UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

ESTIMACIÓN DE LA VELOCIDAD Y ACELERACIÓN DE UN SERVOMOTOR A PARTIR DE LAS MEDIDAS DE UN ENCODER INCREMENTAL

TRABAJO FINAL DEL:

Grado en Ingeniería Electrónica Industrial y Automática.

REALIZADO POR:

Alejandro Sebastián Narváez.

TUTORIZADO POR:

Ranko Zotovic Stanisic.

Valencia, septiembre 2019.

Agradecimientos

A mi familia y a Marina por aguantarme cada día.

A mi tutor Ranko por acompañarme en este camino.

Resumen

El uso de *encoders* en el mundo de la electrónica ha sido frecuente y útil. En este trabajo Fin de Grado hemos analizado la capacidad de dicho dispositivo electrónico para llevar a cabo la estimación de la velocidad y la aceleración de un servomotor mediante el análisis de sus mediciones. Se han llevado a cabo distintos tipos de filtrado, desde la estimación diferencial hasta filtro el filtro de *Kalman*, con el fin de estudiar la aplicación óptima. En este TFG utilizamos una plataforma experimental proporcionada por el laboratorio del DISA a la cual se ancla el servomotor y el *encoder* para medir la posición. La comunicación ha sido realizada mediante una tarjeta de adquisición ARM STM32 para la obtención de datos.

Palabras Clave: *Encoder*, servomotor, filtro, *Kalman,* comunicación.

Abstract

The use of encoders in the world of electronics has been frequent and useful. In this end-of-degree Project we have analyzed the capacity of such electronic device in order to obtain the estimation of the velocity and acceleration of a servomotor. A few types of filters have been used, such as the differential method, or the use of a Kalman filter, with the objective of obtaining the best results. In this work we have used an experimental platform, provided by the DISA laboratory in the UPV, to which the servomotor and the encoder are attached. The communication was made using an ARM STM32 acquisition card and a PC.

Palabras Clave: encoder, servomotor, filter, *Kalman,* communication.

Lista de figuras

Lista de tablas

Índice

1. Introducción.

En este trabajo Fin de Grado nos centramos en la obtención de datos a partir de un encoder situado en un servomotor. Este servomotor irá colocado en una plataforma móvil con un rango de 10 cm entre tope y tope. A lo largo de la experiencia hemos probado distintos métodos para la estimación de la aceleración de dicho servomotor. El objetivo es la estimación de la aceleración de nuestro sistema para en un futuro aplicarlo a robótica móvil.

Hemos movido la plataforma con la mano para llevar a cabo la experiencia, y, aunque en principio pretendíamos utilizar un acelerómetro para comprobar nuestros resultados, debido a la calidad del acelerómetro no se obtuvieron resultados fiables, pero si lo suficientemente orientativos. No obstante, toda la explicación y planteamiento para la adquisición de los datos del acelerómetro se ha mantenido en el caso de que se quiera profundizar o gastar más en este aspecto. Los datos para estimar la aceleración han sido procesados a través de funciones de Matlab.

La adquisición de datos y comunicación han sido llevadas a cabo mediante el uso de una tarjeta STM32 con un microprocesador ARM y *Teraterm* para poder almacenar los datos que se obtenían a través de las interrupciones programadas. A lo largo de este TFG explicaremos el procedimiento para llevar a cabo todo este proyecto

2. Estado del arte

Ser capaces de medir la aceleración y velocidad de los robots que comparten el mundo con nosotros se ha vuelto una tarea de más y más importancia. Hoy día se desarrollan más que nunca robots que entran en contacto físico con los humanos y el conocimiento de la dinámica de los autómatas nos permite un uso más preciso y seguro en estas interacciones robo-humanas.

La estimación de la aceleración de los motores a partir del uso de *encoders* ha sido estudiada previamente, no obstante, estamos ante un caso particular con una plataforma específica y unos métodos de muestreo y estimación distintos.

La primera solución que se nos pasa por la cabeza es utilizar un acelerómetro para medir la aceleración del motor. El problema que existe con el uso de dicho dispositivo es que la

medición obtenida es la suma de la aceleración, gravedad, fuerza centrífuga y la aceleración de Coriolis.; por lo que no es sencillo aislar la aceleración del resto.

Otra solución, y la adoptada en este caso, es el uso de los sensores de posición integrados en los robots, estos sensores suelen ser *encoders*. La estimación de la aceleración a partir de la posición es una conclusión lógica evidente, pero el *encoder* tiene un problema, el error de cuantificación digital. Para obtener la aceleración se debería llevar a cabo la doble derivada de la posición, y teniendo en cuenta el error de cuantificación nos podría llevar a errores demasiado elevados.

El error de cuantificación es el problema más difícil de solventar, varios métodos son válidos para reducir el error, el uso de un filtro paso bajo es común y útil en frecuencias bajas, pero en una experiencia con frecuencias altas podría llevarnos a un error mayor aún.

Otra manera de solucionar el problema es cambiar el método de muestreo a utilizar. El método más común que se aplica es el muestreo constante, estableciendo un periodo que proporciona medidas en intervalos constantes. Una alternativa sería muestrear cada vez que se produce un pulso del *encoder*. Hoy día se trabaja también con una nueva técnica que combina ambos métodos de muestreo, obteniendo la aceleración estimada mediante la extrapolación de diferentes medidas.

La estimación de la aceleración a través del acelerómetro queda parcialmente descartada, ya que con un equipo de alta calidad se podría llevar a cabo una medición más o menos exacta, pero en el fondo solo serviría como método comparativo y en ese caso ya se dispone de acelerómetros de baja gama.

La posibilidad de estimar la aceleración a partir del *encoder* de los robots ha sido estudiada a lo largo de la última década. Se han propuesto gran cantidad de métodos, y los autores diferencian entre filtros predictivos y observadores de estado. Los primeros han probado ser resolutivos, solo pueden aplicarse cuando la curva de aceleración puede aproximarse con un polinomio de bajo grado y por lo tanto son útiles en nuestro caso. En el caso de los observadores de estado no se produce esta limitación. El método más utilizado suele ser el filtro de *Kalman*.

De este modo, aplicaremos conocimientos empleados por investigadores, pero trabajaremos con un método de muestreo distinto, el que combina el muestreo por pulsos del *encoder* y un muestreo constante.

3. Análisis técnico.

Una vez introducido el trabajo y situado el marco tecnológico en el que nos encontramos, vamos a profundizar en los objetivos y las teóricas que hemos utilizado a lo largo de este trabajo.

3.1. Objetivo general.

El objetivo es establecer una estimación de la aceleración y la velocidad del motor que hemos anclado a la plataforma. En el trabajo hemos incluido un acelerómetro de gama baja cuyo propósito es obtener una aceleración aproximada que nos permita verificar el correcto funcionamiento (o ausencia de este), de la estimación realizada.

La plataforma se mueve bajo acción humana, es decir, nosotros marcaremos el ritmo que nos parezca apropiado, llevándola hasta los topes y variando las velocidades para obtener ritmos distintos y así tener variedad en los datos recogidos.

El microprocesador ARM de la tarjeta de adquisición se configura de tal manera que los datos de posición del *encoder* se reciben y se guardan en el PC mediante el uso de *Teraterm* y *Keil uVision*. Estos datos los procesaremos en Matlab, hemos usado métodos de derivadas, filtros e interpolación para obtener el método óptimo de estimación a usar con el método híbrido de muestreo. En el siguiente apartado hemos ahondado en los requerimientos teóricos necesarios para la puesta en marcha de la experiencia.

3.2. Conocimientos teóricos.

El típico bucle de control trabaja con un periodo de muestreo constante, por lo que resultaría lógico leer los pulsos del *encoder* en intervalos constantes de tiempo.

Asumiendo que la resolución del *encoder* es *r* (pulsos por radianes) y la posición medida es *p*, el valor real de la posición se encontrará entre *p* y *p+r*. Se podría encontrar en cualquier lugar dentro de este intervalo y no existe manera de determinar la posición con mayor precisión. Asumiendo que el valor medio del error es *r/2,* la varianza de la posición será dada por la siguiente expresión:

$$\sigma_p = \frac{1}{r}\int_0^r (\rho - \frac{r}{2})^2 d\rho = \frac{1}{3}r^2$$

Si la velocidad y la aceleración son estimadas siguiendo un sencillo sistema de derivación con tiempo de muestreo T:

$$\hat{v} = \frac{p - pz^{-1}}{T}$$

$$\hat{a} = \frac{v - vz^{-1}}{T}$$

Las varianzas de la velocidad y la aceleración son, respectivamente:

$$\sigma_v = \frac{2\sigma_p}{T^2} = \frac{2r}{3T^2}$$

$$\sigma_a = \frac{2\sigma_v}{T^2} = \frac{4r}{3T^4}$$

Por lo que un menor período de muestre produce un error de varianza que se incrementa conforme aumenta la frecuencia de muestreo. Por otra parte, intervalos de muestreo más altos tienden a ser inadecuados para controlar movimientos rápidos. La varianza dada por las expresiones podría mejorarse mediante el uso de un *encoder* con resolución más elevada, no obstante, esto encarecería nuestro proyecto.

Con el fin de mejorar el proceso se puede calcular la aceleración a cada pulso del *encoder*. De este modo, la medida se toma en cada momento que sucede un cambio de posición del motor. Las medidas se toman de una en una, por lo que el error de cuantificación digital desaparece y con ello la necesidad de utilizar un encoder de alta resolución.

En el momento en que se genera un pulso del *encoder*, se produce una interrupción en nuestra CPU y se toman las medidas. De este modo, la fórmula de la varianza no es válida. En un encoder ideal la varianza es cero, ya que el muestreo sucede en el instante exacto que el encoder produce un pulso. No obstante, en la aplicación real, las hendiduras del sensor no son equidistantes debido a interferencias externas como la luz del LED, desalineamiento…

Podemos estimar la varianza de los *encoders* mediante una distribución normal, esta varianza debería ser considerablemente más pequeña que para el período de muestreo.

El sistema de muestreo decidido es el variable, tomamos medidas cada vez que hay un cambio de posición en el *encoder,* pero también se dispone de una interrupción constante que nos permite adquirir una mayor precisión a la hora de medir y no perder pulsos de *encoder* a la hora de llevar a cabo las mediciones [1] .

