRITICAL,"PreArm: stop");
7  }
```

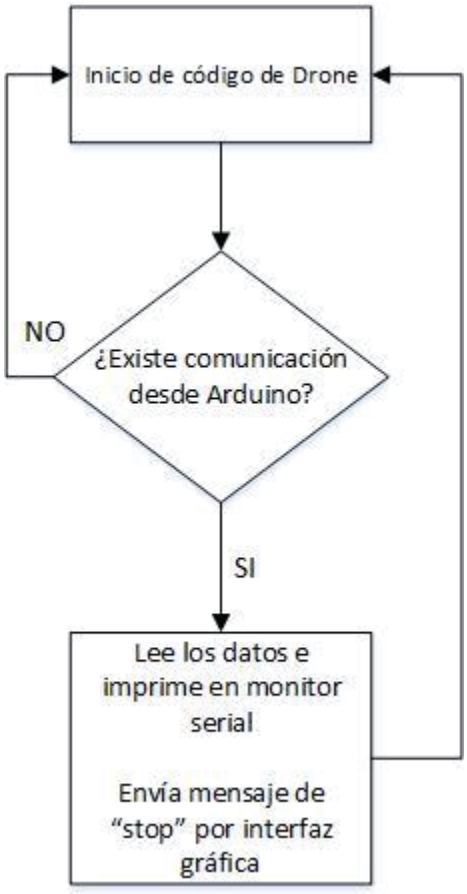


Figura 5-4 Diagrama agregado a código de Drone.

5.4 Resultados

Como primer resultado, se logra destacar la comunicación entre los tres elementos principales del proyecto, pudiendo enviar información necesaria desde el radar, procesarla a través del Arduino y enviarla al Drone.

Además se corrobora el envío del aviso de parte del Arduino que ha transcurrido en tiempo "x" mediante el monitor serial, como lo muestra la Figura 5-5. Esta cadena de "1" es la recibida y leída por el drone.

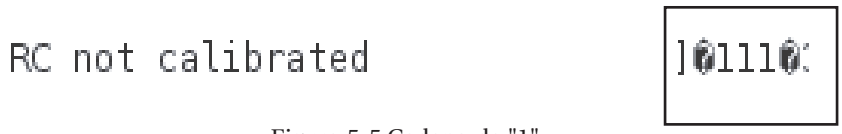


Figura 5-5 Cadena de "1".

Una vez comprobado el aviso desde el Arduino al Drone, el siguiente paso es comprobar la respuesta de este. Sin embargo, este paso no fue posible realizarlo por un motivo en específico que se explica a continuación.

El algoritmo del HKPilot32 al iniciar el programa de APMPlanner, da pie una serie de etapas previas. Dentro de estas se encuentran las calibraciones correspondientes para que el "cerebro" pueda tener una referencia para poder desplazarse. Una de estas calibraciones corresponde a la del radio control. Debido a la ausencia de uno, se consiguió un radio control Fly Sky modelo FSTH9X, mostrado en Figura 5-6. Sin embargo, para poder realizar esta etapa se carecía de otro elemento: un Conversor de PWM a PPM [29] como lo muestra Figura 5-7. Este no fue posible conseguir, debido a no se pudieron resolver todas las problemáticas que se plantearon al desarrollar los objetivos del proyecto.

Figura 5-6 Fly Sky, FS-TH9X (fuente: www.banggod.com).

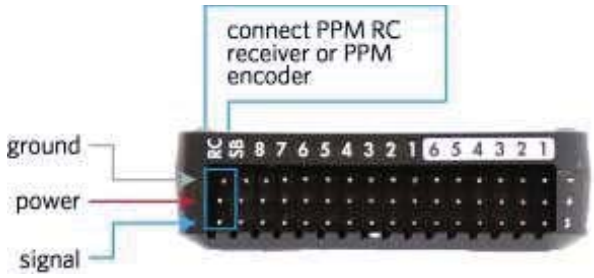


Figura 5-7 Conexión de conversor (fuente: PPM www.ardupilot.org).

Debido a esto, en vez de hacer las pruebas en simulaciones de vuelo, se realizaron en la pantalla principal que entrega el software de APMPlanner mostrada en la Figura 5-8. Dicho lo anterior, se puede adelantar que se obtuvieron resultados positivos del código, dando aviso de "stop" una vez que transcurre un tiempo determinado. Para realizar las pruebas se escogió un tiempo de 5(s). La Figura 5-9 muestra cómo se envía por puerto serial el mensaje emitido al software APMPlanner.

Figura 5-8 Pantalla inicial APMPlanner

Figura 5-9 Mensaje de stop en monitor serial

Discusión y conclusiones

En este proyecto de titulación se ha dado una solución a una de las principales problemáticas a resolver en los vuelos automáticos de los vehículos no tripulados. Dicho problema es la posible colisión con algún obstáculo que se encuentre dentro de la ruta de vuelo del Drone. Es debido a eso que se propone como solución tener una notificación en formato audio en caso de que exista algún obstáculo en su ruta de vuelo.

Para llegar a la solución planteada se estudió sobre algunas empresas, investigaciones, proyectos, con el fin de averiguar si algún sistema existente ha implementado la autonomía en el vuelo. Se encontró que la empresa DJI Phantom con su último drone sacado al mercado logró una autonomía total en los vuelos, sin embargo, se guía mediante cámaras y sonares siendo dependiente del campo visual que pueda tener. Para evitar esto se implementó un radar Doppler de onda continua de la empresa RFBeam modelo RSP1 con las propiedades de poder despreciar la vibración que pueda tener el Drone debido a la naturalidad de su movimiento. Otra propiedad importante de este radar es ser I&Q, que sirve para poder obtener un resultado detallado sobre la velocidad a la cual el obstáculo frente al radar se esté moviendo o dicho de otra forma, a la velocidad que el radar verá que se mueve.

