Trabajo Fin de Grado

Grado en Ingeniería Aeroespacial

Integración de sensores doppler para el cálculo de odometría en vehículos aéreos

Autor: Juan Carlos Ortiz Ronda

Tutores: Jesús Capitán Fernández

Fernando Caballero Benítez

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2016

Trabajo Fin de Grado
Grado en Ingeniería Aeroespacial

Integración de sensores doppler para el cálculo de odometría en vehículos aéreos

Autor:
Juan Carlos Ortiz Ronda

Tutores:
Jesús Capitán Fernández
Profesor Ayudante Doctor
Fernando Caballero Benítez
Profesor Contratado Doctor

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2016

Trabajo Fin de Grado: Integración de sensores doppler para el cálculo de odometría en vehículos aéreos

Autor: Juan Carlos Ortiz Ronda

Tutores: Jesús Capitán Fernández
Fernando Caballero Benítez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2016

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

En primer lugar me gustaría agradecer a mi tutor, Jesús, por haberme dado la oportunidad de trabajar en este proyecto, como también a Fernando, así como a todos los maestros y profesores que me han ido formando desde que era pequeño hasta el día de hoy.

En segundo lugar agradecer a mis padres por apoyarme en todo momento, aconsejarme lo mejor posible y depositar una enorme confianza en mí, brindándome la oportunidad de estudiar lo que me gusta.

También agradecer a mi hermana, que seguro estará orgullosa de su hermano.

A mi pareja por su paciencia y cariño, animándome y apoyándome cada vez que lo he necesitado.

Y por último, a aquellas personas que he conocido a lo largo de estos años de carrera, con quien he compartido buenos y malos momentos.

Todos formáis parte de este trabajo y sin vosotros no hubiera sido posible, muchas gracias.

Juan Carlos Ortiz Ronda

Sevilla, 2016

Resumen

Este documento recoge el estudio y análisis de la posibilidad de integración de sensores Doppler de bajo coste en vehículos aéreos no tripulados para el cálculo de la velocidad y la posición en interiores, de forma que no sea necesaria la utilización de otras formas de navegación no autónomas. Para ello se utiliza un sensor Doppler junto con un Arduino, a través del cuál será posible calcular la frecuencia Doppler y de ahí la velocidad. Se han utilizado distintas librerías y funciones con el fin de ver cuál de ellas es la más favorable para nuestro objetivo.

Para comprobar la veracidad de los resultados ha sido posible realizar experimentos en una habitación de "Motion-tracking" de forma que se ha podido comparar resultados precisos de velocidad recogidos por una cámara con los obtenidos del sensor.

Prácticamente lo que se estudia es una manera de abaratar costes para la navegación autónoma en interiores.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

Hoy en día existen diferentes métodos de navegación para los vehículos aéreos, de los cuales los más usados son el GPS y la unidad de medidas inerciales. Para estos dos, se suele realizar una fusión mediante un filtro, ya sea de Kalman u otro, de forma que la posición sea estimada de una manera más precisa. Sin embargo, cuando en un pequeño vehículo se desea estimar la posición en el interior de edificios, la navegación por GPS es imposible debido a la gran pérdida de potencia de la señal GPS cuando atraviesa paredes, suelos y otros objetos.

Es posible a partir de un punto conocido utilizar los acelerómetros y giróscopos que incluye la IMU para estimar la posición, pero esto no es favorable ya que comprende la doble integración de las medidas de aceleración con el consiguiente aumento del error, cuyo valor crece conforme sea mayor el tiempo que se realice la integración. Es debido a que se suele aunar junto con otro sistema, de los cuales hablaremos a continuación.

Se denominan IPS (Indoor Positioning System), y son sistemas de posicionamiento para interior de edificios que permiten localizar un objeto mediante tecnología óptica, de radiofrecuencia, e incluso ultrasónica. Estos pueden ser integrados junto con la IMU para obtener una mayor precisión. [1]

Cabe destacar, que dentro de los sistemas de navegación, estos pueden ser autónomos o no autómos, siendo los autónomos aquellos que no dependen de ningún sensor exterior que mande información hacia el móvil para poder posicionarse y los no autónomos aquellos que no necesitan de ningún sensor externo ya que les es suficiente con los equipados a bordo. De entre los sistemas no autónomos podemos encontrar el posicionamiento por Wi-Fi que necesita de diferentes puntos de acceso para medir la intensidad de la señal recibida y localizar el objeto mediante triangulación. Otro de ellos es la navegación por visión, la cuál necesita de unas referencias o marcadores visuales, fundamento en el que se basa la cámara de "Motion-tracking" que utilizaremos para nuestras pruebas experimentales.

En cuanto a los sistemas autónomos encontramos la navegación mediante campo magnético. Sin embargo esta necesita de un mapa realizado previamente del campo magnético del área y es muy susceptible a interferencias.

Actualmente se están desarrollando diversas técnicas distintas de las anteriores comentadas utlizando tecnología basada en láser. Es el caso de la tecnología LIDAR, que permite determinar la distancia desde un emisor láser a un objeto o superficie utilizando un haz láser pulsado. Esta distancia al objeto se determina midiendo el tiempo de retraso entre la emisión del pulso y su detección a través de la señal reflejada. Es así como es posible crear mapas del entorno, como por ejemplo, mapas de interiores de edificios. Si se realizan varias mediciones de distancia en un intervalo de tiempo, es posible calcular la velocidad de movimiento del vehículo, que es el objetivo final del estudio. Sin embargo, su alta precisión se traduce en un alto coste.

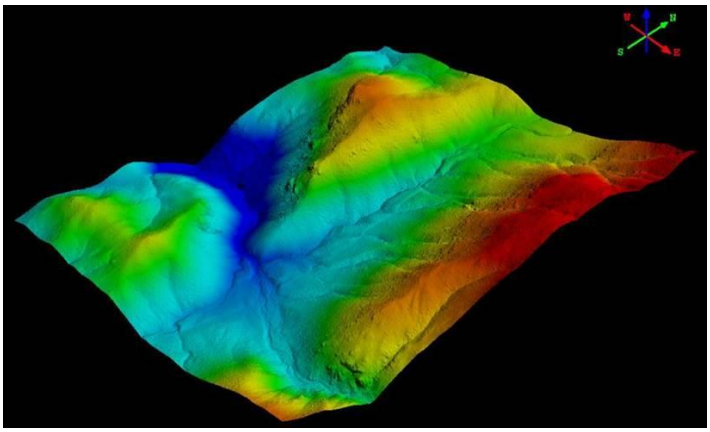


Figura 1-1: Mapa realizado con tecnología LIDAR

Es por ello, que se quiere buscar otro método que permita el cálculo de la odometría, es decir, la posición del vehículo, de modo que sea más barato y menos complejo de implementar que la tecnología LIDAR. Esto es posible a través del uso del efecto Doppler mediante un sensor que permita calcular la velocidad del móvil. En esto es en lo que se fundamenta el presente trabajo y es lo que abordaremos en los siguientes capítulos.

Por tanto, si se cumple la posibilidad de utilizar el sensor Doppler para la navegación interior, será útil su implementación en UAV's de bajo coste, permitiendo abaratar su construcción por parte de las empresas obteniendo así de un sistema de navegación autónomo cuya precisión puede mejorarse mediante su unión junto con otros sensores mediante un filtro de Kalman.

Una vez obtenida la velocidad, si esta es bastante precisa es posible utilizar la integración para obtener el desplazamiento del móvil y permitir así el posicionamiento en interiores.

2 FUNDAMENTOS DEL EFECTO DOPPLER

La forma por la que se va a calcular la velocidad es mediante la aplicación del efecto Doppler. Existen distintas maneras de utilizar esta técnica dependiendo del modo en el que el emisor transmita la onda. Un sensor Doppler es un tipo de radar, por tanto, a continuación se explicará el fundamento del mismo y los distintos tipos de radar que pueden hallarse.

2.1. Efecto Doppler

En primer lugar, ya que el trabajo se basa en la técnica Doppler, explicaremos los fundamentos de la misma. [2]

El efecto Doppler es un fenómeno producido durante la observación física de las ondas. Se produce cuando la fuente emisora de ondas y el observador se encuentran en movimiento relativo con respecto al medio material en el que se propaga; la frecuencia de las ondas observadas es distinta a la frecuencia de las ondas emitidas por la fuente. Esta variación es lo que produce el llamada efecto Doppler y la desviación en la frecuencia suele llamarse frecuencia Doppler. Esta relación entre la frecuencia observada y la frecuencia emitida viene dada por la siguiente ecuación:

$$f = (\frac{c+v_r}{c+v_s})f_0 \tag{2-1}$$

Siendo:

- f la frecuencia emitida.
- f_0 la frecuencia observada.
- c la velocidad de la luz: $3 \cdot 10^8\ m/s$.
- v_r es la velocidad del receptor en relación con el medio, positiva si el receptor se mueve hacia el emisor.
- v_s es la velocidad de la fuente con respecto al medio, positiva si la fuente se aleja del receptor.

Este argumento es aplicable a cualquier tipo de ondas, ya sean ondas magnéticas, sonoras… Por ejemplo, para el caso de las ondas sonoras, podemos poner como ejemplo la sirena de una ambulancia [3] que pasa por delante de nosotros. Si la ambulancia se acerca a cierta velocidad, la escucharemos cada vez más con un tono más agudo que el real, debido a que la frecuencia va aumentando. En cambio, si se aleja de nosotros, la escucharemos más grave y por tanto la frecuencia de las ondas sonoras disminuye, tal y como se muestra en la siguiente imagen.

Figura 2-1: Observación del efecto Doppler en la sirena de una ambulancia

En este caso, como el receptor y el emisor son los mismos ya que se encuentran ambos en el vehículo, y además, como la señal transmitida hace el mismo camino que la recibida por ser un radar primario, término que se explicará en el siguiente subapartado, $v_r = -v_s$. La ecuación queda entonces de la siguiente forma.

$$f = \left(\frac{1+\frac{v_r}{c}}{1-\frac{v_r}{c}}\right) f_0 \tag{2-2}$$

$$\Delta f = f - f_0 = \left(\frac{1+\frac{v_r}{c}}{1-\frac{v_r}{c}}\right) f_0 - f_0 = f_0\left(\frac{1+\frac{v_r}{c}}{1-\frac{v_r}{c}} - 1\right) \tag{2-3}$$

$$\Delta f = \frac{2\frac{v_r}{c}}{1-\frac{v_r}{c}} f_0 \tag{2-4}$$

Siendo $\Delta f = f - f_0$, la desviación de frecuencia o frecuencia Doppler. Como $\frac{v_r}{c} \ll 1$:

$$\Delta f = 2\frac{v_r}{c} f_0 \tag{2-5}$$

Ahora bien, si el movimiento no es colineal con la trayectoria de las ondas del sensor, la velocidad varía en función del ángulo que forman las ondas con el movimiento del vehículo.

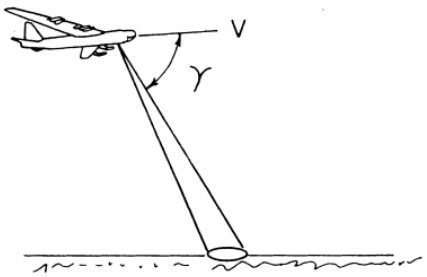


Figura 2-2: Haz del radar Doppler

$$v_r = v \cdot \cos\gamma \tag{2-6}$$

De esta forma, la ecuación quedaría:

$$\Delta f = 2\frac{v}{c} f_0 \cos\gamma \tag{2-7}$$

Por otro lado, como $\frac{c}{f}$ es igual a la longitud de onda λ, esta última ecuación se podrá escribir como:

$$\Delta f = 2\frac{v}{\lambda}\cos\gamma \tag{2-8}$$

Donde λ es la longitud de onda de la señal transmitida.

Resumiendo, cuando montemos el sensor en el robot, este transmitirá una señal en una frecuencia. Estas ondas rebotarán en las paredes y volverán al sensor, que recibirá la señal. Si el robot se encuentra parado, la frecuencia transmitida y recibida serán la misma por lo que no se detecta movimiento y la velocidad es nula. No obstante, si el robot se encuentra en movimiento, la frecuencia recibida será de unos herzios mayor o menor de la transmitida, y por tanto se detecta que se ha movido y es posible calcular la velocidad.

A continuación, se explicará más en detalle en qué consiste el radar [4] y los diferentes tipos que existen, donde incluiremos nuestro sensor.

2.1 Tipos de Radar

En primer lugar, hay que diferenciar entre los radares primarios y secundarios. Un radar primario es aquel que funciona con independencia del blanco, en el que su funcionamiento se basa en el eco de la señal. En cambio, un radar secundario es aquel que manda una señal al blanco, lo interroga y este responde con otra señal.

Figura 2-3: Radar primario (izquierda) y radar secundario (derecha).

Se pueden encontrar distintos tipos de radar dependiendo de cómo se transmita la onda. En función de cómo sea, este permite calcular la velocidad y/o la distancia. Tras el siguiente párrafo se explicará detalladamente como es cada uno de ellos.

Otro de los conceptos importantes es el RCS (Radar Cross Section) o Corte Transversal de Radar e indica en un blanco cuál es el área efectiva de éste que permite reflejar parte de la potencia de la señal incidente hacia el transmisor. Es una medida que permite saber cuán de detectable es un objeto mediante radar. Por tanto, un RCS mayor indica que el objeto es más fácil de detectar. En el caso del sensor Doppler, se necesita que se tenga un RCS alto a la hora de reflejar la señal. Como este estará orientado hacia el suelo, no habrá problema en cumplir este requisito.

2.2.1. Radar de Onda Continua (CW)

Este es el más simple de todos y el más barato de implementar ya que el Tx, es decir, el transmisor, envía continuamente y el Rx, el receptor, recibe continuamente. Una forma de onda continua pura solamente sirve para medir velocidad radial por efecto Doppler, por lo que tiene sus limitaciones a la hora de medir distancia. Para poder medirlas con un radar CW es necesario modularlo, por ejemplo, mediante modulación FM lineal con la que pueda realizarse una marca de pulso. Aunque de esto se hablará en el radar CW-FM que explicaremos en breves.

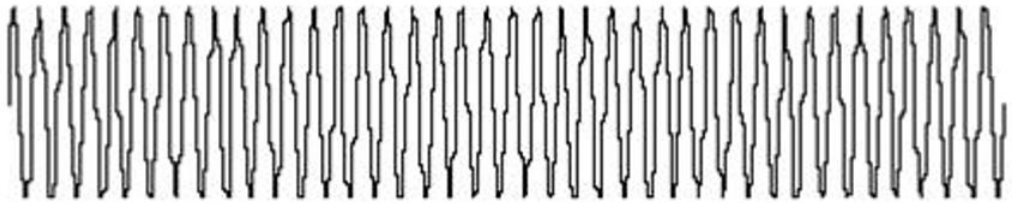


Figura 2-4: Transmisión de onda continua a una cierta frecuencia

De este tipo es nuestro sensor Doppler, es un radar de onda continua que transmite una señal a una frecuencia especifica sin interrupciones a la vez que recibe el eco.

2.2.2. Radar de Impulsos

Este tipo de radar emite un tren periódico de pulsos y detecta los ecos reflejados por los blancos. El tiempo transcurrido entre la emision y la recepción de un pulso resuelve la distancia al blanco según la formula:

$$R = \frac{cT}{2} \tag{2-9}$$

Siendo c la velocidad de la luz y T el tiempo transcurrido entre emisión y recepción.

Si la emisión y la recepción son coherentes, es decir, conservan la fase de portadora, es posible determinar la frecuencia Doppler y medir la velocidad relativa del blanco. Para ello aplican el efecto Doppler tal y como lo conocemos, aplicando la formula 2-7.

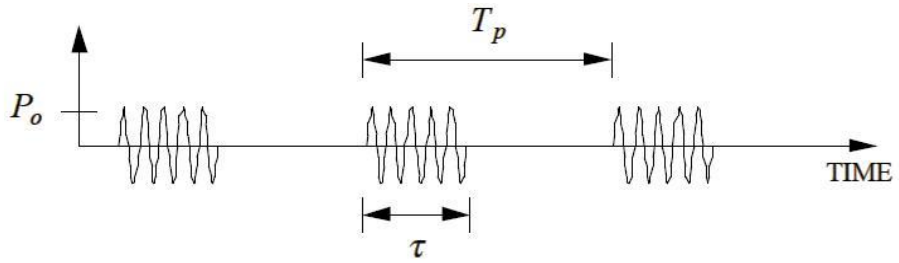


Figura 2-5: Transmisión de un radar de impulsos coherente

2.2.3. Radar CW-FM

Este es un tipo especial de radar que envía una señal continua como el radar CW. Sin embargo, a diferencia de este ultimo, esta se envía con modulación lineal en frecuencia durante un intervalo T_m. Los ecos se reciben de modo continuo, sin interrumpir la transmisión. La señal recibida se mezcla en el receptor con la señal transmitida y el resultado pasa por un filtro paso bajo para producir una superposición de frecuencias $f_b = f_{TX} - f_{RX}$, denominado frecuencia de batido. A partir de esta, es posible considerando el resto de características de la señal como parámetros, calcular la distancia al blanco, aplicando la siguiente formula:

$$R = \frac{cT_m}{2}\frac{f_b}{\Delta f} \tag{2-10}$$

Un radar FMCW es la forma más simple de radar de impulsos doppler. El "impulso" en este caso va desde el inicio de la modulación hasta su final, es decir, pulsos de Tm, sin separación entre ellos.