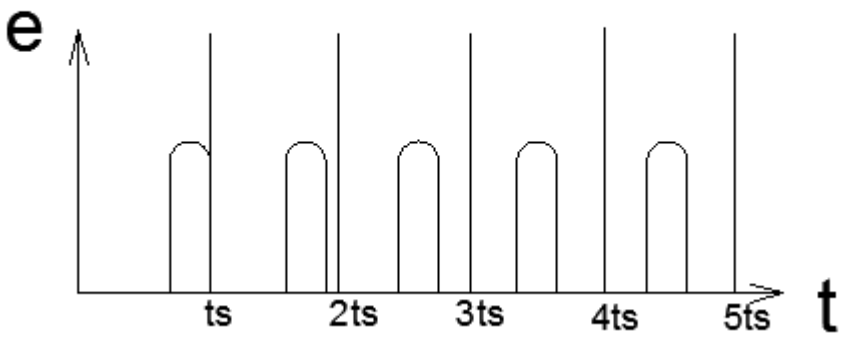

Figura 1: Representación gráfica del muestreo del encoder

Teniendo en cuenta que el muestreo puede coincidir directamente sobre los pulsos del *encoder* en movimiento, es preciso realizar un doble muestreo con el fin de tener un proceso sin pérdida de datos, cosa que influiría negativamente en la estimación de las medidas a realizar.

El método más directo para estimar la aceleración es el método diferencial puro, asumir que un intervalo, la distancia recorrida por el motor se corresponde con un pulso del encoder, y entonces estimaríamos la velocidad de la siguiente manera:

$$\hat{v}_i = \frac{r}{T_i}$$

Y la aceleración:

$$\hat{a}_i = \frac{\hat{v}_i - \hat{v}_{i-1}}{T_i} = \frac{\frac{r}{T_i} - \frac{r}{T_{i-1}}}{T_i} = r\frac{T_{i-1} - T_i}{T_i^2 T_{i-1}}$$

Este método, como hemos mencionado, es el más simple para la obtención de la aceleración, tiene un bajo coste de cálculo y no requiere mucho tiempo, lo cual permite mayores velocidades y resolución de los *encoders*. No obstante, el error se acumula en exceso y a velocidades bajas es poco fiable. Ya que en nuestro caso hemos movido la plataforma utilizando la mano, no se alcanzan velocidades elevadas y este método no es el óptimo para la aplicación deseada [2].

Otra manera de estimar es mediante la derivación a través de un filtro paso bajo. La aceleración estimada pasa a través de un filtro de la forma:

$$Output = (1 - alpha) * input + alpha * output * z\text{^} - 1$$

Este filtro permite suavizar los picos que se generan debido a la doble derivación, mediante este método se puede realizar una estimación más educada de cómo funciona el sistema. No obstante, conforme se aumenta la frecuencia de corte aparece un desfase. A frecuencias muy altas obtenemos un suavizado que puede acercarse al objetivo deseado pero el desfase es demasiado elevado como para tomarlo como un resultado aceptable.

El filtro de *Kalman* es un método muy conocido para la estimación de estados en teoría de control. Éste filtro se compone de dos fases, la fase de predicción y la fase de corrección. En este proyecto hemos asumido que la dinámica del sistema se describe conforme a la siguiente ecuación:

$$\dot{X} = AX + W$$

Dónde X' representa el vector de variables de estado del sistema, A es la matriz de estado y W es el ruido del proceso. Este ruido se obtiene de una distribución normal de mediana cero, con covarianza Q.

En este trabajo el vector de estados del sistema contiene la posición, la velocidad y la aceleración.

$$X = \begin{bmatrix} x \\ \dot{x} \\ \dot{x} \end{bmatrix}$$

La matriz de estado A tiene la siguiente forma:

$$e^{A\Delta t} = \begin{pmatrix} 1 & \Delta t & \Delta t^2 \\ 0 & 1 & \Delta t \\ 0 & 0 & 1 \end{pmatrix}$$

Ésta matriz representa un sistema de aceleración constante. Dado que el par del motor es constante en un periodo de muestreo, esta suposición es posible.

En un sistema convencional con muestreo constante, el filtro de Kalman se modela a partir de la siguiente serie de ecuaciones.

El predictor:

$$\overline{X}_k^- = A_d \overline{X}_{k-1}$$

$$P_k^- = A_d P_{k-1} A_d^T + W_d$$

$$K = P_k^- C^T (C P_k^- C^T + R)^{-1}$$

El corrector:

$$\overline{X}_k = \overline{X}_k^- + K(z_k - C\overline{X}_k^-)$$
$$P_k = (I - KC)P_k^-$$

Como en nuestra aplicación hemos utilizado un sistema de muestreo variable, no podemos emplear el filtro estacionario, esto se ve reflejado en un cambio de la matriz A, en el que elaboramos más adelante.

La matriz W representa la varianza del predictor. Es decir, representa la precisión del modelo y la probabilidad de interferencias, esta matriz es complicada de ajustar ya que puede variar

dependiendo de las condiciones de trabajo del modelo. Hemos utilizado la siguiente matriz W para el proceso.

$$W = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & w \end{bmatrix}$$

La explicación es que la aceleración es el único estado que pueda tener ruido blanco independiente. El ruido del resto de estados depende del ruido de la aceleración. Esta relación se procesa en el filtro de *Kalman* a través de la matriz de estados A [3].

La matriz R representa la varianza de las mediciones, o lo que es lo mismo, la precisión de los sensores a la hora de medir la posición. Ya que los datos se miden en cada pulso del *encoder*, el error de cuantificación digital, y la varianza R, deberían ser cero en el caso ideal. Pero como en el mundo real no existen los casos ideales y la configuración e instalación del *encoder* pueden dar lugar a pequeños errores debido a la manipulación humana. Estos errores se considerarán de por lo menos un orden de magnitud por debajo de la resolución del *encoder*.

El filtro de *Kalman* no estacionario se aplica cuando las medidas se toman en períodos de tiempo irregulares, como es nuestro caso, la matriz Ad se computa a partir de la matriz A y el diferencial de tiempo.

La matriz A para estimar la aceleración en nuestro caso es la siguiente, nos basamos en distintos modelos y en el ámbito práctico los mejores resultados se han obtenido con esta forma.

$$A = \begin{bmatrix} 1 & 0.5 & 0.5\text{^}3 \\ 0 & 1 & 0.5 \\ 0 & 0 & 1 \end{bmatrix}$$

En el apartado resultados de la experiencia se profundiza más en la programación del filtro de Kalman y la aplicación particular a la que nos hemos dedicado. Para la experiencia también se requieren conocimientos de programación y adquisición de datos para la parte de comunicación y procesado de datos.

4. Solución técnica.

La solución al problema planteado en este TFG tiene una amplia variedad de componentes para formar el sistema que nos permitirá realizar las medidas de una manera satisfactoria. Dividimos la solución técnica en varias partes. La definición y diseño de las herramientas tecnológicas a utilizar, el montaje de estas, y su funcionamiento.

La solución técnica viene dada, en mayor parte, por los elementos que hay disponibles en el laboratorio. Hemos decidido el diseño de la solución sobre todo en el ámbito de la programación, ya que es la parte con menores limitaciones.

4.1. Herramientas tecnológicas utilizadas.

Los elementos más importantes de esta solución son, como ya hemos mencionado antes, el encoder y la comunicación entre el servomotor y la tarjeta. Hemos realizado un análisis de cada uno de los componentes que hemos utilizado en la experiencia, como se unen y cómo se diseñan.

4.1.1. Plataforma

La plataforma sobre la cual hemos realizado la experiencia es una plataforma de movimiento lineal rígido de la marca *Quanser*. La plataforma tiene unos diez centímetros de maniobra entre sus dos extremos, espacio en el cual realizaremos nuestra experiencia. La plataforma es rígida y resistente a los golpes, cosa que nos ha permitido experimentar con controladores propios para la estimación de la aceleración.

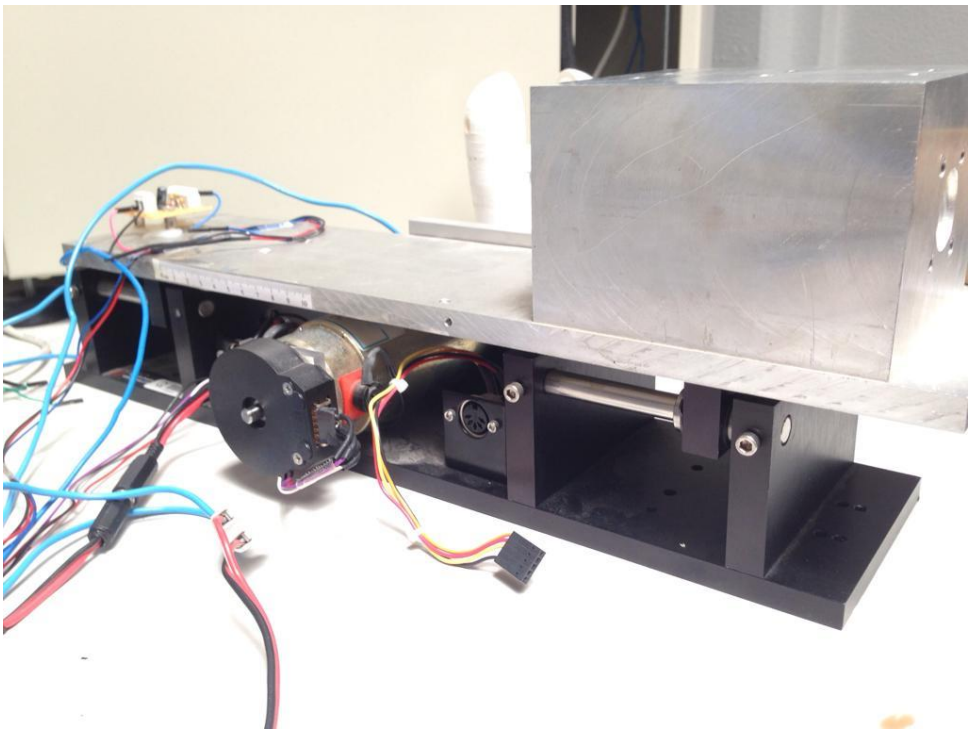

Figura 2: Plataforma Lineal Quanser

La plataforma tiene un hueco en el cual situaremos el motor, la linealidad de la plataforma permite que no se produzca una pérdida de datos por inclinaciones o desviaciones en el funcionamiento esta.

En la parte superior de la plataforma hemos situado el acelerómetro, que ha sido comprobado previamente para que la aceleración se calcule en el eje de coordenadas correcto, en nuestro caso el acelerómetro no proporcionó los resultados deseados, pero el montaje es el mismo en el caso del uso de un acelerómetro con más precisión.

Cabe indicar la plataforma alberga la mayoría de los componentes y los cables se mueven a lo largo de esta, por lo que hay que tener especial cuidado a la hora de manipular la plataforma o los cables para evitar accidentes, ya sea para evitar cortocircuitos o malos funcionamientos de componentes o para evitar daños humanos.