Un elemento importante en este radar son sus puertos seriales, principalmente el X7a debido a que posee pines de transmisor, receptor, alimentación y la correspondiente conexión tierra, además el protocolo usado por este puerto es el "UART" y con ello se puede establecer una comunicación.

Para realizar la comunicación fue necesario interpretar la trama de datos entregada por el radar, y con ello poder manipular la cadena de información y tener el control de los datos que entrega. Dicha interpretación y obtención de datos la hizo un estudiante francés en práctica de nombre Grégoire Dery quien se encargó del desarrollo del código para procesar la información entregada por el radar mediante un arduino uno. Para esto debió analizar la trama de datos otorgada por el radar mediante un osciloscopio, para que una vez entendido como se componía aquella trama, descomponerla en el arduino y obtener un control total de los datos entregados. Es debido a lo anterior que el tratamiento de la señal no la realiza el drone si no que el propio arduino.

Además se estudió puertos del drone. Para poder establecer una interacción con los puertos, se les asigna un protocolo de comunicación dependiendo del fin que posean dentro del mismo

código. Por ejemplo los puertos que debían tener una comunicación con instrumentos que funcionaran con radio frecuencias establecen el protocolo de comunicación MAVLink, por otro lado los que debían tener una comunicación serial el protocolo UART. En vista de esto se usó un puerto que era confinado para la comunicación serial, pero se debió ajustar la velocidad de baudios para que trabajara de la misma forma a la que se configuró el Arduino.

Para obtener control sobre el vuelo del drone se modificó levemente el código en el arduino. Para que cuando un obstáculo sea detectado y transcurra un determinado tiempo, el algoritmo envíe un aviso al usuario sobre una posible colisión.

Por otro lado se debe considerar la conexión física, debido a que no siempre se trabaja con las mismas tecnologías y por ello puede haber diferencias en sus voltajes, corrientes, potencias necesarias para poder trabajar de forma óptima. En este caso las tecnologías eran distintas, trabajando con TTL y con CMOS, existiendo una diferencia de voltaje en los pines de los elementos, para evitar problemas se debió realizar un puente de resistores para que nada se viera en riesgo de quemar algún elemento del proyecto y se pudiera trabajar de forma segura.

Para poder manipular el drone se debió trabajar con el sistema operativo Linux, debido a las librerías que se requieren para poder descargar todo el código desde la plataforma de librería abierta GitHub no se encuentran habilitadas en Windows 8.1, diferentes es el caso de Windows 10 en donde se puede descargar una máquina virtual que tenga todo lo necesario. Otro punto es el del complemento que debieron tener los softwares que se usaron. Tales como: QTCreator, APMPlanner, Arduino IDE y la ventana terminal del sistema operativo Linux. Cada uno con una función muy específica, QTCreator para visualizar el código de HKPilot32 y las modificaciones, APMPlanner para ver los cambios realizados en la interfaz gráfica de HKPilot32, Arduino IDE para tener acceso a los puertos seriales del mismo y la ventana terminal para compilar y cargar los cambios realizados al código del HKPilot32.