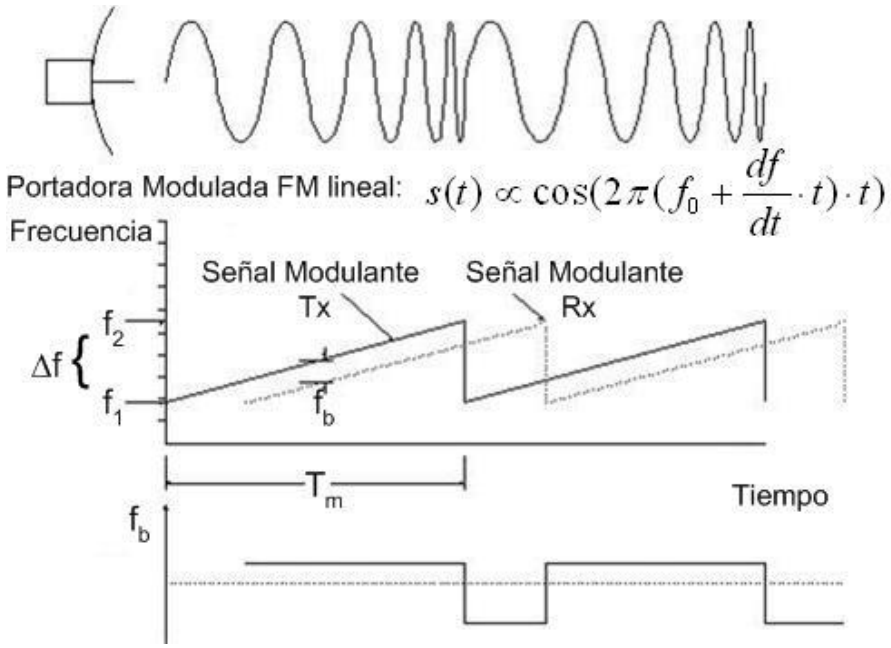


Figura 2-6: Transmisión y recepción de radar CW-FM

Como T_m es el tiempo entre pulsos triangulares, la máxima distancia a un blanco que un radar FM-CW puede medir sin ambigüedades es:

$$R_{máx} = \frac{cT_m}{2} \tag{2-11}$$

Es decir, la máxima distancia es aquella que permite a la señal la ida y vuelta a la velocidad de la luz antes de que se comience a transmitir el siguiente pulso modulado. Si el blanco se encuentra en movimiento, la forma de onda detectada tendrá un desplazamiento Doppler en frecuencia. A partir de esto, sería posible calcular la velocidad.

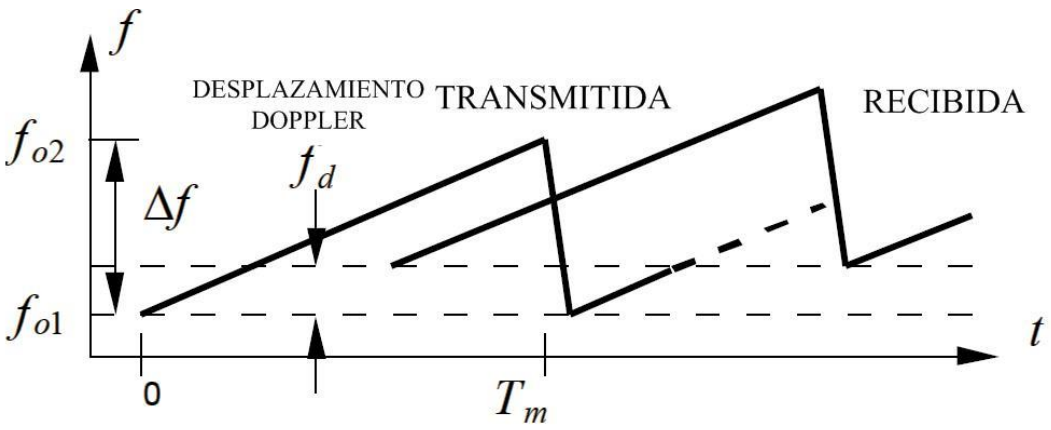


Figura 2-7: Transmisión y recepción en radar CW-FM con desviación Doppler

Por tanto, con un radar de onda continua modulado en FM es posible detectar tanto la velocidad como la distancia al blanco.

3 OBJETIVOS

Una vez entendido el efecto Doppler, éste lo vamos a aplicar al trabajo mediante un sensor de bajo coste, concretamente el X-Band Motion Detector de la empresa Parallax. La idea final es montar los sensores necesarios para que sea posible el cálculo de la velocidad respecto de tierra en un vehículo aéreo, sin la necesidad de GPS y otros sensores no autónomos.

Para ello, en primer lugar, se procederá a comprobar si esto es viable en un único sensor. Se ha comprado dicho componente junto con un Arduino Leonardo. Tras realizar la conexión de ambos correctamente será posible hallar la frecuencia Doppler, es decir, la desviación de la frecuencia recibida respecto de la emitida, que se muestra mediante la oscilación de un tren de pulsos entre valores 1 y 0. Esta oscilación será leída por el Arduino a través de uno de los pines. A partir de esto, se aplicarán distintos modos de lectura de dicho valor mediante la utilización de librerías y funciones en el código del Arduino de forma que sea posible calcular la velocidad.

Como se aplican distintos métodos, se va a permitir comprobar cuál es el más preciso de los utilizados mediante la comparación de los datos recogidos por el sensor, y con los de una cámara de detección de movimiento y seguimiento. Para ello se ha montado el dispositivo en un robot en movimiento en una habitación habilitada para la detección y seguimiento a partir de marcadores visuales. Por un lado tendremos la velocidad calculada por el sensor y el Arduino, y por otro, la velocidad calculada por la cámara instalada en la sala.

Tras una comparación de los resultados, se obtendrá la forma que permita calcular la velocidad con el menor error posible.

Una vez que se haya comprobado qué método es el más favorable para calcular la velocidad, se estudiará la posibilidad de integrar sensores Doppler en un UAV y de cómo habría que hacerlo.

Por otro lado, en los Anexos se adjuntan tanto los códigos y programas utilizados, como una breve guía de la estructura de un programa de Arduino

4 HARDWARE

En este capítulo se describe el hardware específico que se ha utilizado para la realización del trabajo. Se explicará el sensor Doppler y la tarjeta de Arduino Leonardo, utilizada para recoger los datos del sensor y realizar el cálculo de la velocidad a partir de los mismos, así como también servir de enlace entre el sensor Doppler y el PC.

4.1 X-Band Motion Detector

Éste es el sensor Doppler [5], que si bien no está diseñado para el cálculo de la velocidad sino para la detección de movimiento, se ha podido utilizar para nuestro fin. Se trata de un componente de bajo coste, que cuesta alrededor de unos ~30€.

Éste opera en la frecuencia de banda X, a unos 10.525 Ghz e indica que existe movimiento mediante oscilaciones en su salida, que varía entre `HIGH` (1 binario) y `LOW` (0 binario). Tiene un alcance de entre ~2.4 a 9+ metros, dependiendo de la sensibilidad, ajustable con un potenciómetro.

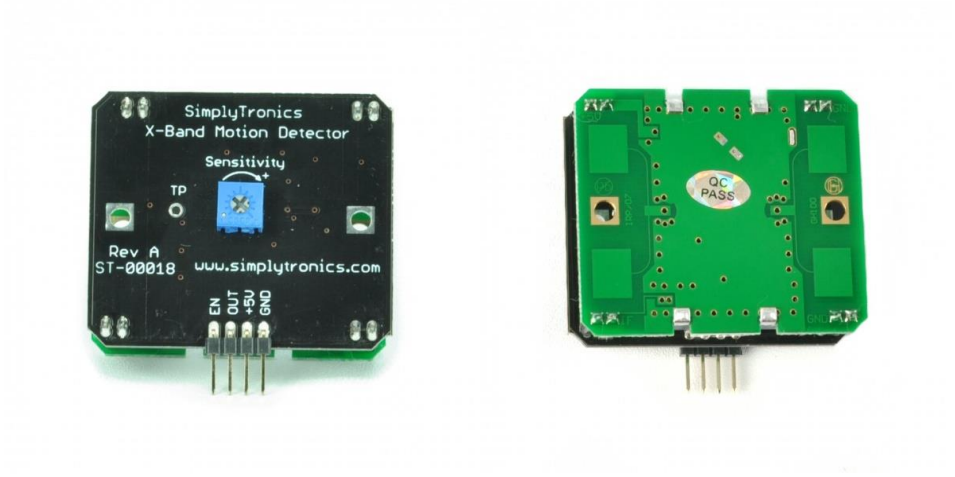


Figura 4-1: Sensor X-Band Motion de Parallax

El sensor está compuesto de dos placas unidas. En la cara de una de las placas podemos encontrar la antena PCB con los módulos de transmisión y recepción. El dispositivo debe estar orientado de forma que esta superficie mire hacia el área de detección. En la cara de la otra placa encontramos el potenciómetro para ajustar la sensibilidad y los pines de conexión.

Encontramos cuatro pines en el sensor:

- **EN:** Cuando este pin se encuentra activado (o en HIGH), el dispositivo comienza a realizar mediciones de radar Doppler periódicas, breves y de baja potencia.
- **OUT:** Oscila entre HIGH y LOW, y cuya fluctuación depende de la velocidad del movimiento. Es un tren de pulsos periódico con un duty cycle de un 4% y frecuencia variable.
- **+5V:** Conexión con una alimentación de 5V.
- **GND:** Conexión a tierra.

Además de variar la sensibilidad con el potenciómetro, esta también varía con el ángulo del objeto en el que refleja la onda debido al patrón de radiación de la antena, donde es máxima en la región que abarca en el rango de [0dB, -3dB] como se puede comprobar en la siguiente ilustración.

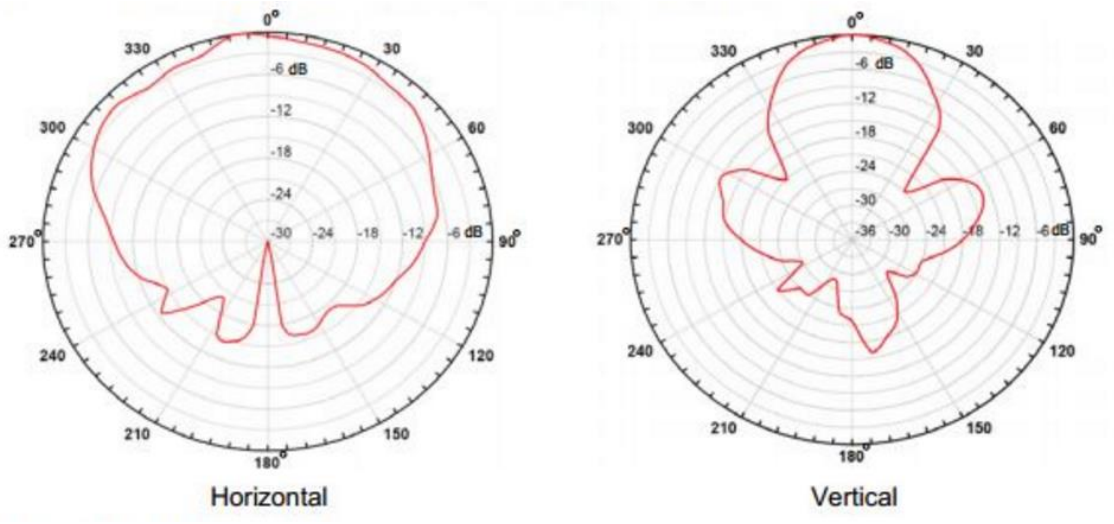


Figura 4-2: Patrón de radiación horizontal y vertical de la antena PCB

Su funcionamiento interno es sencillo. Cuando el pin EN se activa, la placa de control activa el sensor Doppler con un duty cycle de un 4%. El oscilador crea una señal de 10.525Ghz y la manda al Tx de la antena, y también a un diodo mezclador. La salida de este último contiene señales con la suma y la diferencia de las frecuencias transmitidas y recibidas junto con las componentes de la señal original y algunos armónicos.

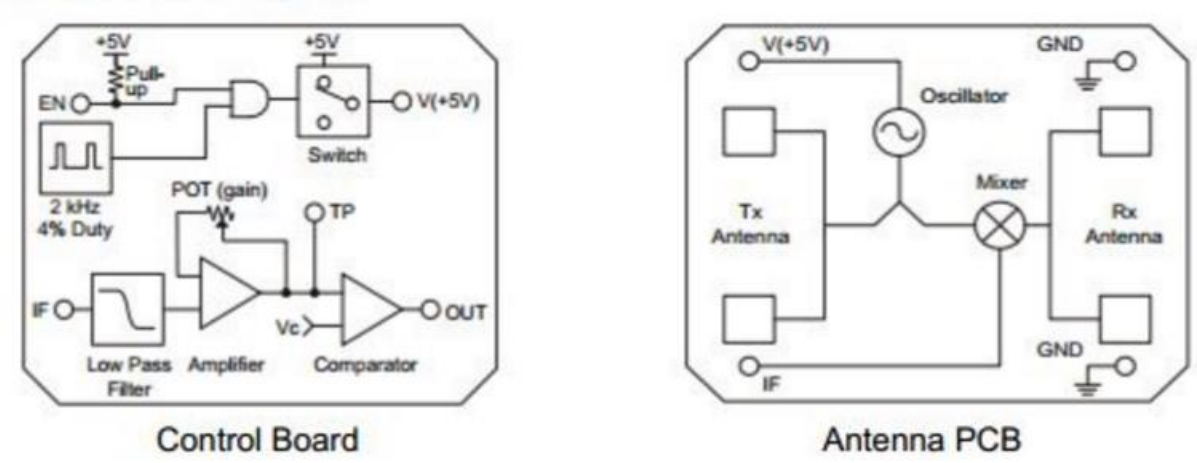


Figura 4-3: Diagrama de bloques de las placas del sensor Doppler

Como el haz de la antena tiene un ancho finito y debido a la dispersión producida por la tierra, la información recibida no es una única frecuencia, ya que incluye ruido. Tal y como se puede ver en la siguiente figura donde se muestra como ejemplo el espectro en frecuencia de una señal Doppler.

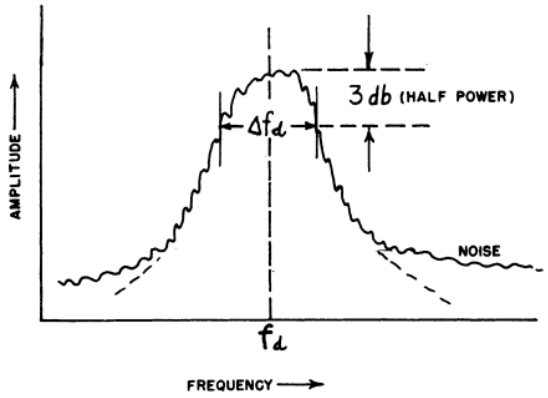


Figura 4-4: Espectro típico de una señal Doppler

Por ello, una vez que la señal se recibe en la placa de control, esta pasa por un filtro paso bajo que elimina las componentes de alta frecuencia y deja solamente la diferencia de las señales. Esta última pasa por un amplificador cuya ganancia puede ser ajustada mediante el potenciómetro de la placa de control. Posteriormente pasa por un comparador que transforma la diferencia de frecuencias a una salida digital high/low. La desviación de la frecuencia de la señal saliente respecto a la entrante está relacionada con la velocidad del objeto mediante la siguiente ecuación, ya vista anteriormente cuando explicamos el efecto Doppler:

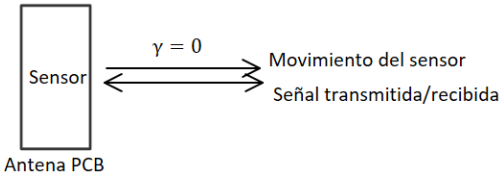
$$f_d = 2V\frac{F_t}{c}\cos\gamma \tag{4-1}$$

Donde:

- f_d es la desviación de frecuencia, o también llamada frecuencia Doppler.
- V es la velocidad del objeto.
- F_t es la frecuencia de transmisión, en nuestro caso 10.525 Ghz.
- c es la velocidad de la luz: $3 \cdot 10^8\ m/s$.
- γ es el ángulo de desviación de la dirección del movimiento respecto a la perpendicular de la antena PCB.