4.1.2. El encoder incremental.

La parte principal de este TFG se centra en el estudio de los sensores de posición del motor, o lo que es decir el *encoder*. Son conocidos como codificadores o descodificadores, cuyo objetivo es transformar unos datos o información de un tipo de formato o lenguaje a otro. Esto permite

transmitir información cifrada, conseguir una mayor velocidad de transmisión o simplemente permitirnos trabajar de una manera sencilla con el mismo set de datos.

En este caso hemos decidido trabajar con *encoders* para el control de motores, que es lo prevalente en nuestra aplicación. La función de estos *encoders* es convertir el movimiento mecánico del giro de su eje en pulsos digitales. Estos pulsos digitales pueden ser procesados por una tarjeta de adquisición o por un controlador.

Figura 3: Servomotor con encoder incorporado

El *encoder* de motor se compone de un disco conectado a un eje giratorio. Este disco está hecho de un material como plástico fino o vidrio y tiene pequeñas hendiduras a lo largo del borde exterior del círculo. Estas hendiduras son opacas o transparentes con el fin de bloquear o permitir el paso de la luz emitida por una fuente situada a uno de los lados del disco.

Conforme el eje gira, el emisor de luz proyecta el haz que recibe el sensor óptico (dependiendo de si la hendidura bloquea o permite el paso) generando así los pulsos digitales que indican el movimiento en un sentido u otro del motor. Esto produce una secuencia que proporciona los datos necesarios para el control o procesado de datos del servomotor que estamos utilizando en la aplicación [4].

Existen dos tipos de encoder según su funcionamiento, el incremental y el absoluto. Existen otros tipos de *encoders* para otro tipo de aplicaciones, pero en nuestro caso utilizaremos el *encoder* para servomotor incremental.

Figura 4: Encoder usado en el experimento

El encoder incremental determina el ángulo de posición mediante el cálculo de cuentas incrementales. Siempre tiene una posición desde la que comenzará la cuenta. La posición es incremental cuando se compara con la última posición registrada por los sensores. Estos *encoders* incrementales son del tipo óptico y cada posición es inédita.

El *encoder* que hemos utilizado en nuestro proyecto es un *encoder* HEDS-9000 de Avago Technologies. Este componente dispone de un coste bajo para su alto rendimiento mediante un sistema óptico. Los módulos del HEDS-9000 consisten en un emisor LED, un detector, y la rueda de codificación. Es importante montar estos módulos de manera correcta ya que son muy sensibles a errores de desplazamiento.

En el HEDS-9000, la rueda codificadora gira entre el emisor y el detector, causando así que el haz de luz se interrumpa por las hendiduras marcadas en la rueda. Los detectores están situados de tal manera que coincida con el ancho y radio de la rueda. Estos detectores también están espaciados de tal manera que un período de luz en uno de los detectores corresponde a un período de oscuridad en el par de detectores adyacentes. La salida de los fotodiodos pasa a través de la circuitería de procesado para dar lugar a las señales A y B y sus correspondientes señales invertidas. La salida digital del canal A está en cuadratura con el canal B (90 grados de desfase). Las condiciones máximas de funcionamiento se detallan a continuación.

Condiciones máximas HEDS-9000	
Temperatura de alamacenaje	De -40ºC a 100ºC
Temperatura de operación	De -40ºC a 100ºC
Voltaje de entrada	-0,5 V a 7 V
Voltaje de salida	-0,5 V a V de entrada
Intensidad de salida por Canal	-1 mA a 5 mA

Tabla 1: Especificaciones de trabajo del encoder

Del mismo modo, cabe indicar la disposición de los distintos pines del *encoder* para asegurarnos de su correcto funcionamiento para la transmisión de datos. También evitaremos problemas de sobretensión.

Conexiones HEDS-9000	
PIN 1	GND
PIN 2	N.C.
PIN 3	Canal A
PIN 4	Alimentación
PIN 5	Canal B

Tabla 2: Conexión de los pines del encoder

Es importante observar el *encoder* desde la posición correcta para evitar errores de conexión, se detalla en la imagen siguiente.

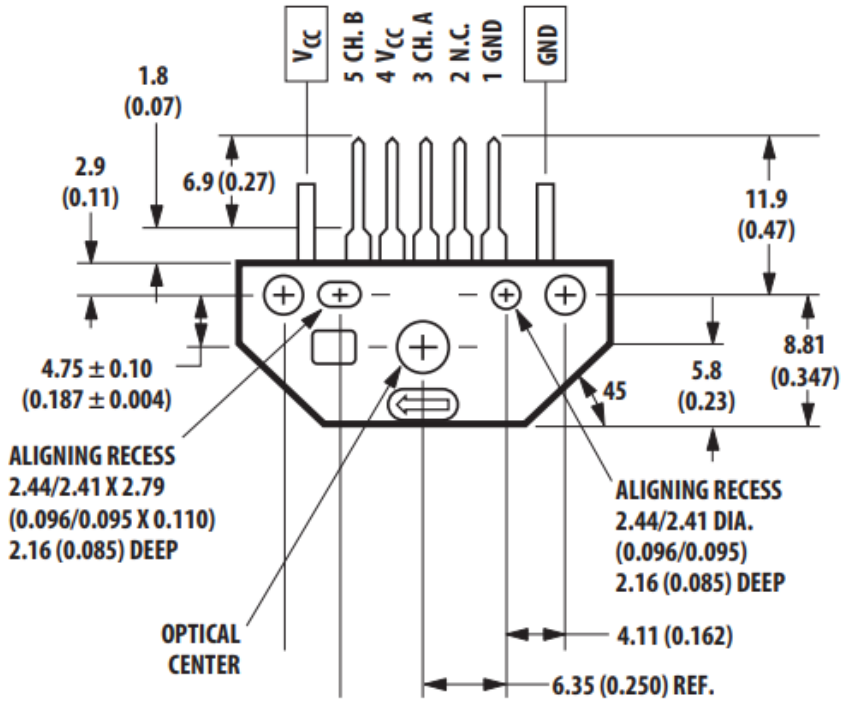

Figura 5: Visualización correcta para identificar los pines del encoder

4.1.3. Servomotor.

Un servomotor es un actuador lineal que permite un control preciso de una posición angular o lineal. Consiste en un motor conectado a un sensor con capacidad de medir la posición. Requieren normalmente de un sofisticado controlador, en muchas ocasiones un módulo de uso específico para esos servomotores.

Los servomotores más simples utilizan únicamente la medición de la posición mediante el uso de un potenciómetro y un control todo o nada del motor (funciona a toda potencia o completamente parado). Este tipo de servomotores no se usa en la industria de control, pero forma la base para los servomotores utilizados en aplicaciones como juguetes de radio control.

Los servomotores más sofisticados utilizan *encoders* rotatorios para medir la velocidad del motor y un controlador para modificar la velocidad del motor. Estas dos características, junto al uso de un algoritmo PID permiten al servomotor llegar a su posición deseada con más precisión y velocidad, pero disminuyendo el *overshoot*.

Los servomotores se utilizan como una alternativa de más precisión a los motores paso a paso, los cuales tienen más habilidad para controlar la posición debido a los pasos marcados en la

salida. Los hemos considerado como una alternativa a plantearse, no obstante, el hecho de que el motor paso a paso no tenga retroalimentación limita su rendimiento. Los pasos que se pierden a lo largo de la adquisición de datos pueden llevar a errores de posición

El servomotor elegido para llevar a cabo la experiencia es un motor de corriente continua cepillado Pittman 14205, en la versión de 24 voltios. El tamaño del motor permite un fácil acople a la plataforma móvil que se usará en la experiencia. Las especificaciones del motor se encuentran a continuación.

Especificaciones Motor Pittman 14205	
Voltaje de entrada	24 V
Constante eléctrica	7,76 V/krpm
Constante del motor	10 oz-in/sqrt W
Constante del par	10,5 oz-in/A
Intensidad máxima	24,6 A
Inercia del rotor	0,0044 oz-in-sec2
Constante Tiempo/Eléctrica	1,6 ms
Constante Tiempo/Mecánica	6,3 ms
Masa	1,119 Kg
Temperatura máxima	155 Celsius
Par máximo	1,5814 Nm

Tabla 3: Especificaciones del motor

La alimentación se lleva a cabo mediante un controlador Escon 50-5. La configuración del módulo controlador ha de realizarse a través de un ordenador, en el cual se introducen las especificaciones del motor cepillado.

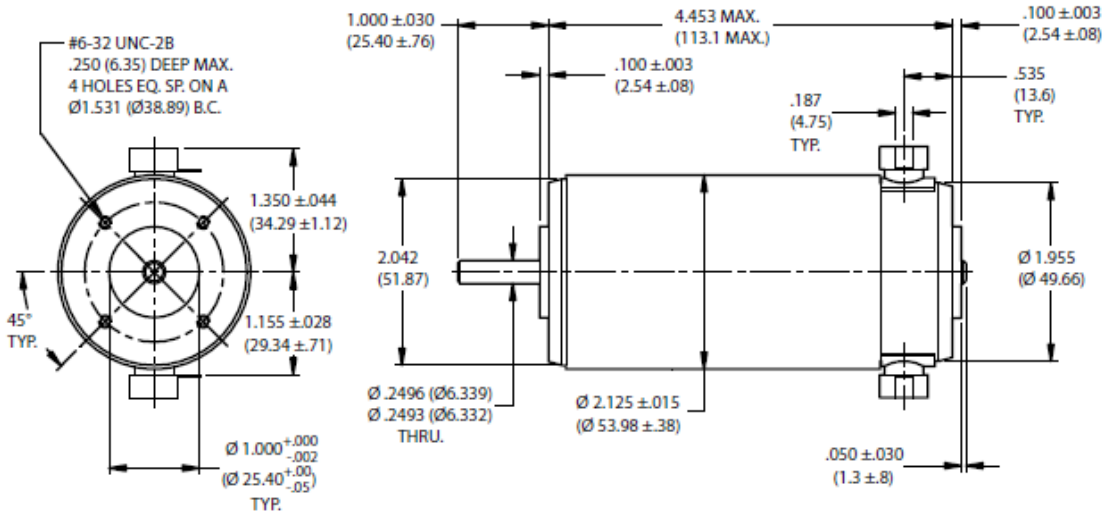

Figura 6: Dimensiones del motor

Los datos que han de introducirse se representan a continuación, esta configuración es únicamente necesaria para la implementación del acelerómetro, ya que en el caso contrario simplemente moveremos la plataforma con la mano.

Primero seleccionamos el tipo de motor que vamos a utilizar, en nuestro caso uno de CC, a ser hemos de utilizar un motor Maxon, pero no es el caso.