Además hay detalles dentro del proyecto que se deben mencionar de forma aparte debido a su importancia. El primer punto habla sobre una de las opciones que entrega el control creado para evitar colisiones y el segundo trata la falta de un conversor de modulación derivando así, en un desfase de tiempo en las pruebas:

- El tiempo de espera que el obstáculo permanece en la ruta de vuelo del drone antes de enviar el aviso, puede ser modificado en la programación del Arduino de acuerdo a lo que uno estime conveniente. Por ejemplo, si se desean vuelos en modos rápidos, en el código se puede modificar el valor “interval” para que el tiempo que deba tener un obstáculo frente sea menor y en el caso de vuelos que se requieran menos velocidades, puedan ser tiempos mayores de espera.
- Un inconveniente fue la falta de un instrumento que necesita el HKPilot32 para la realización completa de su etapa de calibración, dicho instrumento es un conversor PWM a PPM. Por ese motivo cuando el código del drone se ejecuta, permanece constantemente en el loop de calibración haciendo que las pruebas dependan del orden del código de calibración, teniendo que esperar a que se ejecuten las sentencias agregadas para la

notificación del obstáculo en la ruta del drone, agregando un desfase de tiempo en las pruebas.

Por otro lado existen detalles que no se pudieron cubrir:

- No se pudo realizar pruebas en simulaciones de vuelos o con el radar montado en un Drone que esté volando por la falta del conversor PWM a PPM.
- Otro es el de no haber podido transformar los datos entregados por el RSP1 a distancias, por motivos de falta de información del mismo.

A pesar de los detalles que no se lograron cubrir se logró realizar un control, donde al usuario se le avisa del riesgo de una posible colisión. Por otro lado, este proyecto puede ser el punto de partida para muchos otros trabajos a realizar, tales como un control del vuelo mediante cámaras y sonares como en el caso de DJI Phantom, poder combinar el control visual con el del radar, hacer que la comunicación entre el Arduino y el Drone sea de forma inalámbrica y por último sería interesante hacer un trabajo de investigación para comparar todos los sistemas y destacar cual es mejor aplicar en determinados casos y cual no, poder discriminar cual consume más recursos, que sistema es más rápido que otro, entre otros.

Bibliografía

[1] Segunda guerra mundial, «Segunda Guerra Mundial,» Segunda Guerra mundial, Enero 2000. [En línea]. Available: http://segundaguerramundial.es/es/. [Último acceso: 15 Abril 2016].

[2] El diario, «El diario,» Impremedia, Septiembre 2017. [En línea]. Available: http://eldiariony.com/. [Último acceso: 15 Abril 2016].

[3] A. J. barry, «High-Speed Autonomous Obstacle Avoidance with Pushbroom Stereo,» MASSACHUSETTS, 2016.

[4] IRTIC. Institut Universitadi d'investigació de Robóbica i de Tecnologies de la Informació i Comuincaió, «Drones,» WordPress, Agosto 2001. [En línea]. Available: http://drones.uv.es/. [Último acceso: 19 Abril 2016].

[5] El economista, «eleconomista,» Octubre 2015. [En línea]. Available: http://eleconomista.com.mx. [Último acceso: 20 Abril 2016].

[6] DJI, «dji,» dji, Enero 2000. [En línea]. Available: http://www.ehu.eus/es/web/gipuzkoa. [Último acceso: 11 Abril 2016].