En el caso del presente trabajo, se calcula la velocidad longitudinal, es decir, la dirección de la velocidad será perpendicular a la superficie de la antena PCB por lo que el ángulo de desviación γ será nulo. Sin embargo, este ángulo no será nulo en el caso en el que la dirección del movimiento no sea la longitudinal, tal y como se resume en el siguiente gráfico.

Caso A: Movimiento y señal en la misma dirección



Caso B: Velocidad y señales en distinta dirección

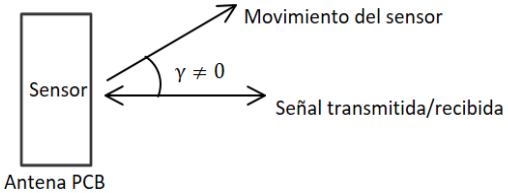


Figura 4-5: Ángulo θ dependiendo de la dirección de la propagación de la señal y el movimiento

Como la tentativa final del trabajo es la implementación en un UAV, será necesario de la utilización de varios sensores Doppler en ejes distintos que permitan medir diferentes componentes de la velocidad.

A diferencia de las pistolas de radar basadas en Doppler, las cuales utilizan un guiaondas para direccionar la antena a un patrón de radiación con un haz más estrecho, nuestro sensor tiene un patrón de radiación mucho más ancho por lo que pueden producirse imprecisiones. Así mismo, en el datasheet [5] se nos informa que aunque es posible calcular la velocidad, el sensor no está desarrollado para ese fin sino para la detección de movimiento.

4.2 Arduino Leonardo

Es una placa microcontroladora [6] basada en el chip ATmega32u4. Tiene 20 pines digitales que pueden actuar tanto de salida como de entrada y un oscilador de cristal de cuarzo de 16 MHz. Es posible su conexión al ordenador mediante un conecto micro-USB, y además tiene la ventaja de que permite obtener la alimentación mediante el puerto micro-USB. Se muestra en la siguiente imagen:

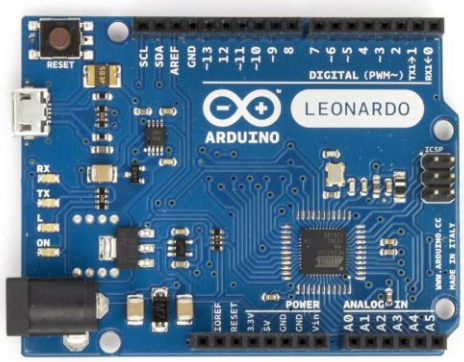


Figura 4-6: Arduino Leonardo

4.3 Montaje de los componentes y conexión

Para el montaje de ambos se ha realizado una placa de madera donde poder colocar el sensor en posición vertical por un lado, y el Arduino por otro lado sujetado mediante bridas.

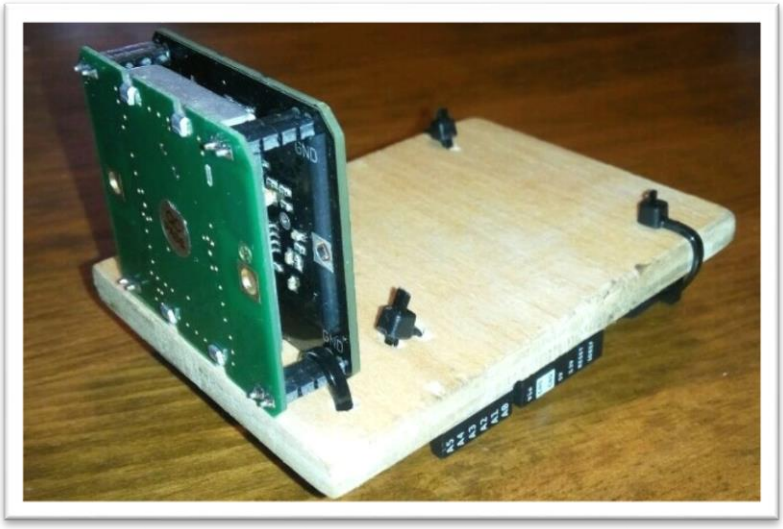


Figura 4-7: Parte superior del montaje

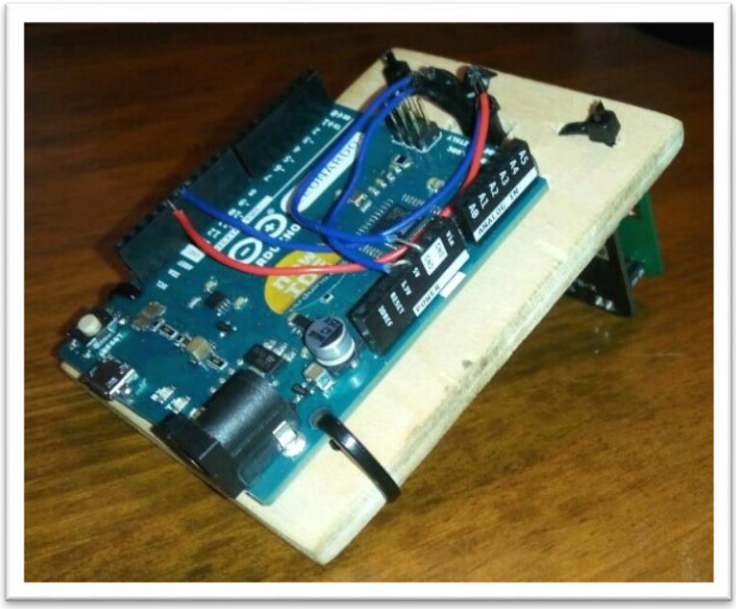


Figura 4-8: Parte inferior del montaje

La conexión con el sensor Doppler la realizaremos de la siguiente forma:

- Pin 5V del sensor ➔ Pin 5V del Arduino
- Pin GND del sensor ➔ Pin GND del Arduino
- Pin EN del sensor ➔ Pin 13 del Arduino (Pin 12 en FreqMeasure)
- Pin OUT del sensor ➔ Pin 12 del Arduino (Pin 13 en FreqMeasure)

Esto es así ya que las librerías utilizan dichos pines para su uso con el TIMER del microcontrolador, necesario para el cálculo de la frecuencia.

Para la conexión de la tarjeta con el ordenador se utiliza el cable usb-microusb proporcionado. Para la programación y la carga del script en el Arduino hemos utilizado el mismo IDE de Arduino.

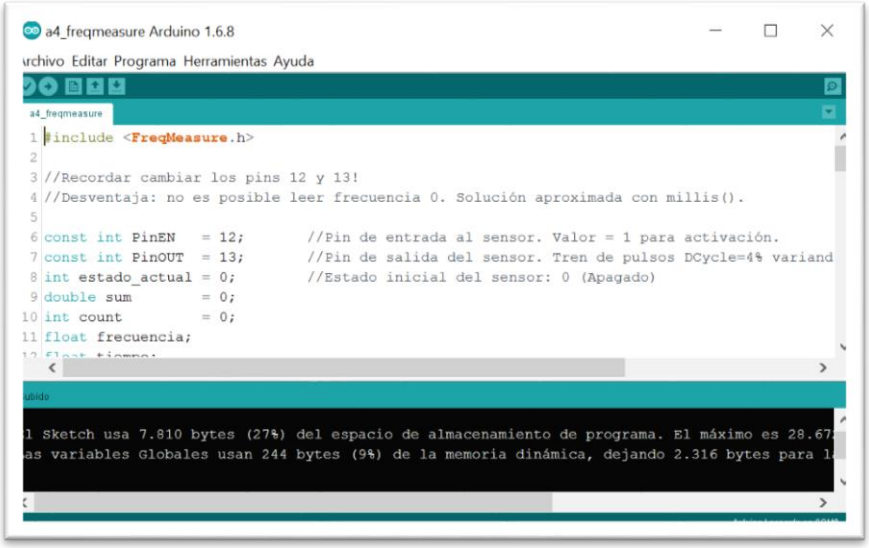


Figura 4-9: Interfaz del IDE de Arduino

Por otro lado, para la visualización de datos se ha considerado mejor el software denominado RealTerm, ya que permite leer el puerto serie del Arduino así como poder capturar los datos para posteriormente guardarlos en un archivo de texto y procesarlos en MatLab.

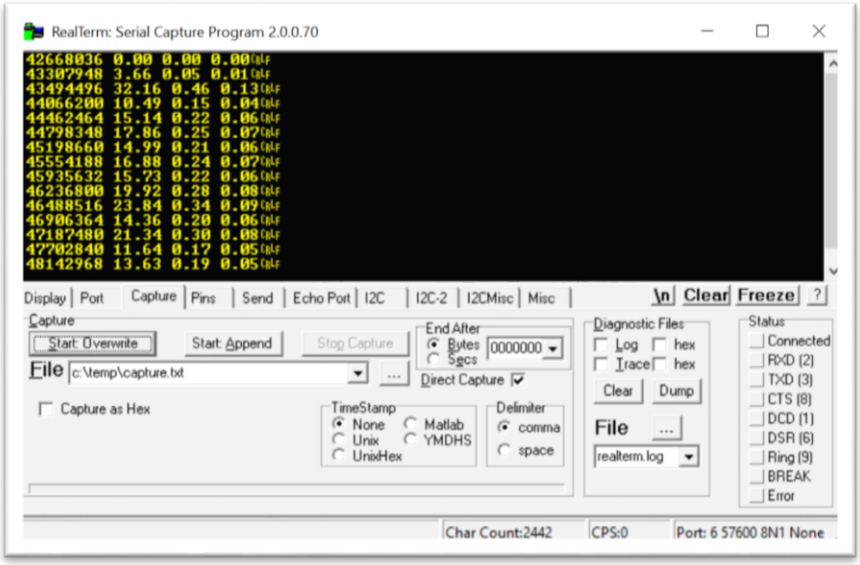


Figura 4-10: Interfaz de RealTerm con visualización del puerto serie

5 ESTUDIO DEL PROBLEMA

Uvez presentados los componentes de los que se dispone, es necesario realizar la programación del microcontrolador para que éste permita obtener la frecuencia del pin de salida del sensor Doppler, y a partir de ahí calcular la velocidad. Como se ha comentado anteriormente, nuestro sensor muestra por el pin de salida un tren de pulsos con frecuencia variable que oscila con valores `HIGH` y `LOW`. Por tanto, tendremos que desarrollar un código que permita observar esa oscilación y mostrarnos el número de veces que cambia en un intervalo de tiempo prefijado. Para ello hemos investigado y hemos dado con diferentes librerías y funciones que son aplicables a nuestro problema.

Figura 5-1: Tren de pulsos en la salida del sensor Doppler

Antes de explicar en qué consiste el código, se recomienda ver en el Anexo A el funcionamiento y la estructura de un programa en Arduino con el objetivo de entender mejor lo que viene a priori. [Ver Anexo A]

Éste está compuesto principalmente de cuatro bloques, de los cuales sólo uno de ellos cambia dependiendo de la librería o función utilizada para el cálculo de la frecuencia. Se presenta un diagrama de flujo que puede aclarar las ideas sobre este apartado, el cual quiere explicar que una vez se inicia la comunicación con el Arduino, este espera a que se detecte movimiento para continuar con el programa.

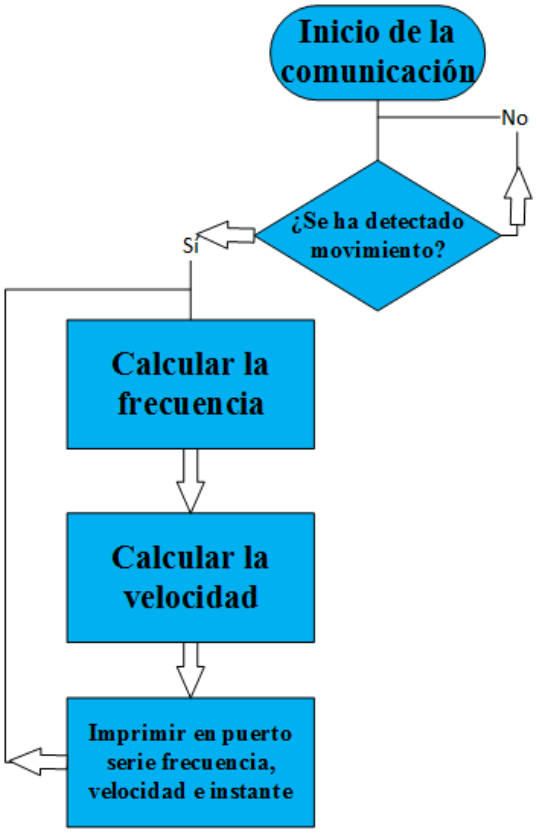


Figura 5-2: Diagrama de flujo del funcionamiento del programa

El funcionamiento de cada bloque lo detallaremos a continuación:

1) Bloque de Inicio: Se inicia la comunicación con el Arduino a una tasa de 115200 baudios. Además, se establecen que pines son los de entrada y salida de datos al sensor. También se crean las variables que van a ser utilizadas en el programa. Un ejemplo de este bloque sería el siguiente:

```
#include <FreqMeasure.h>


const int PinEN   = 12;
const int PinOUT  = 13;
int estado_actual = 0;
double sum        = 0;
int count         = 0;
float frecuencia;
float tiempo;
unsigned long tiempo0     = 0;


void setup() {
  Serial.begin(115200);
  pinMode(PinOUT, INPUT);
  pinMode(PinEN, OUTPUT);
  enable();
  FreqMeasure.begin();
}
```

2) Bloque de Espera: Se inicia el sensor escribiendo en el pin `EN` a `HIGH`. Se espera a que detecte movimiento para ponerse a funcionar y detectar la frecuencia. Se hace mediante un bucle `while` que no deja avanzar hasta que el sensor no modifique el pin de salida. Para más inri, se muestra un ejemplo a continuación.

```
void wait()  {                          //Espera a que el sensor devuelva un valor de 1
  while (digitalRead(PinOUT) != 1)
  {
  }
  estado_actual = 1;    //Ahora el sensor se encuentra activado y saca un tren de pulsos
  //Serial.println("Movimiento detectado!");
```

3) Bloque de Cálculo: Este bloque es diferente para cada uno de los códigos. Se calcula la frecuencia del pulso y se guarda en una variable. Se detallará su funcionamiento en los apartados posteriores.
4) Bloque de Velocidad: Se calcula la velocidad aplicando la siguiente fórmula, que resulta de despejar la velocidad de la ecuación 4-1.

$$V = \frac{F_d \cdot c}{2 \cdot F_t \cdot \cos\theta} \tag{5-1}$$

Como se ha comentado en apartados anteriores, en nuestro caso el valor de θ será nulo ya que la dirección de movimiento del sensor será colineal con la dirección de la normal a la superficie de la antena. Por otro lado, existe ambigüedad, ya que si $\theta = 90$ la ecuación no podría resolverse. Aún así, este ángulo debe procurarse que sea lo más agudo y pequeño posible para mejorar la precisión.

Finalmente se muestran los valores de velocidad tanto en m/s como en km/h por el puerto serie del Arduino, además del instante de tiempo en el que se han capturado.

```
void print_speed()
{
  Serial.print(" ");
  Serial.print(micros()-tiempo0);
  Serial.print(" ");
  Serial.print(frecuencia);
  Serial.print(" ");
  float ratio = 299792458.0 / (2 * 10525000000); //Razón calculada por c/(2*freq_emitida)
  Serial.print(frecuencia * ratio);
  Serial.print(" ");
  Serial.println(frecuencia * ratio / 3.6);
}
```

A continuación comentamos cada uno de las librerías o funciones utilizadas, que son cuatro. Por tanto, tenemos cuatro formas de calcular la frecuencia, de las cuales elegiremos la mejor de ellas tras las pruebas de ensayo.

5.1 FreqCounter

En este caso se ha tomado una de las librerías realizadas por Martin Nawrath [7] diseñada para medir frecuencias con una resolución alta y precisa. Esta librería hace uso del TIMER1 del microcontrolador del Arduino para provocar una interrupción y poder contar cuántos pulsos se producen en un intervalo prefijado denominado "gatetime". En nuestro caso hemos fijado este valor para 500 milisegundos. Además de ello, contiene un factor de compensación que permite corregir errores en el gatetime. Sin embargo, no hemos tocado este valor y lo hemos dejado por defecto ya que para calibrarlo es necesario una comparación con un contador profesional que no disponemos.

Cabe decir que esta librería en un principio no era posible utilizarla para nuestro microcontrolador. Es por ello que se ha tenido que modificar para que fuese funcional con el Arduino Leonardo.

La desvestaja de utilización de ésta reside en que afecta a las salidas PWM de los micros ATMega, incluido en la tarjeta microcontroladora.