Figura 7: Selección de servomotor

De la hoja de especificaciones obtenemos los valores a introducir. La constante de velocidad viene dada en V/Krpm, por lo que sí:

$$\frac{V}{Krpm} = 7.76$$

Entonces:

$$\frac{rpm}{V} = 128.38$$

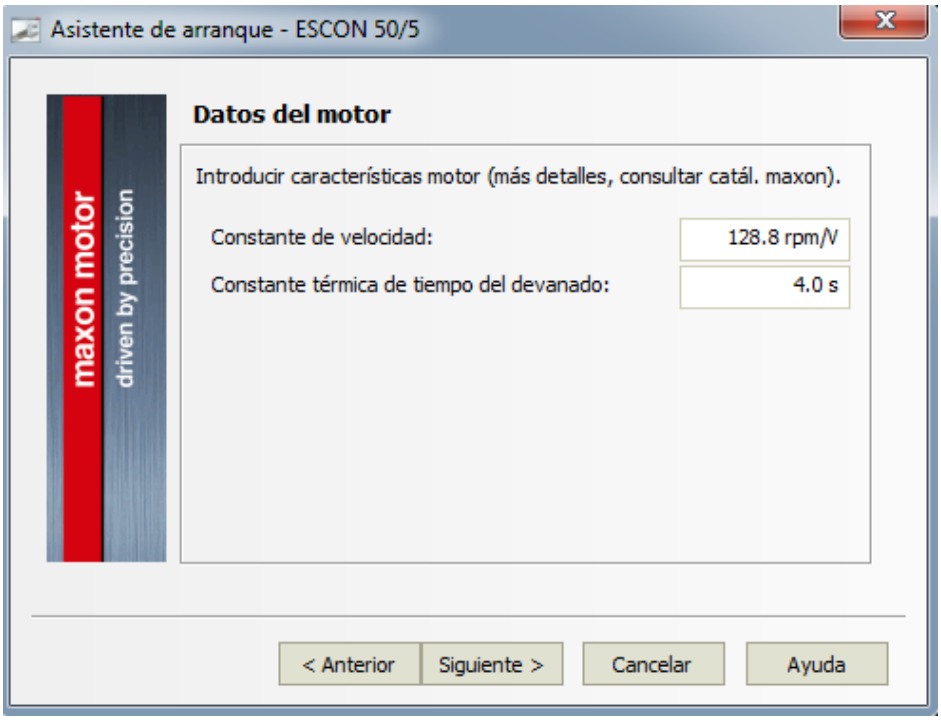

Figura 8: Selección de las constantes del motor

En el caso de la velocidad límite hemos dejado la configuración tal y como se encuentra de serie, ya que nuestro motor es un poco distinto. Hemos cambiado solo la configuración en la intensidad nominal, para la cual hemos introducido el valor dado por nuestra hoja de datos.

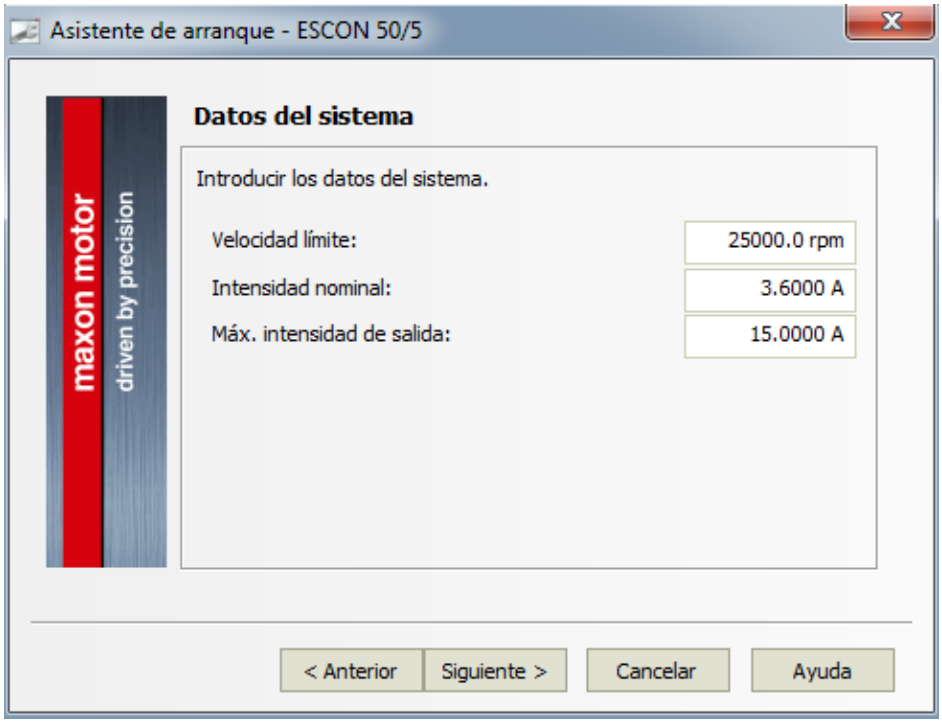

Figura 9: Selección de límites de corriente

Para el modo de operación hemos seleccionado el modo variador de velocidad, ya que el modo de control de corriente provoca demasiada inestabilidad en el motor que hemos utilizado, cosa que se atribuye al hecho de que las marcas no son las mismas y puede haber incompatibilidades a causa de ello.

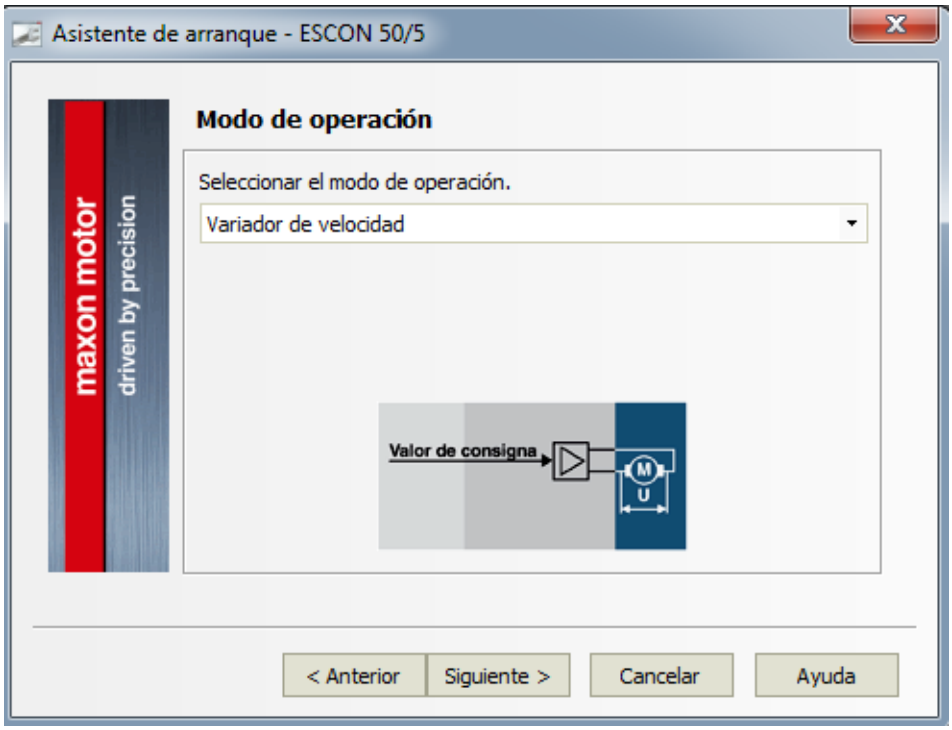

Figura 10: Selección de modo de operación

Hay que designar un canal que permita la operación del dispositivo, este canal será la entrada digital 2. Para que el módulo controlador esté funcionando habrá que cortocircuitar esta entrada con 5 voltios para tener en nivel lógico alto.

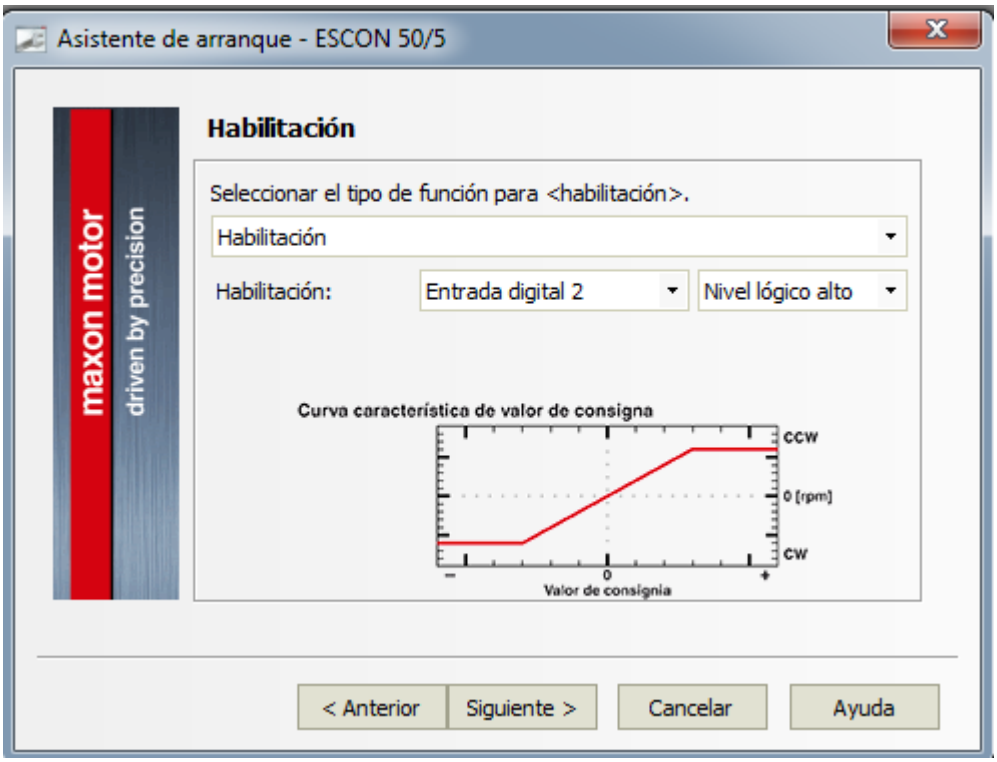

Figura 11: Selección de modo de habilitación

Hemos establecido los límites de velocidad en el caso de que el voltaje supere los valores establecidos, para evitar que el motor se sobrecargue.