[7] Definicion, «Deficion,» WordPress, Mayo 2008. [En línea]. Available: http://definicion.de/. [Último acceso: 10 Abril 2016].

[8] F. R. y. G. Coate, Principles of Radar, MIT Radar School Staff, 1952.

[9] W. Melvin, Principles of modern radar: advanced techiques, Stevenage, 2012.

[10] D. T. &. W. J. M. Victor C. Chen, Radar Micro-Doppler Signature, The Institution of Engineering and Technology, 2014.

[11] A. D. M. &. M. S. Greco, Mdern Radar Detection Theory, Scitech Publishing, 2016.

[12] J. Penalva, «xataka,» weblogsSL, Febrero 2004. [En línea]. Available: www.xataka.com. [Último acceso: 10 Abril 2016].

[13] P. A. Pereyra, «Reconocimiento Facial Mediante Imágenes Esteoscópicas para control de Ingreso,» Buenos Aires, 2005.

[14] TODRONE, «TODRONE,» BRAND SPEAKER, FEBRERO 2015. [En línea]. [Último acceso: 11 ABRIL 2016].

[15] El pais, «Elpais,» PRISA, Enero 2000. [En línea]. Available: www.elpais.com. [Último acceso: 11 Abril 2016].

[16] A. Giusti, «A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots,» Texas, 2015.

[17] C. Alabaster, Pulse Doppler Radar, Cranfield, 2012.

[18] J. F. C. Rico, «ESTUDIO Y CARACTERIZACIÓN DE ECOS DE RADAR DE ONDA CONTINUA APLICADOS A LA MEDICIÓN DE NIVEL DE FLUIDOS HOMOGÉNEOS Y HETEROGÉNEOS,» Bogotá, 2012.

[19] D. V. González, «Detección de movimento mediante ténicas radar CW-FM en banda W,» Barcelona, 2014.

[20] organismo especializado de las Naciones Unidas para las Tecnologías de la Información y la Comunicación, «itu,» uit, Marzo 2017. [En línea]. Available: https://www.itu.int. [Último acceso: 11 Abril 2016].

[21] F. Pizarro, «Sistemas de radiofrecuencia y microondas. Tema II: Antenas y propagación,» 2015.

[22] M. Otero, «Application of a continuous wave radar for human gait recognition,» Bedford, 2007.

[23] RFBEAM, *RSP1 Datasheet,* RFbeam Microwave GmbH.

[24] ARDUPILOT, «ArduPilot,» ArduPilot, Febrero 2016. [En línea]. Available: http://ardupilot.org/ardupilot/index.html. [Último acceso: 14 Junio 2016].

[25] C. Wolff, «radar tutorial,» Creative Commons Attribution-Share, 1998 November. [En línea]. Available: http://www.radartutorial.eu/. [Último acceso: 11 May 2016].

[26] B. R. Mahafza, Radar Systems analysis and Design Using MATLAB, CHAPMAN & HALL/CRC.

[27] National Instruments, «NI,» National Instruments, November 2007. [En línea]. Available: http://www.ni.com/. [Último acceso: 11 May 2016].

[28] RFBEAM, *RSP1 Evaluation Kit,* RFBEAM.

[29] ArduPilot, «ArduPilot,» ArduPilot, Febrero 2016. [En línea]. Available: http://ardupilot.org/dev/docs/learning-the-ardupilot-codebase.html. [Último acceso: 11 Julio 2016].

[30] Universidad de Malaga, «Universidad de Malaga: Departamento de Electrónica,» Departamento de Electrónica, Enero 2012. [En línea]. Available: http://www.el.uma.es/marin/Practica4_UART.pdf. [Último acceso: 3 Agosto 2016].

[31] G. Dery, «Rapport the stage,» Pontificia Universidad Católica de Valparaíso, Valparaíso, 2016.

[32] Arduino, «Arduino,» Arduino, Junio 2017. [En línea]. Available: https://www.arduino.cc/. [Último acceso: 20 Julio 2016].

[33] Opinno, «technologyreview,» technology review, Abril 2013. [En línea]. Available: https://www.technologyreview.es/informatica/42965/aprendizaje-profundo/. [Último acceso: 11 Abril 2016].

[34] P. H. Edde, Introduction to Airborne Radar, 1993.

A Un apéndice

Código que se hizo en Arduino para la realizaci0ón del proyecto.

A.1 Archivo principal

Listado A-1 Código principal en Arduino