Las funciones de está libreria las comentamos en las siguientes lineas:

- `FreqCounter::f_comp=8` ➔ Este es el valor de calibración para el gatetime que no hemos tocado y lo hemos dejado por defecto.
- `FreqCounter::start(gatetime)` ➔ Comienza a contar la frecuencia para cada interval del gatetime.
- `FreqCounter::f_ready` ➔ Devuelve "true" cuando una nueva medida está disponible.
- `FreqCounter::f_freq` ➔ Devuelve el valor de flancos de subida vistos en un intervalo del gatetime.

Un ejemplo de programa donde se utilizan estas funciones sería el que se presenta a continuación:

```
#include <FreqCounter.h>


void setup() {
  Serial.begin(112500);
}
long int freq;
Void loop() {
 FreqCounter::f_comp= 8;
 FreqCounter::start(100);
 while (FreqCounter::f_ready == 0)
 freq=FreqCounter::f_freq;
}
```

5.2 FreqCount

Esta librería según su autor [8] es muy similar a la anterior pero contiene mejoras pues no necesita de un factor de compensación para el gatetime. Funciona de igual forma, provocando una interrupción cada intervalo de tiempo prefijado que en nuestro caso será de 500 milisegundos.

Recomienda utilizarse para lectura de frecuencias comprendidas entre 1 kHz a 8 Mhz. No es nuestro caso pues los valores rondan en un interval de [0, 50] Hz. Aún así se ha querido probar esta librería para comprobar su funcionamiento. Al estar basada en la librería FreqCounter, la funcionalidad PWM queda inutilizada mientras está iniciado el contador de frecuencia.

Las funciones que esta incluye son las siguientes:

- `FreqCount.Begin(gatetime)` ➔ Comienza a contar la frecuencia cada intervalo del gatetime, expresado en milisegundos.
- `FreqCount.available()` ➔ Devuelve "true" cuando una nueva medida está disponible. Sólo una medida será guardada en caché por lo que debe ser leida antes del siguiente intervalo del gatetime.
- `FreqCount.read()` ➔ Devuelve la medida más reciente, el número de flancos ascendentes que se han visto dentro de un intervalo del gatetime.
- `FreqCount.end()` ➔ Detiene el contador de frecuencia. Ahora sí es posible utilizar la funcionalidad PWM.

Se muestra un código que implementa esta librería como ejemplo:

```
#include <FreqCount.h>


void setup() {
  Serial.begin(112500);
  FreqCount.begin(1000);
}
void loop() {
  if (FreqCount.available()) {
    unsigned long count = FreqCount.read();
  }
}
```

5.3 FreqMeasure

Es del mismo autor que freqcount [9]. Su uso está recomendado para frecuencias comprendidas entre 0.1 Hz y 1 kHz, que es nuestro caso por lo que se espera que este sea el método de los más precisos para cumplir nuestro objetivo. La diferencia con la anterior se establece en que en vez de contar el número de pulsos que ocurren en un intervalo, esta librería calcula cuál es el tiempo que transcurre en un solo ciclo. Esta diferencia podemos observarla en la siguiente figura.

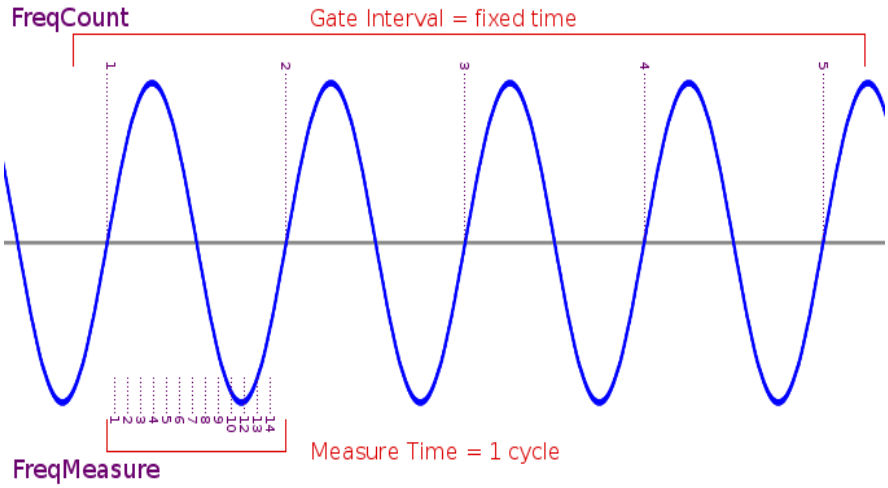


Figura 5-3: Diferencia entre FreqCount y FreqMeasure

Tiene una desventaja y es que el costo computacional aumenta conforme la frecuencia es mayor. Sin embargo, como en nuestro caso manejaremos frecuencias bajas, esta desventaja no nos concierne. Por otro lado, no es capaz de leer frecuencia nula por lo que se ha implementado un "timeout" que muestre frecuencia cero si no se ha detectado ninguna en un intervalo establecido.

Al igual que las funciones anteriores, la funcionalidad PWM es inservible mientras está disponible el contador de frecuencia.

Las funciones que ésta incluye se describen a continuación:

- `FreqMeasure.begin()` ➔ Comienza el contador de frecuencia.
- `FreqMeasure.available()` ➔ Devuelve el número de medidas disponibles para leer, o 0 si no hay ninguna sin leer.
- `FreqMeasure.read()` ➔ Lee una medida, el número de ciclos de reloj de la CPU que han transcurrido durante un ciclo del tren de pulsos. Cada medida comienza inmediatamente después de la anterior, sin retardo ninguno, por lo que varias medidas pueden promediarse para una mejor resolución.
- `FreqMeasure.countToFrequency(count)` ➔ Convierte la medida de la función anterior a la frecuencia actual.
- `FreqMeasure.end()` ➔ Termina el contador de frecuencia. Ahora sí es posible utilizar la funcionalidad PWM.

Como ejemplo de implementación de esta librería tenemos el siguiente programa:

```
#include <FreqMeasure.h>
double sum=0;
int cont=0;

void setup() {
  Serial.begin(112500);
  FreqMeasure.begin();
}

void loop() {
  if (FreqMeasure.available()) {
    sum = sum + FreqMeasure.read();
    cont = cont + 1;
    if (cont > 30) {
      float freq FreqMeasure.countToFrequency(sum / cont);
      sum = 0;
      cont = 0;
    }
  }
}
```

Como vemos, y aprovechando que la función `FreqMeasure.read()` no tiene latencia al leer las medidas, se hace una media durante 30 ciclos del pulso, y luego ya se decide guardar el valor en la variable en `freq`.

5.4 PulseIn

Para este script no hemos utilizado ninguna librería, sino que hemos usado una función incluida en Arduino que permite leer el pulso en un pin prefijado. Cuando el pin de entrada de datos procedentes del sensor está en `HIGH`, se comienza a contar y termina cuando el mismo pin vuelve al estado LOW. El resultado es la longitud del pulso en microsegundos o cero si el pulso no se ha completado dentro del "timeout" fijado.

Una vez obtenido este tiempo, como tenemos el duty cycle del pulso, es decir, el porcentaje que el pulso toma valor unidad en un ciclo entero, tan sólo es necesario aplicar la siguiente ecuación para obtener el tiempo total del ciclo:

$$T_{ciclo} = \frac{T_{pulso}}{Dcycle * 10^6} \tag{5-2}$$

Donde:

- T_{ciclo} es el tiempo en el que transcurre el ciclo entero.
- T_{pulso} es el tiempo en el que se produce el pulso, es decir, la entrada se encuentra a valor 1.
- $Dcycle$ es el valor del duty cycle. En nuestro caso es 0.04 ya que es de un 4%.

Tras tener el tiempo del ciclo ya sólo es necesario realizar la inversa de este para conseguir la frecuencia, lo que ya se podría calcular la velocidad.

$$f = \frac{1}{T_{ciclo}} \tag{5-3}$$

Un ejemplo donde se aplica esta función es el siguiente:

```
float dur;
float tiempo;
float frecuencia;

void setup() {
  Serial.begin(115200);
  pinMode(PinOUT, INPUT);
}

void loop() {
  dur = pulseIn(PinOUT, HIGH, 6000000);   //Duracion de la activación de un pulso
  if (dur != 0) {
    tiempo = dur / (0.04 * 1000000);    //D.Cycle=4%
    frecuencia = 1 / tiempo;
  }
}
```

Cabe mencionar que `FreqCount` y `FreqCounter` pueden calcular sólo la frecuencia en números enteros. En cambio, `FreqMeasure` y `PulseIn` son capaces de calcular las frecuencias con una precisión de centésimas.

5.5 Incorporación de varios sensores

Se ha decidido separar este capítulo pues se considera importante la posibilidad de incluir varios sensores en un mismo Arduino, ya que para el objetivo propuesto, al fin y al cabo, lo que se debe es integrar varios sobre un UAV.

Sin embargo, al hacer uso del TIMER del microcontrolador, y tal y cómo está desarrollada las librerías, sólo es posible calcular la frecuencia para un sensor conectado a un PIN. Por lo que si se quiere hacer uso de varios sensores, habría que utilizar un Arduino para cada sensor. Quizás esto se vea un poco ilógico ya que para su montaje en un UAV el peso es muy influyente, y la inclusión de como mínimo tres tarjetas microcontroladores va a suponer un aumento de peso considerable.

Por tanto, se ha dado con una librería de uno de los mismos autores, denominada `FreqMeasureMulti` [10] que utiliza el mismo principio que la librería `FreqMeasure`, es decir, cuenta el número de ciclos de reloj que transcurren en un ciclo del tren de pulsos. Ahora bien, mediante esta librería es posible medir hasta ocho frecuencias simultáneamente. Pero ocurre un problema, y es que no es válida para nuestro Arduino, sino para Teensy.

Teensy es una plataforma de desarrollo de bajo coste, sobre unos ~25€, basada en un procesador ARM CortexM4 de 32 bits. Es muy parecida a Arduino ya que es posible programarla con el lenguaje Arduino y C, y además de ser muy potente, es compatible con la mayoría de las librerías de este último. Por otro lado, tiene la ventaja de que tiene un menor peso (3 gramos respecto 20 del Arduino), por lo que es favorable para su montaje en vehículos aéreos.

Existen varios modelos, Teensy 2.0, Teensy++2.0, Teensy LC y Teensy 3.2, donde este último es la versión más actual y más potente del dispositivo. Se muestra a continuación:

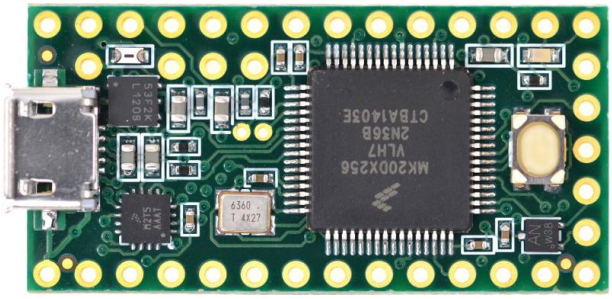


Figura 5-4: Parte frontal de la placa de desarrollo Teensy 3.2

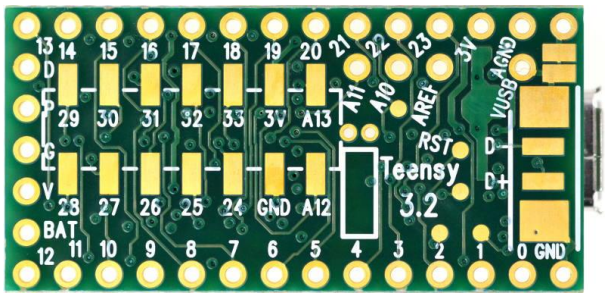


Figura 5-5: Parte trasera de la placa de desarrollo Teensy 3.2

5.5.1 Funcionamiento de FreqMeasureMulti

A continuación se explicará el funcionamiento de esta librería para facilitar la posible integración en trabajos futuros.

FreqMeasureMulti utiliza las mismas funciones que FreqMeasure, excepto begin() ya que se debe especificar en la misma el PIN en el que se desea leer la frecuencia. Con la lectura de un ejemplo de código para esta librería en el que se calculan tres frecuencias procedentes de tres pines distintos y después de haber entendido el funcionamiento de `FreqMeasure` se entiende fácilmente la librería. Para ello se muestra el siguiente.

```
#include <FreqMeasureMulti.h>

FreqMeasureMulti freq1;
FreqMeasureMulti freq2;
FreqMeasureMulti freq3;

void setup() {
  Serial.begin(115200);
  freq1.begin(6);
  freq2.begin(9);
  freq3.begin(10);
}

float sum1=0, sum2=0, sum3=0;
int cont1=0, cont2=0, cont3=0;
elapsedMillis timeout;
```

```
void loop() {
  if (freq1.available()) {
    sum1 = sum1 + freq1.read();
    cont1 = cont1 + 1;
  }
  if (freq2.available()) {
    sum2 = sum2 + freq2.read();
    cont2 = cont2 + 1;
  }
  if (freq3.available()) {
    sum3 = sum3 + freq3.read();
    cont3 = cont3 + 1;
  }
  //Guarda los resultados en una variable cada 500 milisegundos
  if (timeout > 500) {
    if (cont1 > 0) {
      frecuencia_1 = freq1.contToFrequency(sum1 / cont1);
    } else {
      frecuencia_1 = 0;
    }

    if (cont2 > 0) {
      frecuencia_2 = freq2.contToFrequency(sum2 / cont2);
    } else {
      frecuencia_2 = 0;
    }

    if (cont3 > 0) {
      frecuencia_3 = freq3.contToFrequency(sum3 / cont3);
    } else {
      frecuencia_3 = 0;
    }
    sum1 = 0;
    sum2 = 0;
    sum3 = 0;
    cont1 = 0;
      cont2 = 0;
    cont3 = 0;
    timeout = 0;
  }
}
```

6 INTEGRACIÓN EN UAV

Es posible calcular la velocidad respecto de tierra a partir de las señales de GPS, pero como este es un método de navegación no autónomo, que necesita de cobertura de los satélites, se intenta buscar otra solución para el cálculo de la velocidad. El sensor autónomo más utilizado en un avión, o en este caso, en un UAV, es el tubo de Pitot que mide la presión dinámica del aire, a partir de la cual es posible calcular la velocidad respecto del aire, $\vec{V}_{TAS}$. Para saber cuál es la velocidad respecto de tierra, $\vec{V}_{GS}$, es necesario sumar la velocidad del viento, $\vec{V}_{WIND}$, de forma vectorial tal y como se muestra en la siguiente figura.

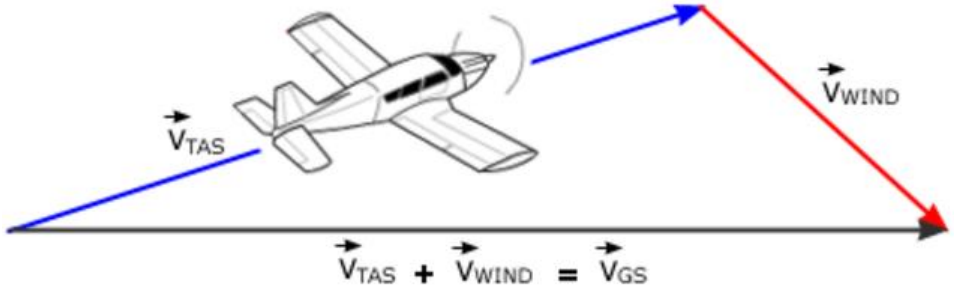


Figura 6-1: Triángulo de velocidades

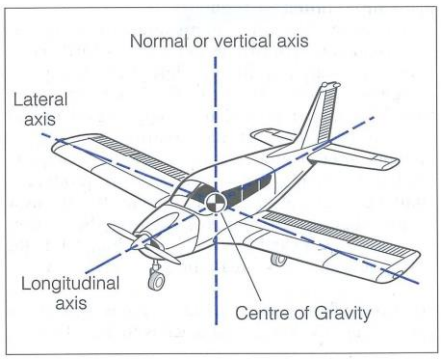


Figura 6-2: Ejes cuerpo de un vehículo aéreo

Sin embargo, la mayoría de tubos de Pitot son insensibles a las componentes de velocidad normales al eje longitudinal del avión o UAV, como el resbalamiento. Para solucionar esto, se pretende utilizar las mediciones del radar Doppler.

Hay que destacar, que el sensor Doppler del que disponemos permite medir como se ha dicho antes la componente de la velocidad en una sola dirección, y además no es posible medirla cuando la superficie de la antena del sensor es colineal con la dirección del movimiento.

Por tanto, se buscará la forma de incorporación de los sensores necesarios para que sea posible el cálculo de las tres componentes de la velocidad, en tres ejes diferentes, tal y como muestra la siguiente figura, siendo V_D la velocidad de resbalamiento o deriva (o de drag), V_H la velocidad horizontal en dirección longitudinal del avión y V_V la velocidad vertical.