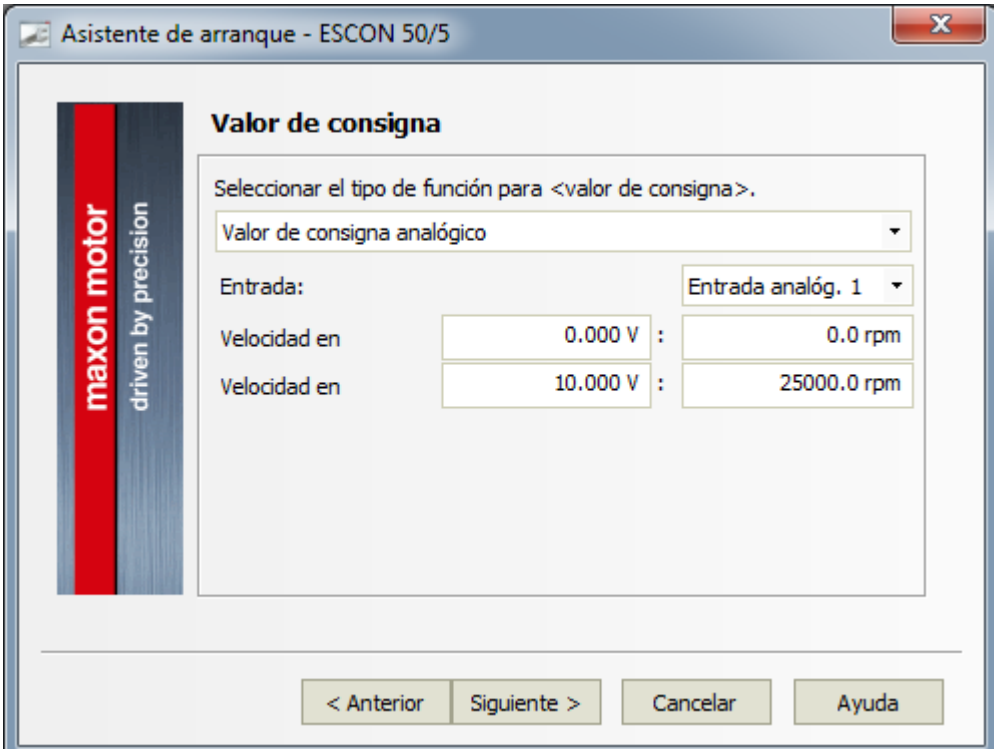

Figura 12: Selección del valor de consigna analógico

La rampa analógica se establece entre 0 y 5 voltios que es el rango de funcionamiento del motor dependiendo de la entrada analógica.

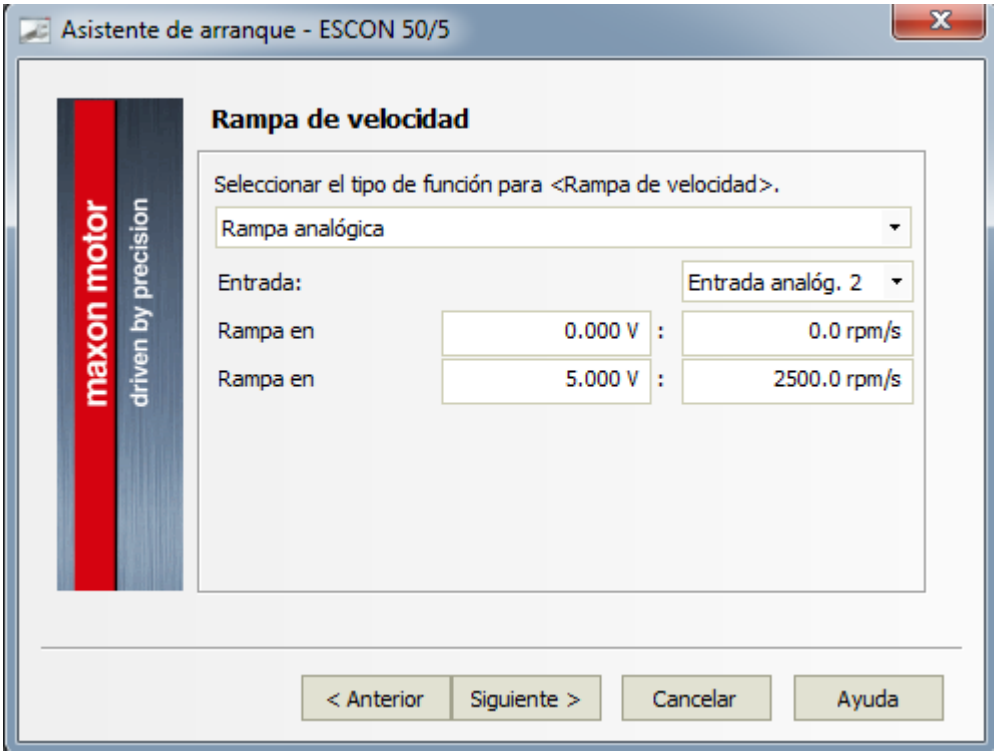

Figura 13: Selección de la rampa de velocidad

No hemos situado ningún nivel de *offset* ya que se lo hemos dado a la señal en el apartado de comunicación. La señal que le llega al controlador ya estará situada entre 0 y 5 por lo que no hay necesidad de añadir ningún *offset*.

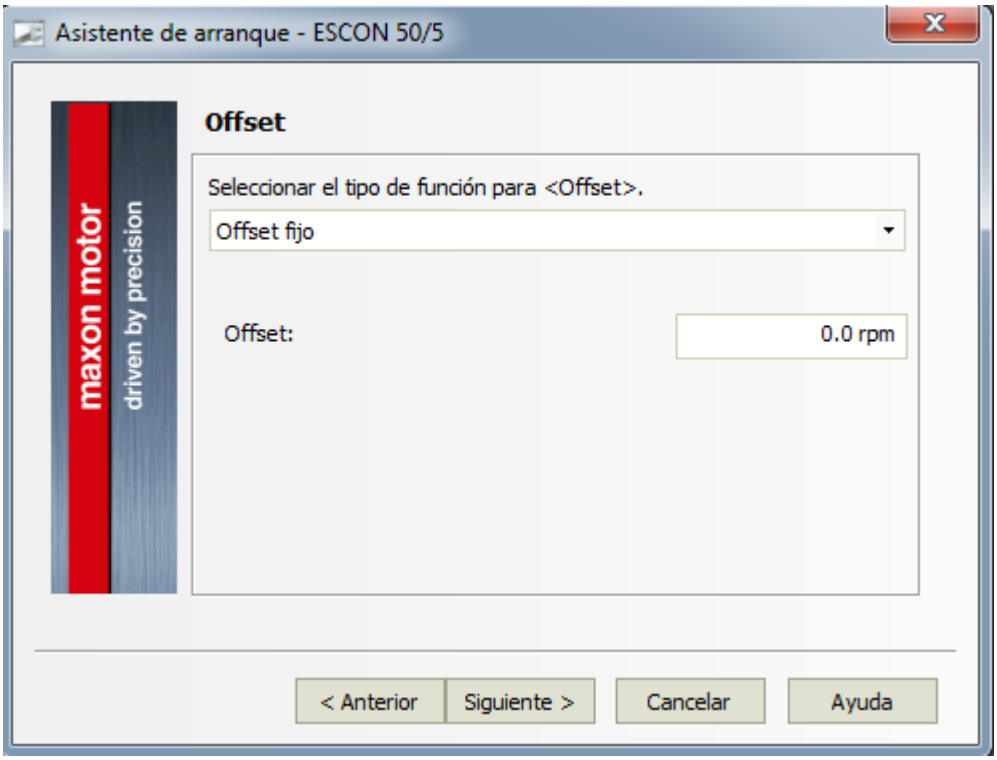

Figura 14: Selección del offset

Hemos definido la salida analógica 1 como una salida de corriente real para poder trasladar el valor calculado para la acción de control directamente al motor. Este paso permite indicar al motor una trayectoria como por ejemplo senoidal.

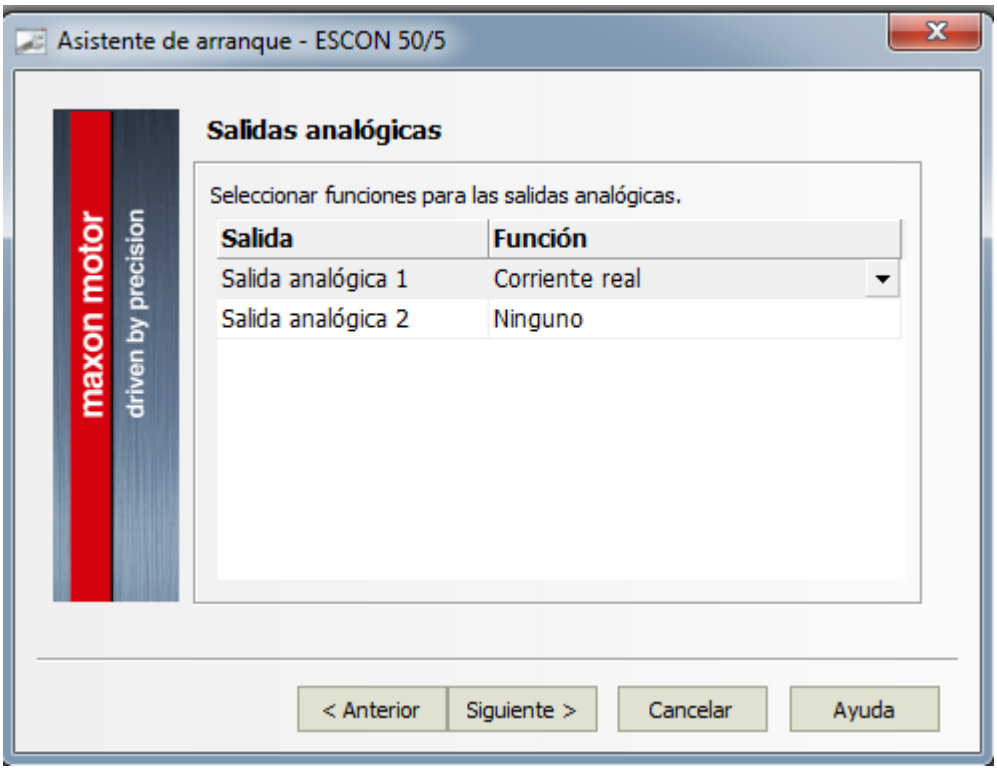

Figura 15: Selección de la salida analógica

Con esta configuración el elemento controlador ya está preparado para funcionar. A lo largo de los siguientes puntos hemos indicado las conexiones entre los distintos dispositivos y la programación que permite que esta configuración funcione correctamente.

4.1.4. Comunicación.

Para la comunicación entre el motor/*encoder* y un PC para almacenar los datos, hemos utilizado un microcontrolador del tipo STM32F4, del fabricante ST Microelectronics. Utiliza un procesador del tipo ARM Cortex.