```
#include <SoftwareSerial.h>
#include "RSP1_function.h"


/* serial UART port of RSP can't be plugged to those of Arduino.
They are already use for the USB communication with computer*/

SoftwareSerial rspSerial(10,11);
// Rx : pin 10
// Tx : pin 11

SoftwareSerial radarSerial(12,13);
int speed_power[4];
int counter;
int frequency_sample;

float resolution;

boolean passage;
boolean data_ready;
boolean lect_S03;
```

```
boolean isNotWritten_S03;
char charact;
unsigned long interval=5000; // the time we need to wait
unsigned long previousMillis=0; // millis() returns an unsigned long.

void setup()
{
 //serial UART ports initialisation
 Serial.begin(38400);
 radarSerial.begin(38400);
 rspSerial.begin(38400);


 data_ready = false;
 passage = true;
 lect_S03 = true;
 isNotWritten_S03 = true;

 frequency_sample = 0;
 resolution = 0;
 charact = 0;

 init(speed_power,&counter);
}

void loop()
{

  if(lect_S03)
  {
      write_inRSP("$S03", &rspSerial, &isNotWritten_S03);
      gettingS03Values(&charact, &frequency_sample, &resolution, &lect_S03,
&counter, &rspSerial);
  }
  else
  {
      if(rspSerial.available()) //rspSerial.available() is true when the RSP1 send
information
      {
            if(passage==true) //The first 8 bytes are an answer of the command
we sent (for example $L00 -> @L0000 and then, the RSP1 send the values)
            {
              delay(10);
              for(int i=1;i<=8;i++)
              {
               Serial.write(rspSerial.read());
              }
              passage = false;
            }
```

```
            charact = rspSerial.read(); //We read one caractere by loop of the
arduino
            speed_power_reading(speed_power,charact,&counter);  //This
function  is to put the value send by the RSP1 in variables so as we can use
these value
            data_ready = (counter==12);  //When couter==12,  the datas sent by
the RSP1 are in variable and then can be used by the user


            if(data_ready)  //It is in this 'if' that all the manipulations with the
speed and the power can be done.
            {

 /*******ALL  THE MANIPULATION  OF THE VALUE  HERE !!!*******/

            Serial.print("fwd speed : ");
            Serial.print(speed_power[0]);

            Serial.print("   fwd speed :");
            Serial.print(speed_power[0]*resolution);
            if(speed_power[0]*resolution>0)
            {
            //radarSerial.print(speed_power[0]*resolution);
            unsigned long currentMillis = millis(); // grab current time
            // check if "interval" time has passed (1000 milliseconds)
             if ((unsigned long)(currentMillis  - previousMillis) >= interval)
             {
             radarSerial.print("1");//send  "1" to Drone
             // save the "current"  time
             previousMillis = millis();
            }
            }
            Serial.print("   Bwd speed :");
            Serial.print(speed_power[1]);
            Serial.print("   fwd power :");
            Serial.print(speed_power[2]);
            Serial.print("   Bwd power :");
            Serial.println(speed_power[3]);
init(speed_power,&counter);  //Do not remove this line. It initialise the value
to allow the update of the datas.
            }
        }
if(Serial.available()) //Serial.available() is true when the computer  sends
information  to arduino.
        {
        rspSerial.write(Serial.read());
        passage=true;
        counter=0;
        }
   }