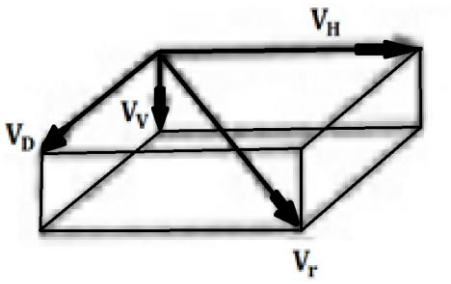


Figura 6-3: Descomposición de velocidad

Para medir ambas componentes horizontales de la velocidad, tanto la de dirección longitudinal como la de resbalamiento, es necesario tener al menos dos haces radiando energía hacia la superficie de la Tierra, tal y como se muestra en la siguiente figura.

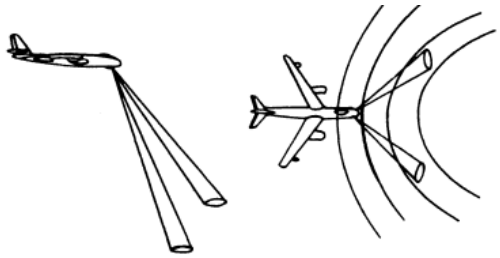


Figura 6-4: Configuración de Doppler con dos haces

Sin embargo, una de las técnicas más utilizadas para la configuración de varios sensores Doppler es la llamada formación JANUS [11] que consiste en utilizar haces opuestos, donde lo más común es emplear tres o cuatro. Las ventajas de este sistema son:

- Permite medir la componente vertical de la velocidad, cuando no era posible con dos haces. De hecho, si se incorporan dos haces que no cumplen la formación Janus, el ratio de descenso o de subida debe calcularse con otros sensores, como un sensor barométrico. Sin embargo, esto no sería muy preciso para navegación de pequeños UAV's y en interior de edificios por lo que se descarta esta última idea.
- Permite cancelaciones de la componente vertical cuando los haces delanteros y traseros se combinan para medir la componente horizontal de la velocidad. Por ejemplo, cuando el vehículo cabecea.
- Las mediciones de las velocidades horizontales son menos sensibles a errores.

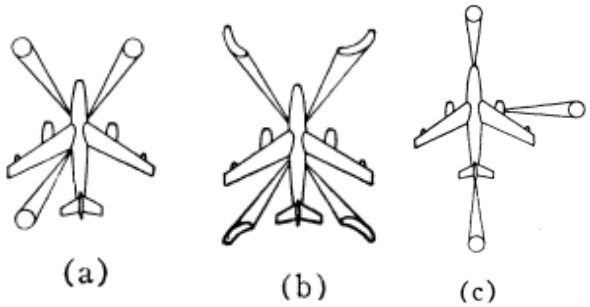


Figura 6-5: Tipos de formación Janus

En una configuración Janus, la desviación de frecuencia Doppler obtenida con el haz delantero es posible sumarla a la obtenida con el haz trasero. En principio, para simplificar los cálculos, consideraremos la condición de que no existe ni balanceo (roll) ni cabeceo (pitch) en el UAV. De modo que las frecuencias Doppler delanteras y traseras son iguales para el total del desplazamiento Doppler procedentes de dos haces, y esto toma forma en la siguiente ecuación, muy parecida a la 4-1.

$$f_d = 4V\frac{F_t}{c}\cos\gamma \tag{6-1}$$

Que como vimos anteriormente puede transformarse mediante la relación $\lambda = \frac{c}{f}$ en:

$$\mathrm{f_d} = 4\frac{V}{\lambda}\cos\gamma \tag{6-2}$$

La formación particular Janus de tres haces [12] [13] que se muestra en la figura 6-5(a) es llamada configuración λ y es la más óptima para vehículos aéreos de ala fija. Por otro lado, la formación de la figura 6-5(c) llamada configuración T es la más utilizada para helicópteros. Aún así, nos centraremos en principio en la llamada λ que es usada en un gran número de sistemas Doppler modernos y es quizás la configuración de haces más óptima que existe. Es posible calcular la componente de velocidad longitudinal, la de deriva o de resbalamiento y la vertical. Para la condición simplificada tomada anteriormente, las expresiones matemáticas que permiten calcular las velocidades se muestran a continuación.

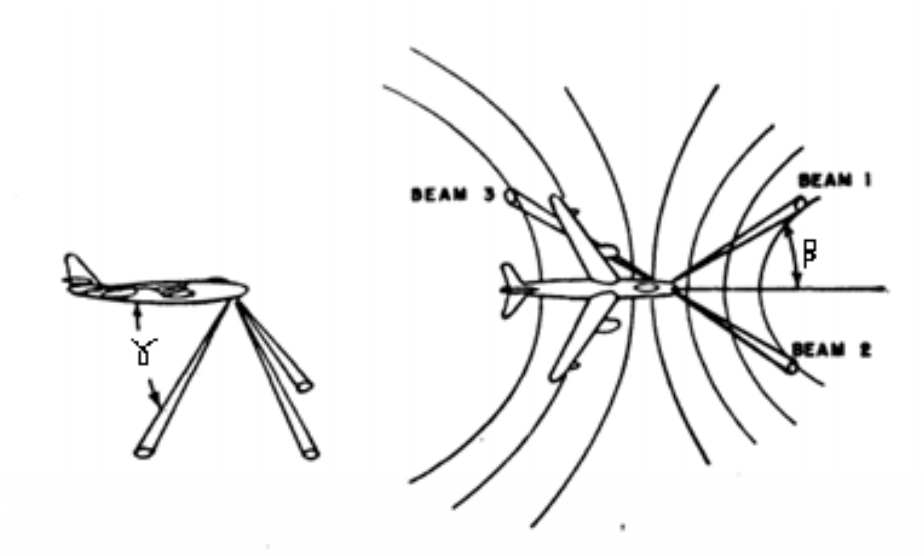


Figura 6-6: Configuración JANUS de tres haces, la más óptima

$$\mathrm{V}_H = \frac{(f_{d1} - f_{d3})}{4\lambda\cos\gamma\cos\beta} \tag{6-3}$$

$$\mathrm{V}_D = \frac{(f_{d2} - f_{d1})}{4\lambda\cos\gamma\sin\beta} \tag{6-4}$$

$$\mathrm{V}_V = \frac{(f_{d3} + f_{d2})}{4\lambda\sin\gamma} \tag{6-5}$$

Donde:

- V_H es la componente de la velocidad en dirección al morro del UAV, positiva hacia delante.
- V_D es la componente de la velocidad en dirección perpendicular al morro o de resbalamiento positiva hacia la derecha.
- V_V es la componente vertical de la velocidad, positiva hacia abajo.
- f_{d1} la desviación de frecuencia Doppler del haz 1.
- f_{d2} la desviación de frecuencia Doppler del haz 2.
- f_{d3} la desviación de frecuencia Doppler del haz 3.
- γ el ángulo que forma la superficie terrestre con la dirección de propagación del haz.
- β ángulo de azimut del haz
- λ longitud de onda de la señal propagada.

Cabe destacar que en nuestro caso, el sensor siempre saca una desviación de frecuencia positiva, por lo que no es posible saber si el vehículo va hacia delante o hacía atrás o si desciende o asciende, por lo que las ecuaciones deben ser modificadas. Para ello se simplifica y se considera que el móvil va a ir siempre hacia delante y que la deriva no va a ser muy grande. Si se desplaza hacia atrás, V_H será negativo. Lo mismo ocurre con V_D, cuya dirección, positiva o negativa, deberá ser averiguada de otra manera. Las ecuaciones finales serían:

$$V_H = \frac{(f_{d1}+f_{d3})}{4\lambda \cos\gamma \cos\beta} \tag{6-6}$$

$$V_D = \frac{(f_{d2}-f_{d1})}{4\lambda \cos\gamma \sin\beta} \tag{6-7}$$

$$V_V = \frac{(f_{d3}+f_{d2})}{4\lambda \sin\gamma} \tag{6-8}$$

Cada haz en particular debe transmitir a un cierto ángulo, γ, de entre 60º y 70º. Esto lleva a un compromiso ya que si el ángulo γ es cercano a 90º, la frecuencia Doppler es aproximadamente nula, como explicamos en capítulos anteriores. Ahora bien, si el ángulo es demasiado pequeño, las ondas impactarán contra la superficie terrestre con un ángulo muy pequeño, haciendo que las señales se reflejen lejos de la aeronave, lo que resulta en señales recibidas muy débiles. Una de las soluciones más óptimas adoptadas por la mayoría de equipos de desarrollo es un ángulo γ de 70º.

Por otro lado, la elección del ángulo β depende en gran medida de la aplicación del sistema y de ciertas consideraciones técnicas tales como la sensibilidad del ángulo de deriva o resbalamiento y el retorno de la señal. Aún así, valores típicos se encuentran en el intervalo de 20º a 90º.

Como hemos aplicado la condición de que no existe ni balanceo ni cabeceo, las velocidades pueden ser definidas directamente respecto a Tierra. Sin embargo, aún siendo esto una simplificación, es posible cumplirlo siempre y cuando se realice una estabilización de la antena mediante una plataforma giroestabilizada o una estabilización de los datos obtenidos, de modo que los datos de las frecuencias no sean afectados por el cambio de orientación de la aeronave.

Si no se cumple ninguna de las dos ideas mencionadas en el párrafo anterior, existirá cabeceo y balanceo de modo que las velocidades calculadas se encontrarán sobre los ejes cuerpo del UAV. Por ello, para realmente saber cuál es la velocidad respecto a Tierra debería transformarse a otro sistema de ejes como el siguiente.

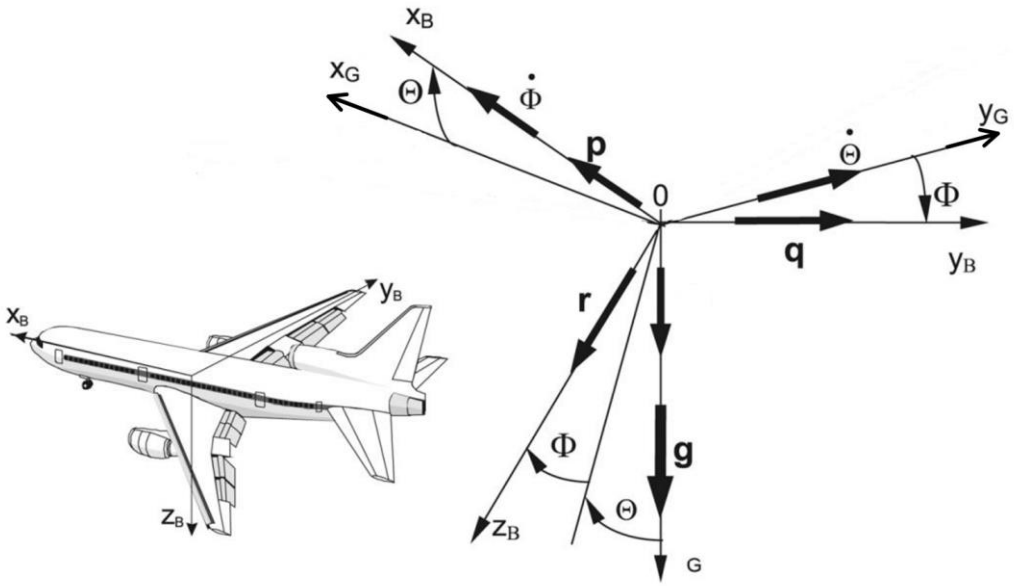


Figura 6-7: Orientación de la aeronave

En este caso, tras obtener las tres componentes de la velocidad en ejes cuerpo: $V = \begin{bmatrix} V_H \\ V_D \\ V_V \end{bmatrix}$ sería posible pasarlo a velocidades respecto a Tierra tomando los ángulos de cabeceo Θ (positivo cuando el morro apunta hacia arriba) y de balanceo Φ (positivo cuando el ala derecha apunta hacia arriba) de la aeronave. Estos ángulos son posible obtenerlos a través de la unidad de medidas inerciales equipada a bordo. De esta forma, mediante una matriz de rotación sería posible transformar desde los ejes cuerpo a velocidades respecto a Tierra, como se muestra a continuación.

La velocidad respecto a Tierra cumpliría la ecuación $V_{tierra} = C_b^g \cdot V$, siendo C_b^g la matriz de rotación que transforma desde los ejes cuerpo a ejes de gravedad. Esta matriz es ortogonal y es la siguiente:

$$C_b^g = \begin{bmatrix} \cos\theta & \sin\theta\sin\Phi & \cos\Phi\sin\theta \\ 0 & \cos\Phi & -\sin\Phi \\ -\sin\theta & \sin\Phi\cos\theta & \cos\Phi\cos\theta \end{bmatrix}$$

Por tanto, una vez obtenido los ángulos de orientación del vehiculo mediante la IMU y las componentes de la velocidad mediante los sensores Doppler, la velocidad respecto a Tierra se puede obtener aplicando la la ecuación anterior, quedando entonces:

$$V_{tierra} = \begin{bmatrix} \cos\theta \cdot V_H & \sin\theta\sin\Phi \cdot V_D & \cos\Phi\sin\theta \cdot V_V \\ 0 \cdot V_H & \cos\Phi \cdot V_D & -\sin\Phi \cdot V_V \\ -\sin\theta \cdot V_H & \sin\Phi\cos\theta \cdot V_D & \cos\Phi\cos\theta \cdot V_V \end{bmatrix} \quad (6\text{-}9)$$

Esta velocidad es respecto a Tierra y una vez obtenida es posible realizar una integración para calcular el desplazamiento del móvil.

7 PRUEBAS EXPERIMENTALES

Tras haber realizado la programación del microcontrolador, en este capítulo se hará incapié en las pruebas experimentales que se han hecho en la habitación de detección y seguimiento.Para ello se ha colocado el conjunto del sensor con el Arduino encima de un robot cuyo movimiento puede controlarse con un mando a distancia. La habitación de "Motion-Tracking" utilizada se encuentra situada en el Centro de Postgrado de la Universidad Pablo de Olavide de Sevilla.

Por un lado tendremos la velocidad calculada mediante el sensor Doppler, que será como ya hemos comentado anteriormente, la velocidad longitudinal del vehículo ya que $\theta = 0$. Un aspecto a mencionar es que el sensor del que disponemos no puede dar valores de velocidad negativos, por lo que no se puede saber si el móvil va hacia delante o hacia atrás.

Por otro lado, tendremos la velocidad calculada mediante los sensores de la sala. Sin embargo, en éste no es posible calcular la velocidad longitudinal, sino todas las componentes de la velocidad. A pesar de haber intentado realizar desplazamientos en línea recta, ha habido que tomar giros que posiblemente pueden provocar que las medidas sean muy diferentes con ambos sensores en ciertos instantes. Por tanto, para su comparación con los valores del sensor Doppler se ha tomado el módulo de la velocidad.

Después de haber recogido los datos de las pruebas, se calcularán una serie de errores para conocer cuanto de buena es la medición del sensor y si es factible su uso para el cálculo de velocidad en UAV's.

Se dividirá el capítulo para cada uno de los scripts que tenemos.

7.1 Método FreqCounter

En primer lugar presentamos la gráfica de las velocidades calculadas con este código junto con las calculadas por los sensores ópticos de la cámara.

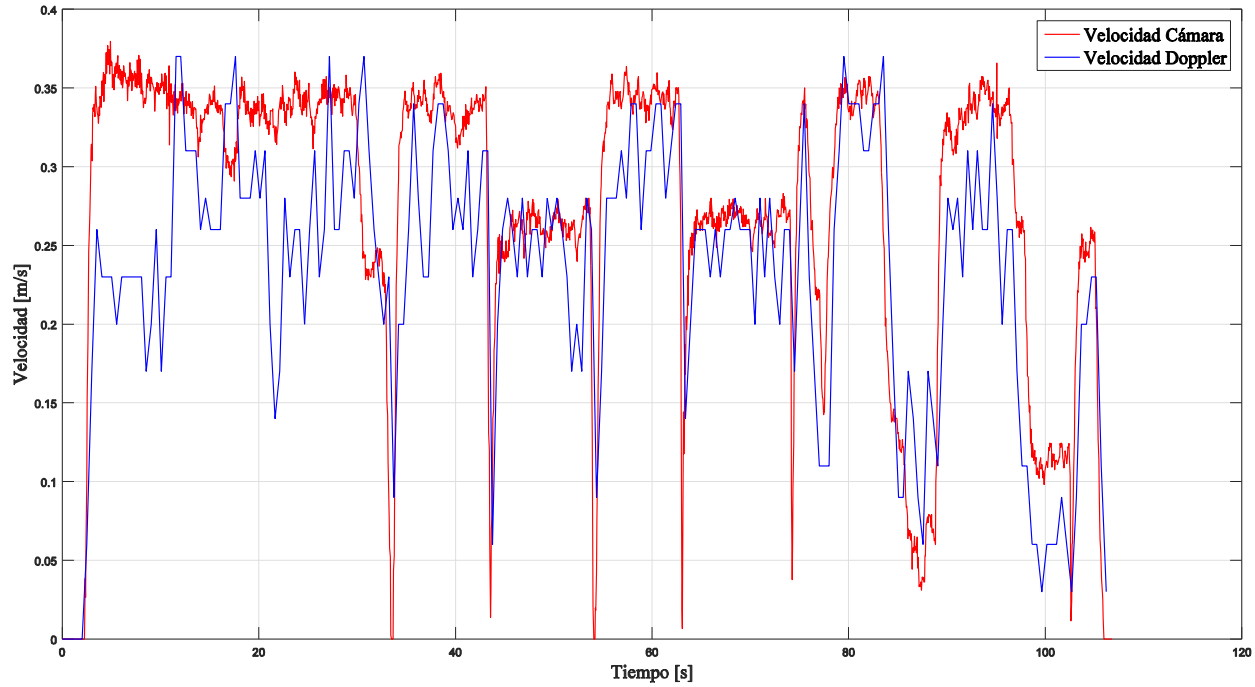


Figura 7-1: Representación de velocidades mediante FreqCounter

Como podemos observar el sensor Doppler sigue bastante los cambios de velocidad, sin embargo, no permite dar valores precisos del valor en sí. Cabe mencionar que durante alrededor de los primeros 40 segundos se hicieron bastantes giros y es por ello que las velocidades no coinciden demasiado.