A lo largo del grado, hemos utilizado tarjetas de este tipo para desarrollar diversas aplicaciones sencillas. En este microcontrolador es necesario conocer su arquitectura para encajar bien la aplicación que hemos desarrollado con los componentes de los que hemos dispuesto.

Para establecer la transmisión de los datos calculados en el microcontrolador, hemos utilizado el programa *TeraTerm* que dispone de una configuración serial muy sencilla. Si hemos configurado bien nuestra tarjeta STM, aparecerá directamente en la pantalla de conexiones en serie. Con el código que hemos desarrollado, es la única configuración que hay que hacer en *TeraTerm*, aparte de descargar los datos una vez se hayan completado.

Para la obtención de los valores de la aceleración, hemos configurado el microcontrolador para que el pin C1 funcione como la entrada de un convertidor Analógico a Digital. De esta manera podemos incluir un acelerómetro en nuestra aplicación para comprobar el correcto funcionamiento del filtro.

```
void acceleration_Init(void) {

  GPIO_InitTypeDef  GPIO_InitStructure;

  ADC_InitTypeDef       ADC_InitStructure;
  ADC_CommonInitTypeDef ADC_CommonInitStructure;

  /* PORT C ***************************************/
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);

   /* PC1 as analog input for ADC3 */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;
  GPIO_Init(GPIOC, &GPIO_InitStructure);

  /* Enable ADCx ***************************************/
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

  /* ADC Common Init ***********************************************************/
  ADC_CommonStructInit(&ADC_CommonInitStructure);
  ADC_CommonInitStructure.ADC_Mode             = ADC_Mode_Independent;
  ADC_CommonInitStructure.ADC_Prescaler        = ADC_Prescaler_Div4; // max 30 MHz according datasheet
  ADC_CommonInitStructure.ADC_DMAAccessMode    = ADC_DMAAccessMode_Disabled;
  ADC_CommonInitStructure.ADC_TwoSamplingDelay  = ADC_TwoSamplingDelay_5Cycles;
  ADC_CommonInit(&ADC_CommonInitStructure);


  /* ADC Init ******************************************************************/
  ADC_StructInit(&ADC_InitStructure);
  ADC_InitStructure.ADC_Resolution               = ADC_Resolution_12b;
  ADC_InitStructure.ADC_ScanConvMode             = DISABLE;
  ADC_InitStructure.ADC_ContinuousConvMode       = DISABLE;
  ADC_InitStructure.ADC_ExternalTrigConvEdge     = ADC_ExternalTrigConvEdge_None;
  ADC_InitStructure.ADC_DataAlign                = ADC_DataAlign_Right;
  ADC_InitStructure.ADC_NbrOfConversion          = 1;
  ADC_Init(ADC_CHOICE, &ADC_InitStructure);

  /* Set injected sequencer length */
  ADC_InjectedSequencerLengthConfig(ADC_CHOICE, 1);
  ADC_SetInjectedOffset(ADC1, ADC_InjectedChannel_1, 0);

  /* Enable ADC */
  ADC_Cmd(ADC_CHOICE, ENABLE);

  /* ADCx injected channel Configuration */
  ADC_InjectedChannelConfig(ADC_CHOICE, ACCEL_CHANNEL, 1, ADC_SampleTime_15Cycles);
}
```

Figura 16: Inicialización del ADC

Después de hacer la configuración del convertidor hay que definir la función que permite leer los datos obtenidos del acelerómetro. Es importante tener en cuenta que hay que realizar una conversión en los datos para que los mismos tengan sentido.

```
float acceleration_Read(void) {

   uint32_t value;
   float volts, accel;

   ADC_ClearFlag(ADC_CHOICE,ADC_FLAG_JEOC);
   ADC_SoftwareStartInjectedConv(ADC_CHOICE);

   while (ADC_GetFlagStatus(ADC_CHOICE,ADC_FLAG_JEOC) == RESET);

   value = ADC_GetInjectedConversionValue(ADC_CHOICE, ADC_InjectedChannel_1);

   volts = value*(VOLT_REF/4095.0F);
   accel = volts*(G4/VOLT_REF)-G2;

   //*value = (v * VOLT_REF)/0xFFF; // 12 bits ADC
   //*milig = (((int32_t)accel_value - 1468)*1000)/295;

   return accel;
}
```

Figura 17: Función de lectura de la aceleración

Una vez realizada la adquisición de la aceleración. Hay que definir los *timers* que hemos utilizado en el experimento. Estos *timers* forman parte de la rutina de interrupción para obtener los datos mediante el método definido en el apartado 3.2. Para la obtención de la interrupción del encoder y la constante tendremos que definir dos *timers* distintos.

```
#include <stdio.h>
#include "stm32f4_discovery.h"

#include "slider_config.h"

//-----------------------------------------------------------------------------
void encoder_Init(void) {


    GPIO_InitTypeDef  GPIO_InitStructure;
    TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
    TIM_ICInitTypeDef  TIM_ICInitStructure;
    NVIC_InitTypeDef          NVIC_InitStructure;


    /* PORT E ****************************************/
   RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);

   // PC.6 (TIM8_CH1) ,PC.7 (TIM8_CH2) for quadrature encoder
   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7;
   //GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;   //
   GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_DOWN;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
   GPIO_Init(GPIOC,&GPIO_InitStructure);
   GPIO_PinAFConfig(GPIOC, GPIO_PinSource6, GPIO_AF_TIM8);
   GPIO_PinAFConfig(GPIOC, GPIO_PinSource7, GPIO_AF_TIM8);
```

Figura 18: Inicialización de la interrupción del encoder

```
    /* Prepare timer */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM8,ENABLE);

    TIM_TimeBaseStructInit(&TIM_TimeBaseStructure);
    TIM_TimeBaseStructure.TIM_Period = 0xffff;
    TIM_TimeBaseStructure.TIM_Prescaler = 1;
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseInit(TIM8, &TIM_TimeBaseStructure);
    TIM_EncoderInterfaceConfig(TIM8, TIM_EncoderMode_TI12, TIM_ICPolarity_Rising, TIM_ICPolarity_Rising);

    /* Use the CC1 channel to generate an interrupt in each edge */
    TIM_ICStructInit (&TIM_ICInitStructure);
    TIM_ICInitStructure.TIM_Channel = TIM_Channel_1;
    TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Rising;
    TIM_ICInitStructure.TIM_ICSelection = TIM_ICSelection_DirectTI;
    //TIM_ICInitStructure.TIM_ICPrescaler = TIM_ICPSC_DIV1;  <--- every 8 edges
    TIM_ICInitStructure.TIM_ICPrescaler = TIM_ICPSC_DIV8;
    TIM_ICInitStructure.TIM_ICFilter = 0x0;
    TIM_ICInit(TIM8, &TIM_ICInitStructure);

   // Initial interuppt structures

   NVIC_InitStructure.NVIC_IRQChannel = TIM8_CC_IRQn;
   NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
   NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
   NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
   NVIC_Init(&NVIC_InitStructure);


   NVIC_SetPriority (TIM8_CC_IRQn, (1<<__NVIC_PRIO_BITS) - 1);

   // Clear all pending interrupts
   TIM_ClearFlag(TIM8, TIM_FLAG_Update | TIM_FLAG_COM | TIM_FLAG_Break | TIM_FLAG_CC1 | TIM_FLAG_CC2 |

#ifdef ENCODER_INTERRUPT
   TIM_ITConfig(TIM8, TIM_IT_CC1, ENABLE);   //<---- PARA ACTIVAR INTERRUPCION
#endif

   TIM8->CNT = 100;  // the platfor encorders
   //TIM1->CNT = 11820-2149;  // 1 cm (11820 counts is 55000 micrometers)
   //TIM1->CNT = 100; // my start position value
   TIM_Cmd(TIM8, ENABLE);
```

Figura 19: Configuración de la interrupción del encoder

El segundo *timer* viene definido por la función *Systick* y por la variable Milis. Ambas nomenclaturas corresponden a componentes propias de la STM32F4, por lo que simplemente hay que usar estas herramientas para definir el tiempo de muestreo que queramos. Cada vez que la función *Systick* se activa en el período que hemos definido, se procederá al cálculo de la interrupción dada por el muestreo constante.

Cada vez que se activa la interrupción de muestreo constante se lleva a cabo un cálculo que manda la acción de control a un amplificador operacional, el cual mandará la señal al controlador Escon. Esta parte permite dos cosas, primero obtener las medidas de la interrupción constante, y segundo, la posibilidad de darle a la plataforma un movimiento autónomo para recabar información sin necesidad de moverla.

En esta función también se cargan los datos que se transmiten más tarde a través de TeraTerm. La última función coloca los distintos valores a analizar: tiempo, posición… De tal

manera que queden divididos en columnas y sea más fácil su procesado mediante Matlab o Excel.

```
void slider_PeriodicInterrupt(void) // performed each interruption of system_tick
{
   int32_t encoder_value;
   //float speed;
   float t;


   if (slider_experiment_started) {
    //slider_LoggingAddItem(SOURCE_TIMER, elapsed_Get(),encoder_Read(),acceleratic
    encoder_value = encoder_Read();
    if(encoder_value<65535 && encoder_value>50000)
      {
      Nencoder_value=encoder_Read()-65536;
      }
    else{
        Nencoder_value=encoder_Read();
            }

    speed= (Nencoder_value-encoder_value_old)/dt;
    t = timer_interrupt_counter*PERIODIC_INTERVAL;

    //pref=(p_max-encoder_initial_value)*sin(w*t);
   pref=(p_max)*sin(w*t);
   vref=w*p_max*cos(w*t); //
   ep=pref-Nencoder_value; // error proportionaml
   ev=vref- speed;         // error derivator
   u=Kp*ep+Kv*ev;



     u=u+u_offset;

    motorpower_SetVoltage(u);      // test analog output

// maybe put here slider_loggingAddItem(...) function
     slider_LoggingAddItem(SOURCE_TIMER, pref,Nencoder_value/*encoder_Read()*/,speed,u /*, vref, ep*/);
     encoder_value_old = Nencoder_value;
     timer_interrupt_counter++;

   }
}
```

Figura 20: Función de control y carga de datos

La función llamada al final permite incluir los valores en el documento que se transmite a TeraTerm, como hemos indicado anteriormente, los valores se posicionan en columnas para que posteriormente sea más fácil de pasar a un programa del tipo Matlab para llevar a cabo el procesado de los datos.

```
,,
void slider_LoggingAddItem(
   source_t int_source,
   float elapsed_useconds,
   int32_t platform_position,
   float g,
   float u/*
   float qref,
   float qest*/
     )
{

   if (logging_position < MAX_LOG) {
      logging_storage[logging_position].int_source = int_source;
      logging_storage[logging_position].elapsed_useconds = elapsed_useconds;
      logging_storage[logging_position].platform_position = platform_position;
      logging_storage[logging_position].g = g;
      logging_storage[logging_position].u = u;
     /* logging_storage[logging_position].qref = qref;
      logging_storage[logging_position].qest = qest;*/
      logging_position++;
   } else {
      printf("ERROR: slider_LoggingAddItem() exhausted logging space\n");
   }
}
```

Figura 21: Función de transmisión de datos

Para transmitir los datos al PC se utiliza TeraTerm y un cable USB. Para llevar a cabo esta transmisión tenemos que tener en cuenta los dos tipos de interrupciones que tenemos en nuestro sistema y nombrarlas de la manera correcta para poder asegurarnos de que el sistema está funcionando correctamente. Hemos utilizado las librerías que se asocian con la comunicación a través de USB incluidas en el archivo de la tarjeta STM32.