}
```

A.2 Archivo de funciones

Listado A-2 Código de funciones

```
#include "RSP1_function.h"

void gettingS03Values(char* character, int* frequency, float* resolu, bool*
S03_get, int* count, SoftwareSerial* rsp)
{

if(S03_get)
{
	if (rsp->available()) //rspSerial.available() is true when the RSP1 send
information
	{
		*character = rsp->read();
		*count = *count + 1;
	}
	if (*count > 5) //The only thing we have to read is the last bytes, that
why there is this if here. Without it, we would read the 3 of @S03XX.
	{
switch (int(*character))
		{
		case 49:
			*frequency = 640;
			*resolu = 0.11;
			*S03_get = false;
			rsp->read();
			rsp->read();
			break;

		case 50:
			*frequency = 1280;
			*resolu = 0.23;
			*S03_get = false;
			rsp->read();
			rsp->read();;
			break;

		case 51:
			*frequency = 1920;
			*resolu = 0.34;
			*S03_get = false;
			rsp->read();
			rsp->read();
			break;

		case 52:
			*frequency = 2560;
			*resolu = 0.45;
```

```
                *S03_get = false;
                rsp->read();
                rsp->read();
                break;

        case 53:
                *frequency = 3200;
                *resolu = 0.57;
                *S03_get = false;
                rsp->read();
                rsp->read();
                break;
    case 54:
                *frequency = 3840;
                *resolu = 0.68;
                *S03_get = false;
                rsp->read();
                rsp->read();
                break;
        case 55:
                *frequency = 4480;
                *resolu = 0.80;
                *S03_get = false;
                rsp->read();
                rsp->read();
                break;

        case 56:
                *frequency = 5120;
                *resolu = 0.91;
                *S03_get = false;
                rsp->read();
                rsp->read();
                break;

        case 57:
                *frequency = 5632;
                *resolu = 1;
                *S03_get = false;
                rsp->read();
                rsp->read();
                break;

        case 65:
                *frequency = 11265;
                *resolu = 2;
                *S03_get = false;
                rsp->read();
                rsp->read();
                break;

        default:
```

```
                        *frequency = 11265;
                        *resolu = 2;
                        break;
                }
        }
}
}

void speed_power_reading(int param1[], char character, int* count)   //This function is to put the value send by the RSP1 in variables so as we can use these value
{
if (int(character) != 59 && int(character) != 10 && int(character) != 13)
{
                if (*count<3)
                {
                param1[0] = param1[0] * 10 + (int(character) - 48);
                }
                else if (*count<6)
                {
                param1[1] = param1[1] * 10 + (int(character) - 48);
                }
                else if (*count<9)
                {
                param1[2] = param1[2] * 10 + (int(character) - 48);
                }
                else if (*count<12)
                {
                param1[3] = param1[3] * 10 + (int(character) - 48);
                }
                *count = *count + 1;
        }

}


void init(int param1[], int* count) //Reinitialisation of all the variable
{
        *count = 0;
        for (int i = 0; i<4; i++) {
                param1[i] = 0;
        }
}

void write_inRSP(char code[], SoftwareSerial* rsp, bool* written)
{
 if(*written)
 {
       rsp->write(code);
       rsp->write(char(10));
       rsp->write(char(13));
       *written = false;
```

```
  }
}
```

A.3 Archivos de encabezamiento

Listado A-3 Código de encabezamientos

```
#include <SoftwareSerial.h>

void speed_power_reading(int param1[],char character,int* count);//This
function is to put the value send by the RSP1 in variables so as we can use
these value

void init(int param1[],int* count); //Reinitialisation of all the variable

void gettingS03Values(char* character, int* frequency, float* resolu, bool*
S03_get, int* count, SoftwareSerial* rsp); //This founction is made to get the
sampled
                                                    //frequency and
the resolution from SO3 parameters

void write_inRSP(char code[], SoftwareSerial* rsp, bool* written); //Is used
to send command to RSP
```