Se han tomado unas 2000 muestras mediante la cámara de seguimiento y unas 212 con el sensor Doppler en el mismo tiempo de ensayo. Por lo que los valores más precisos contienen unas 10 veces más muestras que los tomados con el dispositivo disponible. Por tanto, para poder calcular el error con mayor precisión, se han modificado los datos obtenidos de forma que sea posible reducir el número más elevado de ellos al menor número.

Como los datos están sacados en tiempos diferentes en ambas formas, se ha tomado el tiempo de referencia de la adquisición del sensor Doppler. De esta forma, si los datos se recogen por la cámara de seguimiento cada décima de segundo y el sensor lo hace cada segundo, se ha realizado una media de los valores obtenidos por la cámara antes de llegar al tiempo de referencia del Doppler, es decir, la media de los valores que adquiere la cámara en cada décima, y se ha hecho que ese valor medio sea la velocidad recogida por el seguidor de movimiento en la unidad de segundo.

De este modo, la representación de las velocidades quedaría de la siguiente forma:

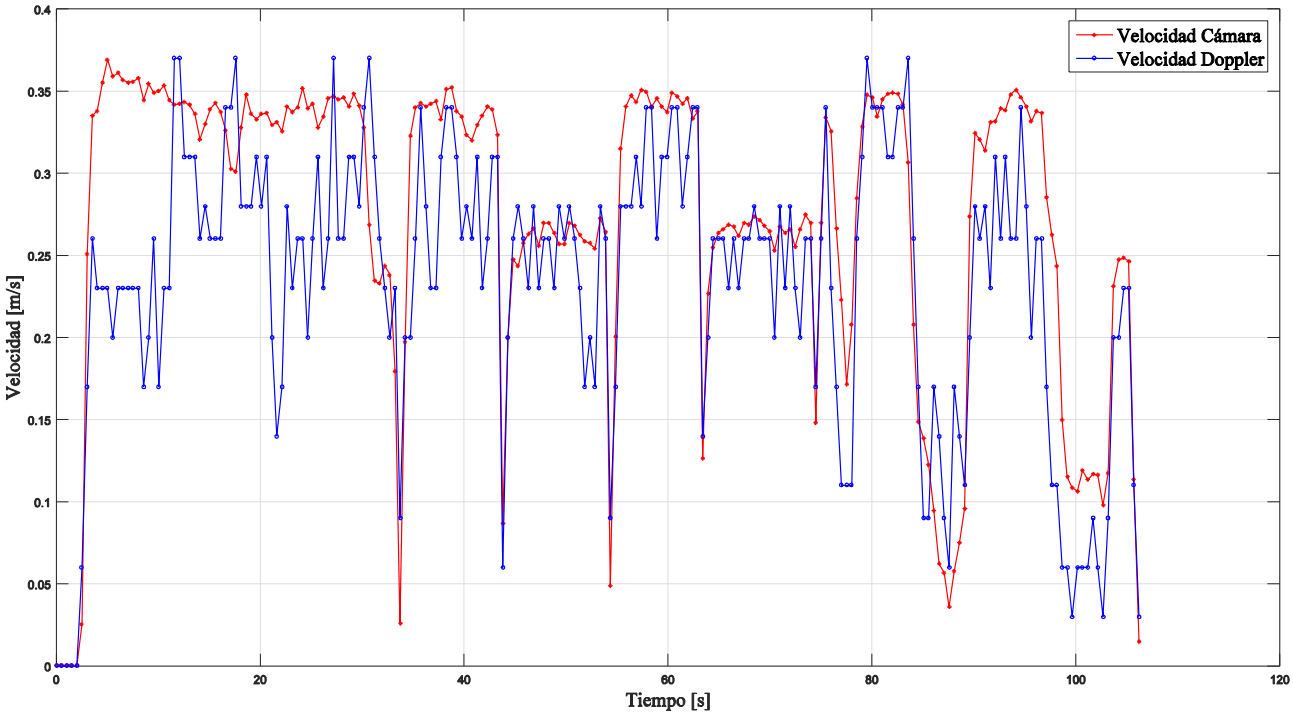


Figura 7-2: Representación velocidades mediante FreqCounter con reducción de muestras

Como observamos, se han reducido el número de muestras de los datos recogidos por el sensor. Aunque se falsean algo los datos, es una manera eficaz para comparar los datos obtenidos de ambos sensores. Se muestra a continuación la representación del error durante el tiempo de ensayo.

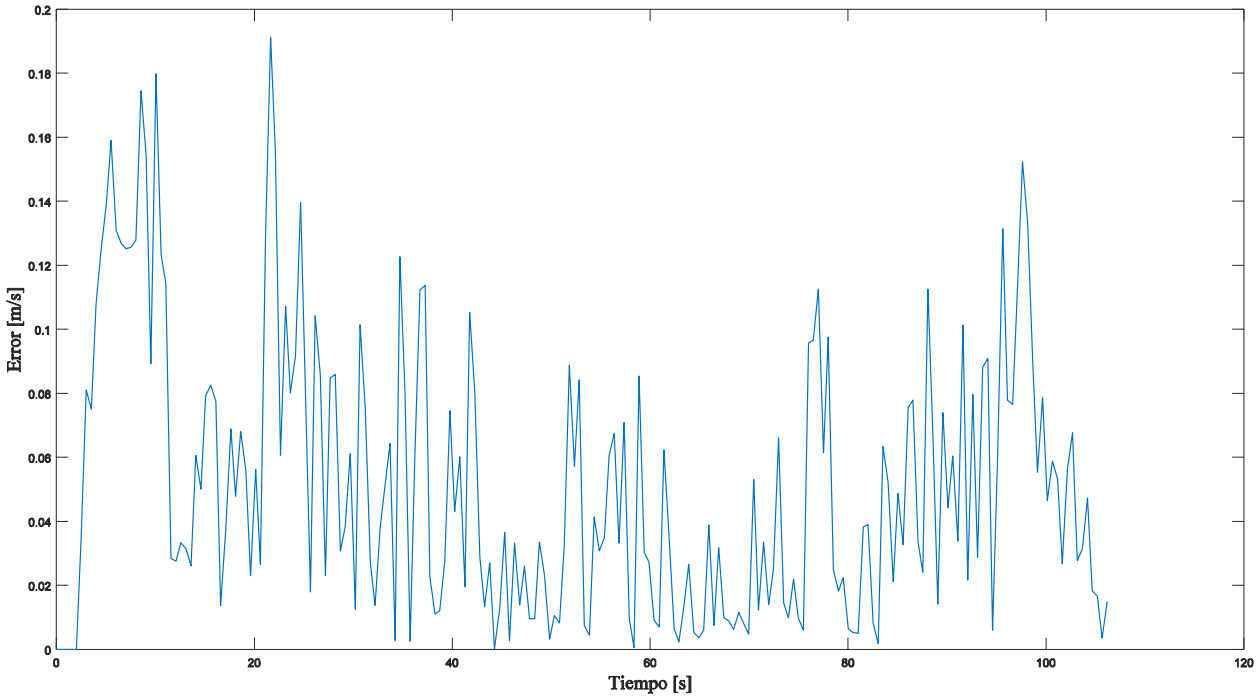


Figura 7-3: Representación del error con el método FreqCounter

Para los errores se obtienen los siguientes valores:

Error medio (m/s)	Desviación típica del error (m/s)
0.0516	0.0430

Tabla 1: Valores estadísticos de los errores de FreqCounter

7.2 Método FreqCount

En este caso, seguimos representando las velocidades, donde podemos observar que al igual que el método anterior el sensor Doppler sigue bien los cambios en la velocidad, pero no permite calcularla de forma precisa.

Cabe mencionar que las velocidades que aparecen tras el segundo 130 en la gráfica es debido al movimiento de personas en la habitación tras la prueba ya que el sensor Doppler es muy sensible a interferencias.

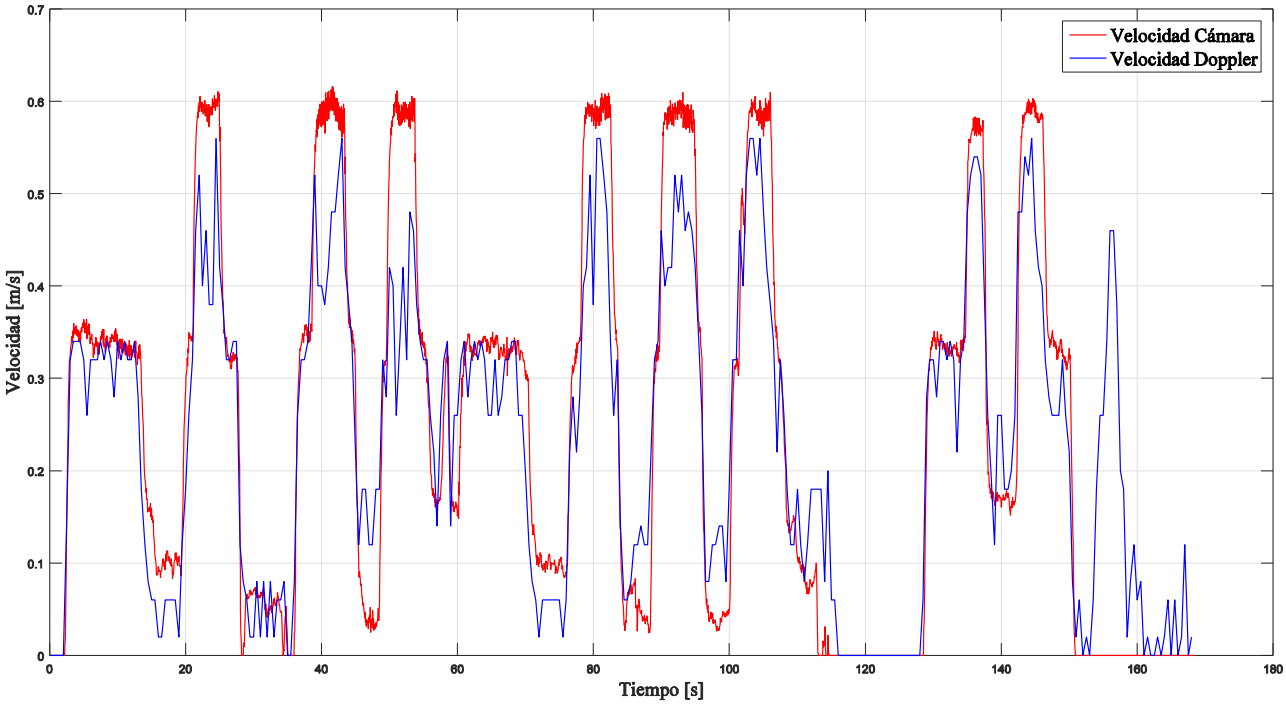


Figura 7-4: Representación de velocidades mediante FreqCount

Procediendo de la misma forma que explicamos en el método anterior, reducimos el número de muestras quedando de la siguiente forma.

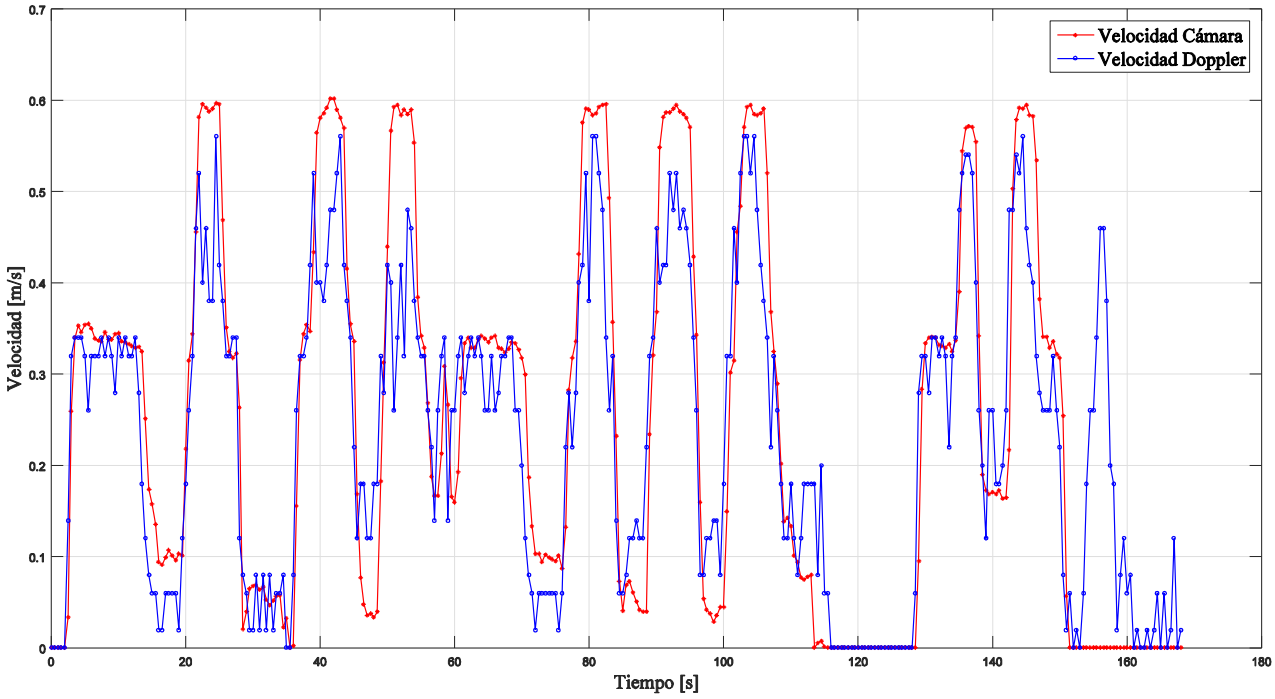


Figura 7-5: Representación velocidades mediante FreqCount con reducción de muestras

Para el cálculo del error se han eliminado las interferencias finales comentadas.

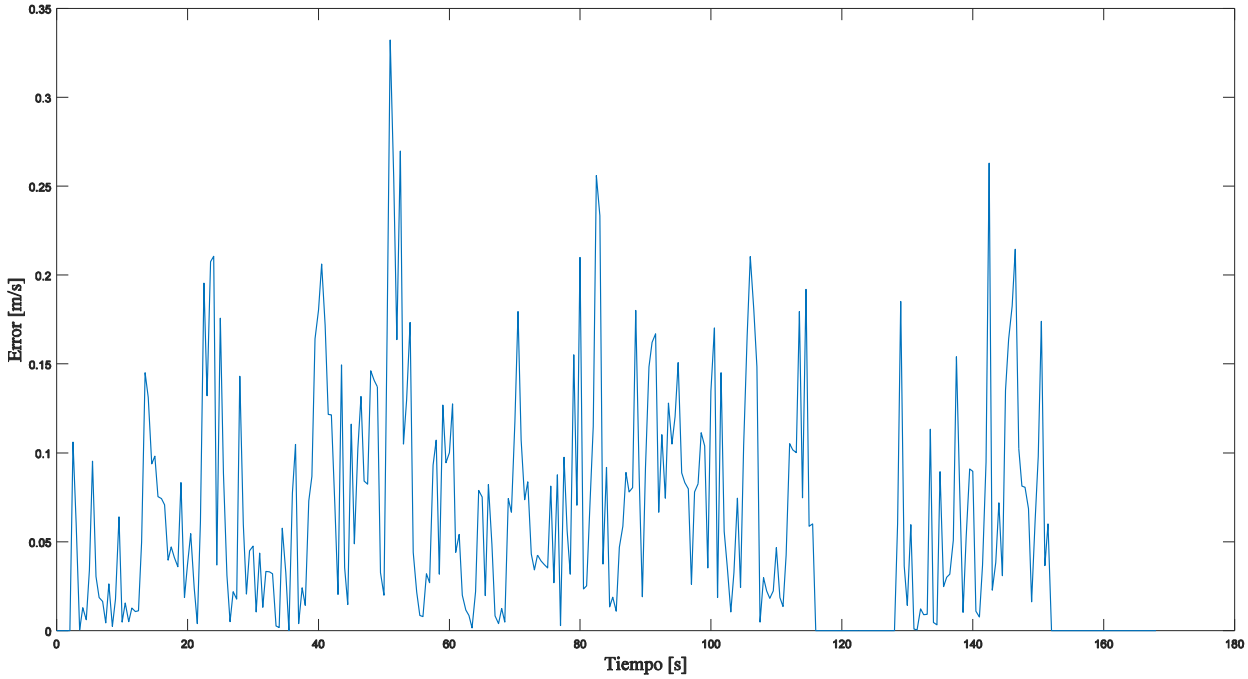


Figura 7-6: Representación del error con el método FreqCount

Para los errores se obtienen los valores de la siguiente tabla:

Error medio (m/s)	Desviación típica del error (m/s)
0.0663	0.0629

Tabla 2: Valores estadísticos de los errores de FreqCount

7.3 Método FreqMeasure

Este es el método que se supone debería ser el más exacto. Como podemos ver en la siguiente representación, no es lo suficientemente preciso tal y como esperábamos. Por tanto quizás haya que desechar la idea de utilización de este método.