Se dejan otras variables comentadas en el caso de que se quieran llevar a cabo otro tipo de mediciones. Cabe tener en cuenta que la capacidad de transmisión entre la tarjeta y el PC viene definida por la cantidad de memoria RAM que tenga nuestra tarjeta. En nuestro caso no es posible la transmisión de más de cinco columnas distintas, por lo que no se pueden llevar a cabo todas las mediciones posibles a la vez.

Para la transmisión de la acción de control se han creado dos funciones, una que permite la inicialización de uno de los pines para funcionar como un convertidor digital a analógico y otra función que lleva a cabo la transformación de los datos calculados y el enviado de los mismos al controlador.

```
void motorpower_Init(void)
{
    DAC_InitTypeDef  DAC_InitStructure;
    GPIO_InitTypeDef  GPIO_InitStructure;

    /* PORT A ****************************************/
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);

    /* PA5 as analog input in order to use DAC channel 2 */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    /* DAC Periph clock enable */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);

    /* DAC channel 2 Configuration */
    DAC_InitStructure.DAC_Trigger = DAC_Trigger_None;    // No trigger
    DAC_InitStructure.DAC_WaveGeneration = DAC_WaveGeneration_None;
    DAC_InitStructure.DAC_OutputBuffer = DAC_OutputBuffer_Enable;
    DAC_Init(DAC_Channel_2, &DAC_InitStructure);

    /* Enable DAC Channel1 */
    DAC_Cmd(DAC_Channel_2, ENABLE);
}


void motorpower_SetVoltage(float volts)
{
  // uint16_t data;

   if (volts >= VOLT_REF) {
      data = 4095;
   } else if (volts < 0) {
      data = 0;
   } else {
      data = (volts/VOLT_REF) * 4095;
   }

    DAC_SetChannel2Data(DAC_Align_12b_R, data);
}
```

Figura 22: Inicialización y configuración de las funciones de control de motor

Para establecer un funcionamiento correcto de la comunicación, hemos desarrollado un sistema de LEDs que indica la parte del experimento en la que nos encontramos actualmente. Para ello hemos llevado a cabo la inicialización de las funciones que nos permiten apagar y encender los LEDs.

```
void LED_Init(void)
{
   GPIO_InitTypeDef  GPIO_InitStructure;  // estructura donde se pone la configuración deseada

   RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOG, ENABLE); //

   /* dar los detalles del puerto/pin */
   GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_14 | GPIO_Pin_13;
   GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_OUT;      // lo vamos a usar como salida
   GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;      // queremos que sea push-pull
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_25MHz;  // a 100 MHz
   GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_NOPULL;   // desactivar los pull-up, no hacen falta

   GPIO_Init(GPIOG, &GPIO_InitStructure);              // hacer efectiva configuración puerto
}
void LED_RedOn(void)
{
    GPIO_SetBits(GPIOG, GPIO_Pin_14);
}

void LED_RedOff(void)
{
  GPIO_ResetBits(GPIOG, GPIO_Pin_14);
}

void LED_GreenOn(void)
{
    GPIO_SetBits(GPIOG, GPIO_Pin_13);
}

void LED_GreenOff(void)
{
     GPIO_ResetBits(GPIOG, GPIO_Pin_13);
```

Figura 23: Inicialización y configuración de los LEDs

Una vez definidas todas las funciones relevantes para el funcionamiento del experimento, hemos realizado una función principal, un *main*, en el que todas las funciones entran en común. Lo primero es definir las variables que entran en juego durante la experiencia, siendo las más importantes las que nos proporcionan las medidas tomadas por el encoder.

```
void slider_EncoderInterrupt(void);


uint32_t val=0;
static char cadena[100];
extern uint8_t slider_experiment_started;

extern int32_t encoder_initial_value, encoder_value_old;


int main(void)
{

   int32_t encoder_value;           //  previously uint
   uint32_t j, aux;
   uint32_t i;
   float data_read;


   slider_experiment_started = 0;  //risk free


   SystemInit();

   TM_DELAY_Init();
   TM_USB_VCP_Init();
```

Figura 24: Primera parte de Main, definiciones

Hemos llevado a cabo la inicialización de todas las funciones relevantes para el experimento. Inicializamos las funciones que llevan a cabo las interrupciones, las que se encargan de leer los datos, de transmitirlos…

```
   LED_Init();
   elapsed_Init();
   elapsed_Get();
   encoder_Init();
   slider_LoggingInit();
   slider_PID_Init();
   acceleration_Init();
   motorpower_Init();
   encoder_initial_value=encoder_Read();
```

Figura 25: Segunda parte de Main, inicializaciones

La parte de la adquisición de datos es la definida a continuación. Primero entran en funcionamiento los LEDs, se enciende el rojo hasta que el cable USB esté conectado a la tarjeta

y al PC, de este modo nos hemos asegurado de que no haya pérdida de datos y actúe como un seguro. Después tenemos que esperar a que movamos la plataforma, mientras tanto el *timer* irá tomando datos para asegurarnos de que el sistema periódico se mantiene.

Los datos se recaban desde que empieza a moverse la plataforma hasta el tiempo especificado, dependiendo de los datos o velocidades que estemos probando hemos cambiado el tiempo de duración del experimento.

Una vez la duración del experimento ha finalizado, se lleva a cabo la transmisión de los datos. El LED rojo se vuelve a encender hasta que todos los datos han sido almacenados a través de TeraTerm en el PC.

```
// adquisition logic

// STEP 1: wait for USB connection
LED_RedOn();  // wait for USB connection
printf("USB: waiting for connection\n");
//while (TM_USB_VCP_GetStatus() != TM_USB_VCP_CONNECTED) {}; //mi
printf("USB: connected\n");
LED_RedOff();

// STEP 2: wait for adquisition when platform moved
LED_GreenOn();
encoder_value = encoder_Read();
//while (encoder_Read() == encoder_value); //mi à voir

encoder_initial_value = encoder_Read();
encoder_value_old = encoder_initial_value;


elapsed_Init();
elapsed_Get();
slider_experiment_started = 1;
LED_GreenOff();

// STEP 3: wait 10 seconds, we assume your experiment is done
for (i=0; i<20; i++) {slider_Delayms(1000);} // while its ewecuted the interruption
                                             // will happens in TIM8 and system clockconfig
```

Figura 26: Tercera parte de Main, lógica de adquisición

```
// STEP 4: dump data to the serial terminal
slider_experiment_started = 0;
LED_RedOn();
slider_LoggingDump();
LED_RedOff();

// Done!!!!
while(1);


aux = 0;

while(1) {
   encoder_value = encoder_Read();

   data_read = TM_SDRAM_ReadFloat(aux*4);
   //data_read = 0.123F;

   aux++;
   if (aux > 10000) {
     aux = 0;
   }

   sprintf(cadena,">> %f %d\r\n",data_read,(int)encoder_value);
 TM_USB_VCP_Puts(cadena);
```

Figura 27: Cuarta parte de main, lógica y carga de datos

4.1.5. Elementos adicionales.

En el experimento hemos incluido elementos adicionales para asegurarnos del correcto funcionamiento. Tenemos dos fuentes de alimentación externas a parte de la propia tarjeta de adquisición. Hemos utilizado una fuente de 24 V para alimentar el motor, y hemos utilizado una fuente de +-15 Voltios para alimentar un amplificador operacional por el que pasa la señal de acción de control.

El amplificador operacional tiene una ganancia de 1.67. El valor máximo de la acción de control es de 3.33 V por lo que una vez haya pasado la señal por el amplificador operacional estamos trabajando en un rango entre 0 y 5 Voltios, lo cual regula la velocidad del motor.

También hemos incluido un acelerómetro. En el caso de nuestro experimento, este acelerómetro probó ser defectuoso y nos privó de la posibilidad de comprobar si los resultados obtenidos se correspondían completamente con lo medido a través del filtro de Kalman, no obstante, hemos considerado la inclusión del acelerómetro como una parte importante a mantener para mejorar el funcionamiento del experimento.

4.2. Montaje de la experiencia.

Para llevar a cabo el montaje de la experiencia hemos necesitado primero el motor y la plataforma deslizante. El motor se acopla en el centro de la plataforma de tal manera que la permite desplazarse libremente entre los topes de los que dispone la misma.

Una vez el motor está acoplado a la plataforma, hemos de colocar el encoder con cuidado, de tal manera que no se produzcan errores de desalineamiento. Estos errores pueden llevar al mal funcionamiento del encoder y por lo tanto a una incorrecta medición y resultados falsos [5].

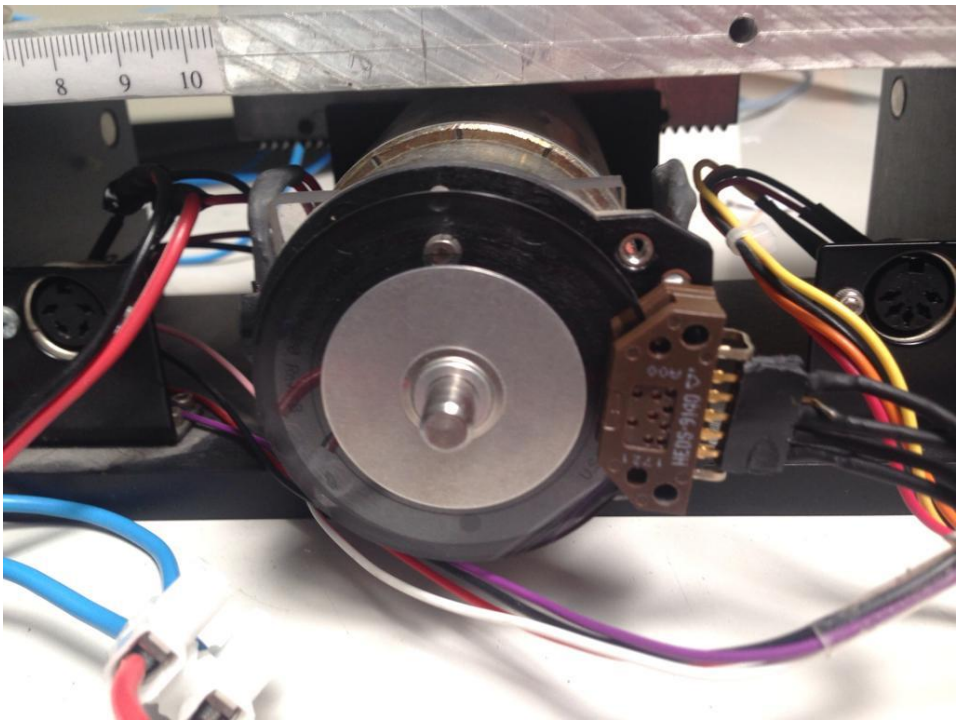

Figura 28: Encoder sobre el servomotor

Hemos conectado al encoder cables siguiendo la arquitectura de los pines que hemos especificado anteriormente. El voltaje de alimentación viene dado por la tarjeta de adquisición, lo mismo con la señal de tierra. El canal A y el canal B se conectan a los pines C6 y C7.