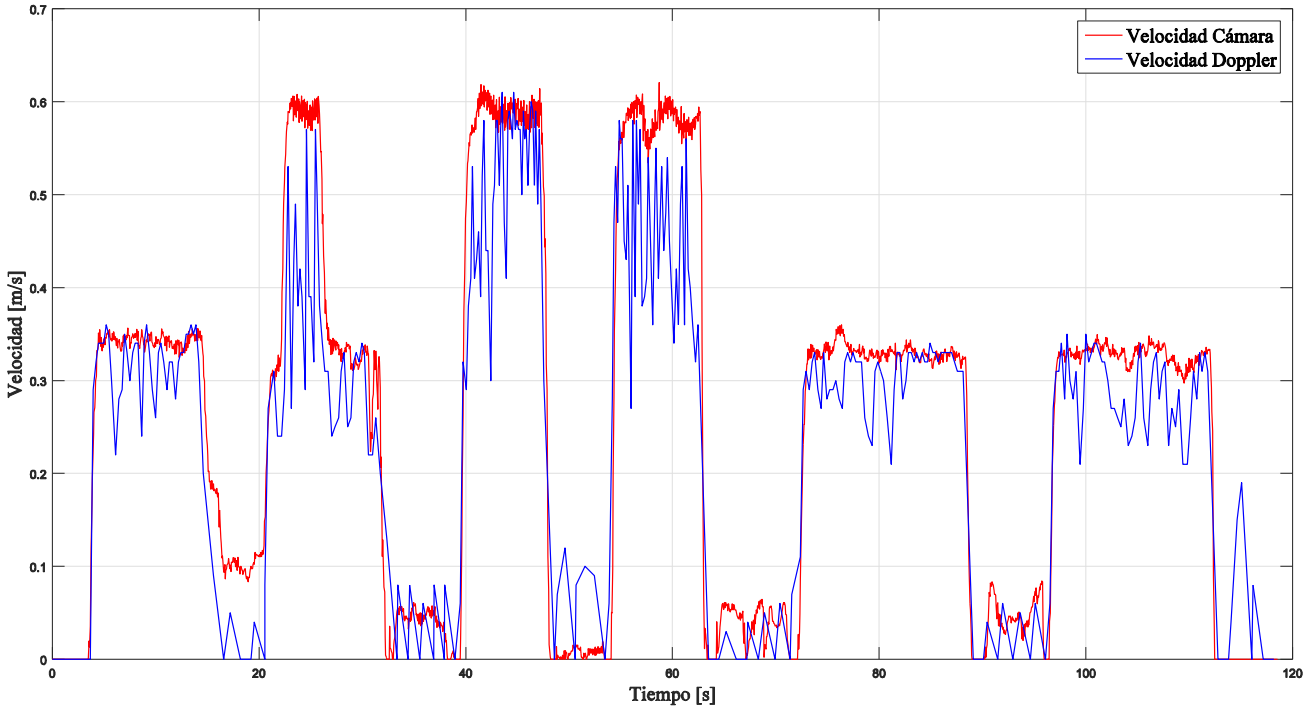


Figura 7-7: Representación de velocidades mediante FreqMeasure

Procediendo de la misma forma anterior y eliminando las interferencias finales en los errores, se obtienen las siguientes gráficas.

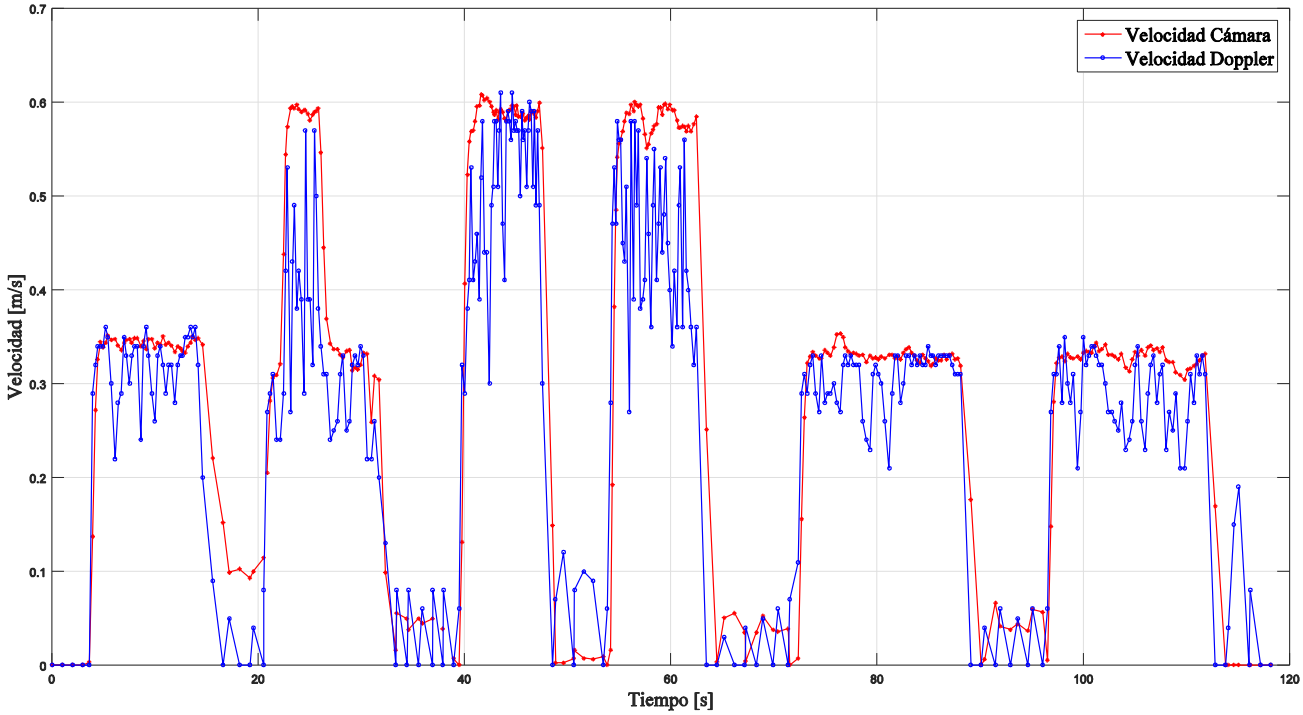


Figura 7-8: Representación velocidades mediante FreqMeasure con reducción de muestras

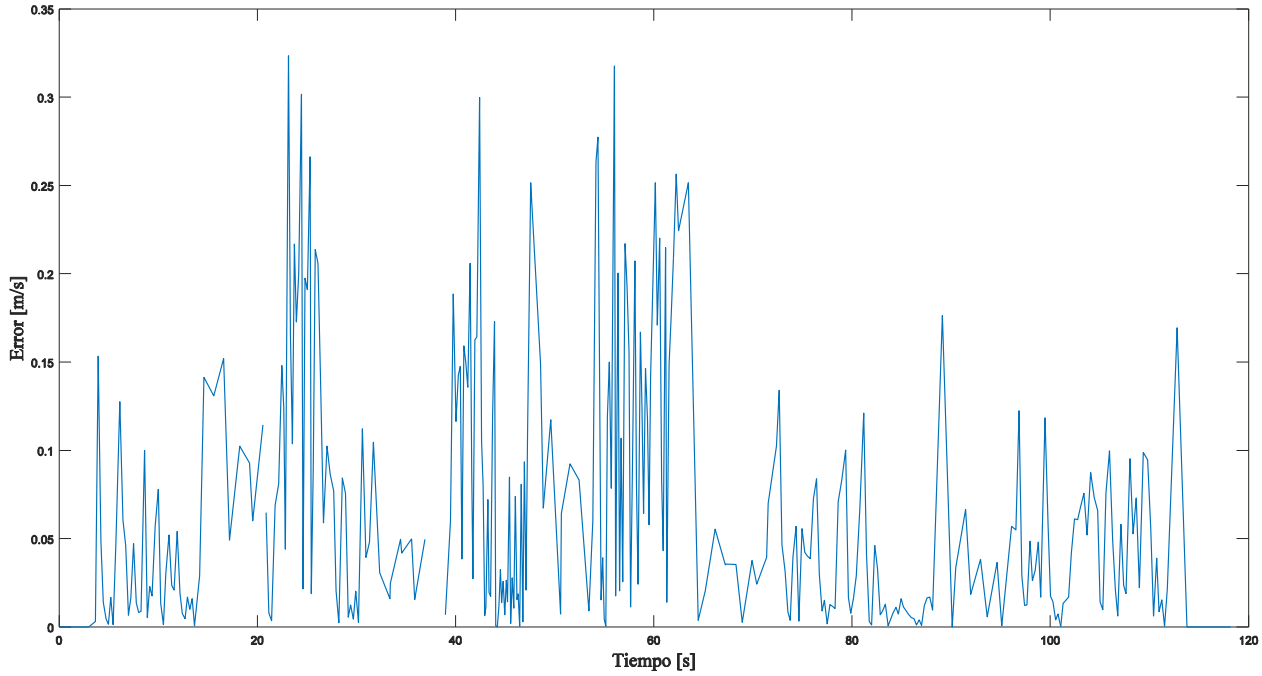


Figura 7-9: Representación del error con el método FreqMeasure

Para los errores se obtienen los siguientes valores:

Error medio (m/s)	Desviación típica del error (m/s)
0.0647	0.0694

Tabla 3: Valores estadísticos de los errores de FreqMeasure

7.4 Método PulseIn

Para este método se aprecia que las medidas contienen mucho ruido, por lo que se rechaza la idea directamente de utilización de esta forma de adquirir la velocidad y no se procede a calcular los errores.

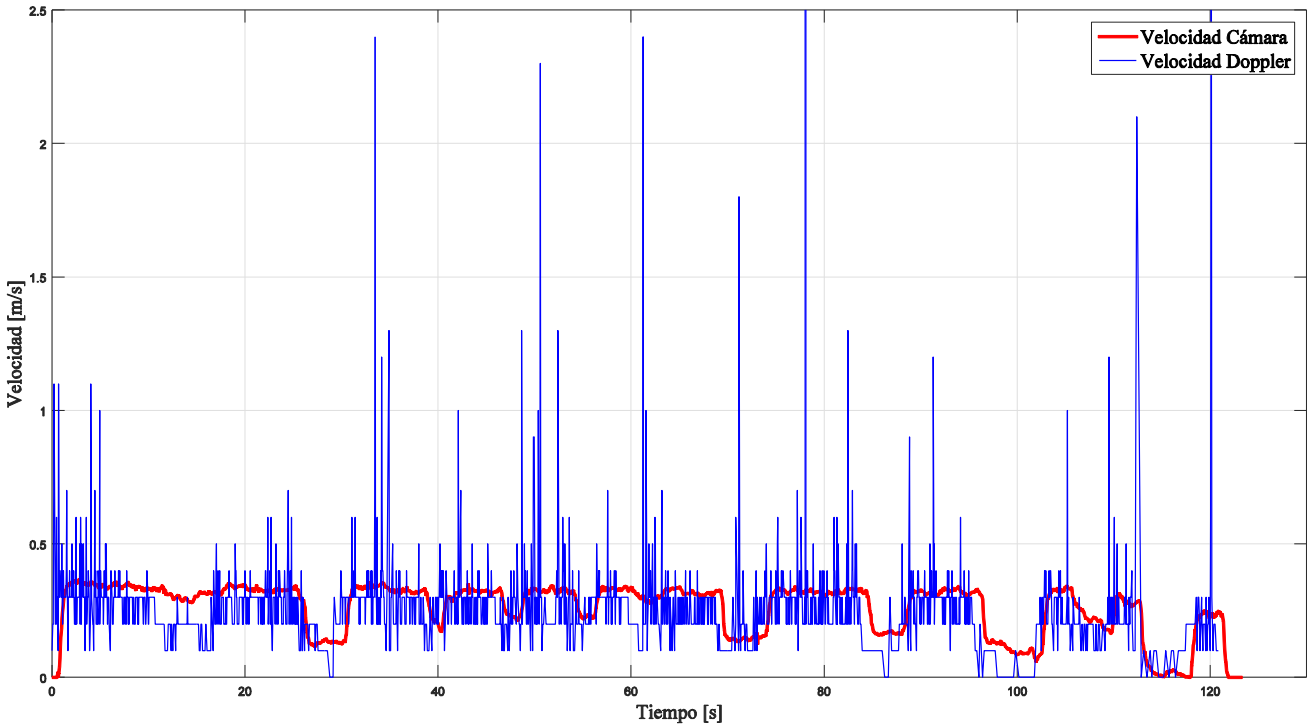


Figura 7-10: Representación de velocidades mediante PulseIn

8 CONCLUSIONES

Una vez realizadas las pruebas y tras haber recogido la información necesaria de ambos sensores y todos los métodos, se obtiene la siguiente tabla de errores.

MÉTODO	ERROR MEDIO (M/S)	DESVIACIÓN TIPICA (M/S)
FREQCOUNTER	0.0516	0.0430
FREQCOUNT	0.0663	0.0629
FREQMEASURE	0.0647	0.0694
PULSEIN	NO PROCEDE	NO PROCEDE

Tabla 4: Recopilación de medias y desviaciones típicas de errores

Se llega a la conclusión de que la forma con la que se permite estimar la velocidad con mayor precisión es con la librería `FreqCounter`. A pesar de que `FreqMeasure` era la esperada para ser la de mejor precisión, no ha sido así. Tras la presentación de los resultados, tal y cómo se encuentra programado el sensor y vistas las velocidades en las que se ha manejado el robot, se llega a la conclusión de que no se podría utilizar para la navegación en interiores ya que es necesario de unos requerimientos más exigentes de lo que se ha conseguido. Un error de 5cm/s en un movimiento en el que como máximo se ha llegado sobre los 60cm/s es demasiado grande para el objetivo que se busca ya que implica una imprecisión de aproximadamente un 8%, que crece al ser integrado para calcular el desplazamiento.

Esto podría haber sido previsto al inicio, ya que como comentamos anteriormente, el sensor del que se disponía no estaba diseñado para cálculo de velocidad sino para detección de movimiento, tal y como nos especificaba el datasheet. Aún así, el error obtenido no está mal si tenemos en cuenta el bajo peso y precio del sensor, además del poco procesamiento añadido que ha necesitado. Por otro lado, la sensibilidad del mismo podría haber sido ajustada con el potenciómetro que incluye pero por falta de disponibilidad de tiempo en la cámara de detección y seguimiento esto no se ha llegado a hacer y se ha dejado a un valor medio.

Se había mencionado anteriormente la posibilidad de integrar varios sensores en un mismo controlador Teensy, permitiendo así aligerar peso en el UAV y medir varias frecuencias en la misma placa. La librería que posibilitaría esto funcionaba exactamente de la misma forma que `FreqMeasure`. Sin embargo, visto los resultados obtenidos no merecería la pena integrarlos ya que no se conseguiría la precisión exigida.

Aún así, sería posible en un futuro desarrollo de este trabajo en el que se implemente el sensor Doppler junto con una unidad de medidas inerciales. De esta forma, primero, sería posible mejorar la precisión de la estimación de la posición mediante implementación con un filtro de Kalman de ambos sensores de manera que disminuya el error, y segundo, la posibilidad de calcular el signo de la velocidad, lo cuál es una incógnita tal y como se comentaba en capítulos anteriores. Si los resultados no mejoraran, sería interesante también intentar utilizar `FreqMeasure` junto con Teensy en otros sensores Doppler que si que estén diseñados para el cálculo de la velocidad. Si se obtienen buenos resultados de una u otra forma se podría implementar una configuración Janus en un UAV tanto λ como T, dependiendo de si se trataría de un UAV de ala fija o de ala rotatoria.

9 ANEXO A: PROGRAMACIÓN EN ARDUINO

El lenguaje de programación de Arduino está basado en lenguaje C, por lo que muchas de las funciones de este mismo son utilizadas en Arduino. En primer lugar, es necesario de la utilización del IDE de Arduino, que es posible descargar en su página oficial, pues es este software el que va a permitir cargar el programa que escribamos en la placa.

Ahora bien, este programa posee una estructura básica que es bastante simple y divide la ejecución en dos partes: setup y loop. Por otro lado, cada instrucción debe acabar con un ; .

- La función `setup()` incluye la declaración de las variables, y en ella se debe escribir las funciones de configuración, pues esta sólo se ejecutará una única vez al inicio del programa. Es empleada para configurar el pinMode.
- La función `loop()` incluye el código que debe ser ejecutado continuamente, pues como su propio nombre indica repite todas las funciones que existan dentro de esta en un bucle infinito.

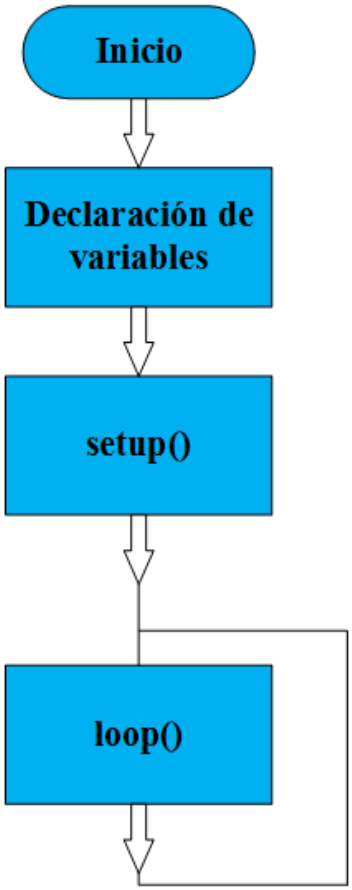


Figura 9-1: Diagrama de flujo de programa básico en Arduino

Cada una de estas debe ser establecida entre llaves {}. Por otro lado, es posible crear funciones independientes de estas, fuera de la estructura básica, que puedan ser llamadas en las funciones setup o loop. A continuación se muestra un ejemplo de esto:

```
void setup() {
   inMode(pin, OUTPUT);          // Establece 'pin' como salida
}
void loop() {
   digitalWrite(pin, HIGH);      // Activa 'pin'
   delay(1000);                  // Pausa un segundo
   digitalWrite(pin, LOW);       // Desactiva 'pin'
   delay(1000);
}
```

En la tabla siguiente se recogen las funciones utilizadas en nuestro programa:

Función	**Descripción de la función**
`#include <librería>`	Incluye al programa la librería especificada, donde librería es el nombre de la misma. Hay que tener en cuenta que debe estar instalada previamente a la ejecución de esta llamada. Normalmente es utilizada al inicio del código.
`pinMode(pin, mode)`	Usada en la función `setup()` para configurar un pin dado para comportarse como `INPUT` o `OUTPUT`
`Serial.begin(rate)`	Establece la tasa de datos en bits por segundo (baudios) para la transmisión de datos por el puerto serie, donde rate es dicha tasa.
`digitalWrite(pin, value)`	Introduce un nivel alto (`HIGH`) o bajo (`LOW`) en el pin digital especificado.
`delayMicroseconds(time)`	Realiza una pausa del programa la cantidad de tiempo especificado en microsegundos.
`millis()`	Devuelve la cantidad en milisegundos que lleva al placa Arduino ejecutando el programa actual como un valor long unsigned.
`micros()`	Devuelve la cantidad en microsegundos que lleva al placa Arduino ejecutando el programa actual como un valor long unsigned.
`Serial.print()`	Imprime datos al puerto serie
`Serial.println()`	Imprime datos al puerto serie, retornando a principio de línea una vez ejecutado.

Tabla 5: Relación de funciones utilizadas en el código de Arduino

Además de estas funciones también es posible la utilización de los operadores suma, resta, división y multiplicación.

Una vez entendido lo anterior, se explicarán los diferentes tipos de variables que existen, lo cual se asemeja completamente al lenguaje C. Una variable debe ser declarada antes de su utilización y manejo.

Tipo de dato	Descripción
int	Almacena un valor entero de 16 bits con un rango de -32.767 a 32.767.
unsigned int	Almacena un valor entero positivo de 16 bits con un rango de 0 a 65.535.
long	Almacena un valor entero de 32 bits con un rango de -2.147.483.648 a 2.147.483.648.
unsigned long	Almacena un valor entero positivo de 32 bits con un rango de 0 a 4.294.967.295.
float	Almacena un valor de tipo flotante de 32 bits con un rango de -3.4028235e+38 a 3.4028235e+38.
double	Almacena un valor de tipo flotante de doble precisión.

Tabla 6: Relación de variables utilizadas en el código de Arduino

10 ANEXO B: ALGORITMOS

10.1 Código FreqCounter

```
#include <FreqCounter.h>

const int PinEN   = 13;      //Pin de entrada al sensor. Valor = 1 para activación.
const int PinOUT  = 12;      //Pin de salida del sensor. Tren de pulsos DCycle=4% variando frecuencia.
int estado_actual = 0;       //Estado inicial del sensor: 0 (Apagado)
int gatetime      = 500;     //Cada cuanto se calcula la frecuencia en ms
int frecuencia;              //Frecuencia de la señal
unsigned long tiempo0 = 0;
unsigned long tiempo;

void setup() {
  Serial.begin(115200);  // Conexión con el puerto serie
  pinMode(PinOUT, INPUT);
  pinMode(PinEN, OUTPUT);
  enable();
}

void loop() {
  FreqCounter::f_comp = 8; // Valor de calibración
  FreqCounter::start(gatetime);
  while (FreqCounter::f_ready == 0)
  frecuencia = FreqCounter::f_freq*1000/gatetime;
  tiempo+= gatetime+tiempo0;
  print_speed();
}

void enable()
{
  digitalWrite(PinEN, LOW);
  delayMicroseconds(5);
  digitalWrite(PinEN, HIGH);
  wait();
}

void wait()                    //Espera a que el sensor devuelva un valor de 1
{
  while (digitalRead(PinOUT) != 1) {
  }
  estado_actual = 1;    //Ahora el sensor se encuentra activado y saca un tren de pulsos
  Serial.println("Movimiento detectado!");
}
```

```
void print_speed()
{
  Serial.print(" ");
  Serial.print(tiempo);
  Serial.print(" ");
  Serial.print(frecuencia);
  Serial.print(" ");
  float ratio = 299792458.0 / (2 * 10525000000); //Razón calculada por c/(2*freq_emitida)
  Serial.print(frecuencia * ratio);
  Serial.print(" ");
  Serial.println(frecuencia * ratio / 3.6);
}
```

10.2 Código FreqCount

```
#include <FreqCount.h>

const int PinEN   = 13;      //Pin de entrada al sensor. Valor = 1 para activación.
const int PinOUT  = 12;      //Pin de salida del sensor. Tren de pulsos DCycle=4% variando frecuencia.
int current_state = 0;       //Estado inicial del sensor: 0 (Apagado)
int gatetime      = 500;     //Intervalo en ms en el que detecta los pulsos. Cada 1 segundo.
int frecuencia    = 0;       //Número de pulsos que se van a contar cada tiempo de gatetime
unsigned int tiempo0 = 0;

void setup() {
  Serial.begin(115200);  // Conexion con el puerto serie a 9600bps
  pinMode(PinOUT, INPUT);
  pinMode(PinEN, OUTPUT);
  enable();
  FreqCount.begin(gatetime);
  }

void loop() {
  if (FreqCount.available()) {
  frecuencia = FreqCount.read();
  print_speed();
  }
}

void enable(){
  digitalWrite(PinEN, LOW);
  delayMicroseconds(5);
  digitalWrite(PinEN, HIGH);
  wait();
}
```

```
void wait()                         //Espera a que el sensor devuelva un valor de 1 para activarlo.
{
  while (digitalRead(PinOUT) != 1){
    }
  current_state = 1;  //Ahora el sensor se encuentra activado y saca un tren de pulsos
  Serial.println("Movimiento detectado!");
}

void print_speed(){
    Serial.print(" ");
  Serial.print(micros()-tiempo0);
  Serial.print(" ");
  Serial.print(frecuencia);
  Serial.print(" ");
  float ratio = 299792458.0 / (2 * 10525000000); //Razón calculada por c/(2*freq_emitida)
  Serial.print(frecuencia * ratio);
  Serial.print(" ");
  Serial.println(frecuencia * ratio / 3.6);
}
```

10.3 Código FreqMeasure

```
#include <FreqMeasure.h>

//Recordar cambiar los pins 12 y 13!
//Desventaja: no es posible leer frecuencia 0. Solución aproximada con millis().

const int PinEN   = 12;      //Pin de entrada al sensor. Valor = 1 para activación.
const int PinOUT  = 13;      //Pin de salida del sensor. Tren de pulsos DCycle=4% variando frecuencia.
int estado_actual = 0;       //Estado inicial del sensor: 0 (Apagado)
double sum        = 0;
int count         = 0;
float frecuencia;
float tiempo;
unsigned long tiempo0     = 0;

void setup() {
  Serial.begin(115200);
  pinMode(PinOUT, INPUT);
  pinMode(PinEN, OUTPUT);
  enable();
  FreqMeasure.begin();
}
```

```
void loop() {
  if (FreqMeasure.available())
  {
    sum = sum + FreqMeasure.read();
    count = count + 1;
    if (count > 5) {
      frecuencia = FreqMeasure.countToFrequency(sum / count);
      sum = 0;
      count = 0;
      print_speed();
      tiempo = millis();
    }
  }
  else {
    if (millis() - tiempo > 1000)  //Permite que si no detecta ninguna frecuencia en un intervalo de tiempo, esta sea 0.
    {
      frecuencia = 0;
      print_speed();
      tiempo = millis();
    }
  }
}

void enable(){
  digitalWrite(PinEN, LOW);
  delayMicroseconds(5);
  digitalWrite(PinEN, HIGH);
  wait();
  tiempo0 = micros();
}

void wait()  {                    //Espera a que el sensor devuelva un valor de 1
  while (digitalRead(PinOUT) != 1)
  {
  }
  estado_actual = 1;    //Ahora el sensor se encuentra activado y saca un tren de pulsos
  //Serial.println("Movimiento detectado!");
}
void print_speed(){
  Serial.print(" ");
  Serial.print(micros()-tiempo0);
   Serial.print(" ");
  Serial.print(frecuencia);
  Serial.print(" ");
  float ratio = 299792458.0 / (2 * 10525000000); //Razón calculada por c/(2*freq_emitida)
  Serial.print(frecuencia * ratio);
  Serial.print(" ");
  Serial.println(frecuencia * ratio / 3.6);
}
```

10.4 Código PulseIn

```
const int PinEN = 13;   //Pin de entrada al sensor. Valor = 1 para activación.
const int PinOUT = 12;  //Pin de salida del sensor. Tren de pulsos DCycle=4% variando frecuencia.
float dur;
float tiempo;
float frecuencia;
int estado_actual = 0;  //Estado inicial del sensor: 0 (Apagado)
unsigned int tiempo0 = 0;

void setup() {
  Serial.begin(115200);   //Conexión con el puerto serie
  pinMode(PinOUT, INPUT);
  pinMode(PinEN, OUTPUT);
  enable();
}

void loop() {
  dur = pulseIn(PinOUT, HIGH, 6000000); //Duracion de la activación de un pulso
  if (dur != 0) {
    tiempo = dur / (0.04 * 1000000);    //D.Cycle=4%
    frecuencia = 1 / tiempo;
    print_speed();
  }
}

void enable()
{
  digitalWrite(PinEN, LOW);
  delayMicroseconds(5);
  digitalWrite(PinEN, HIGH);
  wait();
  tiempo0 = micros();
}
void wait()  //Espera a que el sensor devuelva un valor de 1
{
  while (digitalRead(PinOUT) != 1)
  {
  }
  estado_actual = 1;
  Serial.println("Movimiento detectado!");
}
void print_speed()
{
  Serial.print(" ");
  Serial.print(micros()-tiempo0);
  Serial.print(" ");
  Serial.print(frecuencia);
```

```
Serial.print(" ");
  float ratio = 299792458.0 / (2 * 10525000000); //Razón calculada por c/(2*freq_emitida)
  Serial.print(frecuencia * ratio);
  Serial.print(" ");
  Serial.println(frecuencia * ratio / 3.6);
}
```

10.5 Código interpretación datos y visualización en Matlab

```
close all;
clear all;

% Carga datos
d1 = load('freq_measure_odom.txt'); %Sensor óptico
d2 = load('freq_measure_vel.txt');  %Sensor Doppler

% Pasamos tiempo a segundos
d1(:,3) = d1(:,3)/1000000000;
d1(:,3) = d1(:,3)-d1(1,3);
d2(:,3) = d2(:,3)/1000000000;
d2(:,3) = d2(:,3)-d2(1,3);

% Calculamos el modulo de la velocidad del pose
v1(1) = 0;
for i=2:length(d1(:,1))
    v1(i) = norm(d1(i,5:7)-d1(i-1,5:7))/(d1(i,3)-d1(i-1,3));
end

% Ploteamos velocidades
figure(1);
plot(d1(:,3), v1, 'r', d2(:,3), d2(:,5), 'b');
grid on;
xlabel('Tiempo [s]');
ylabel('Velocidad [m/s]');
legend('Velocidad Cámara','Velocidad Doppler');
```

10.6 Código disminución del número de datos y representación de errores en Matlab

```
n = length(d1(:,3));
m = length(d2(:,3));
aux = zeros(m,1);
sum = 0;
k=1;
a = d1(k,3);
cont=0;

%% Disminución de datos a través de medias
for i=2:m
    b = d2(i,3);
while a<=b
    sum = sum + v1(k);
    k = k+1;
    cont=cont+1;
    a = d1(k,3);
end
aux(i,1) = sum/cont;
cont=0;
sum = 0;
end
```

```
%% Representación de velocidades con menor nº de datos
figure(2);
plot(d2(:,3), aux, 'r-*', d2(:,3), d2(:,5), 'b-o');
grid on;
xlabel('Tiempo [s]');
ylabel('Velocidad [m/s]');
legend('Velocidad Cámara','Velocidad Doppler');


%% Representación de errores
figure(3);
err_abs = abs(aux-d2(:,5));
plot(d2(:,3),err_abs)
xlabel('Tiempo [s]');
ylabel('Error [m/s]');

media_error = mean(err_abs)
desv_error  = std(err_abs)
```

REFERENCIAS

[1] C. R. &. H. Muller, «Low Cost Indoor Positioning System,» de *Ubicomp 2001: Ubiquitous Computing*, Springer Berlin Heidelberg, 2001.

[2] J. Rosen y L. Q. Gothard, de *Encyclopedia of Physical Science*, Infobase Publishing, 2009, p. 155.

[3] C. Herrera Martínez, «Integración de Sensor Doppler en Móvil Romeo 4,» [En línea]. Disponible: http://0-bibing.us.es.fama.us.es/proyectos/use/abreproy/11087/direccion/Memoriapdf%252F.

[4] E. J. Barlow, «Doppler Radar,» *Proceedings of the IEEE.*

[5] Parallax, «DataSheet X-Band Motion Sensor,» [En línea]. Disponible: http://simplytronics.com/products/ST-00018.

[6] «Arduino,» [En línea]. Disponible: https://www.arduino.cc/.

[7] M. Nawrath. [En línea]. Disponible: http://interface.khm.de/index.php/lab/interfaces-advanced/arduino-frequency-counter-library/.

[8] R. &. Paul, «PJRC,» [En línea]. Disponible: https://www.pjrc.com/teensy/td_libs_FreqMeasure.html.

[9] P. &. Robin, «PJRC,» [En línea]. Disponible: https://www.pjrc.com/teensy/td_libs_FreqCount.html.

[10] P. Stoffregen, «Github,» [En línea]. Disponible: https://github.com/PaulStoffregen/FreqMeasureMulti.

[11] W. R. Fried, «Principles and Performance Analysis of Doppler Navigation Systems,» *IEEE Transactions on Aerospace and Electronic Systems.*

[12] U. F. T. F. Cary Spitzer, Digital Avionics Handbook, Third Edition, CRC PRESS, 2014.

[13] R. W. Vopat, «Terrain Bias Compensator for Doppler Navigation Systems». Estados Unidos Patente 5923281, 13 Julio 1999.

[14] «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/.

[15] *Apuntes de la carrera*.

[16] J. Dybedal, «Doppler Radar Speed Measurement Based On A 24 GHz Radar Sensor,» Noruega, 2013.