El acelerómetro LIS344ALH está dispuesto en la parte superior de la plataforma. El acelerómetro ha sido configurado para que calcule la aceleración en el eje en el que hemos movido la plataforma a lo largo del experimento, para asegurar el correcto funcionamiento. El

acelerómetro lo hemos alimentado a través de la tarjeta de adquisición, la señal de salida está conectada a la tarjeta a través del pin PC1.

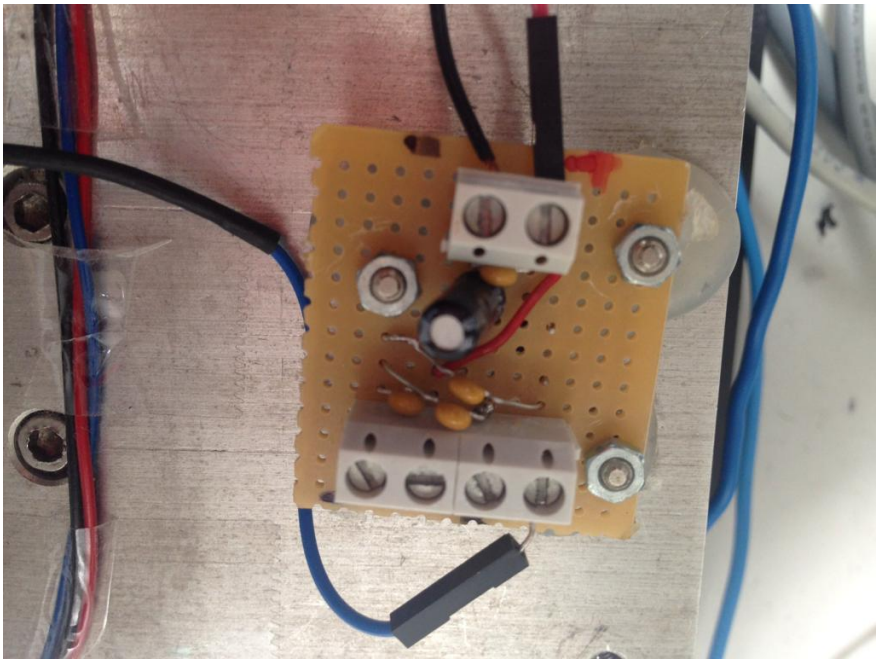

Figura 29: Acelerómetro situado sobre la plataforma

El amplificador operacional recibe alimentación de la fuente externa del laboratorio que tiene 15 voltios tanto positivos como negativos. La señal para amplificar viene desde la tarjeta de adquisición, del pin A5, definido en la función de inicializar el DAC. La salida del amplificador operacional se conecta al pin que hemos configurado como entrada analógica en el controlador ESCON.

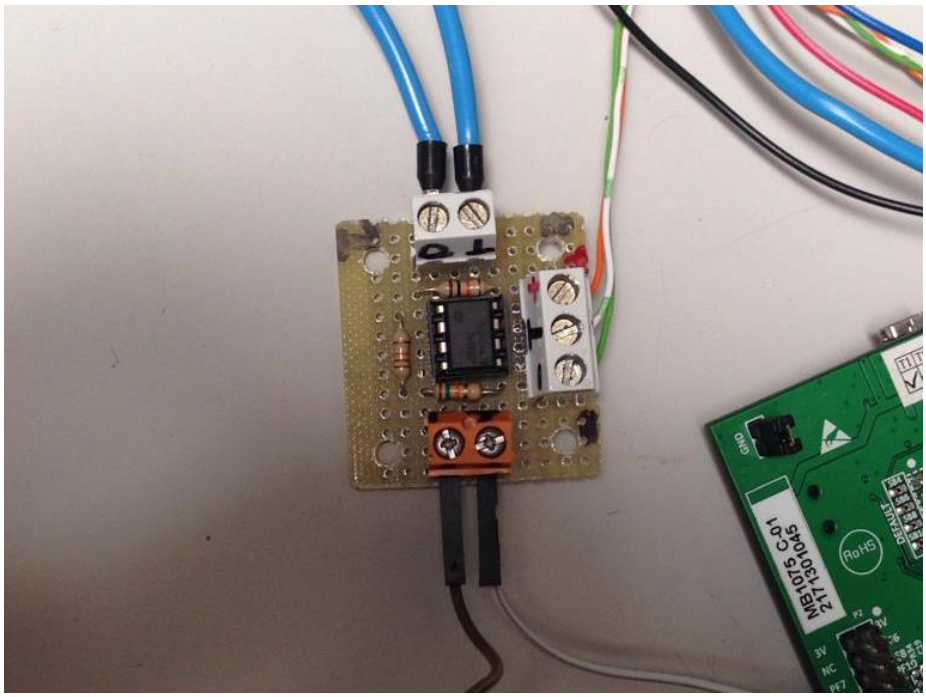

Figura 30: Amplificador Operacional

El controlador recibe la alimentación directamente de la fuente externa de 24 voltios. También tiene activado el modo de trabajo mediante el cortocircuito de sus entradas digitales que hemos configurado. A través de los pines de salida analógica transmite la señal de control de nuevo al motor y por lo tanto cambia el movimiento de la plataforma.

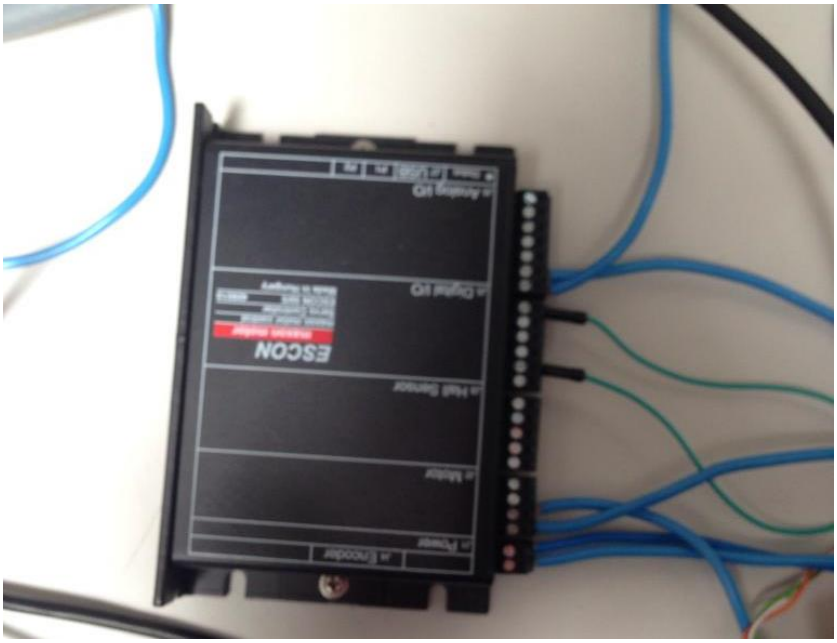

Figura 31: Controlador Escon y conexiones

El último paso es conectar la tarjeta STM32 mediante su alimentación al PC que estemos utilizando y conectar TeraTerm para que en el momento que empiece el experimento ya esté todo configurado para la recogida de datos. Cuando hemos alcanzado el momento de preparación, conectamos el cable de comunicación a la tarjeta de adquisición y comenzamos el experimento.

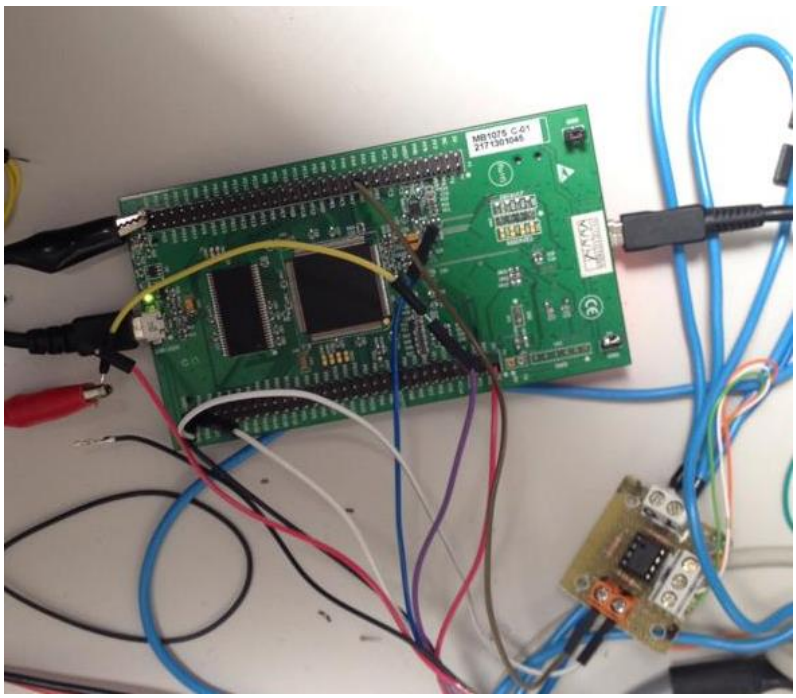

Figura 32: Tarjeta STM32F4

4.3. Funcionamiento de la experiencia.

Siguiendo las indicaciones marcadas en el apartado anterior y cerciorándonos de que hemos conectado todos los pines siguiendo la configuración que hemos programado, podemos comenzar el experimento, pero primero hemos utilizado un osciloscopio que hemos conectado a los bornes del canal A y B del encoder para asegurarnos su correcto funcionamiento a lo largo de la experiencia.

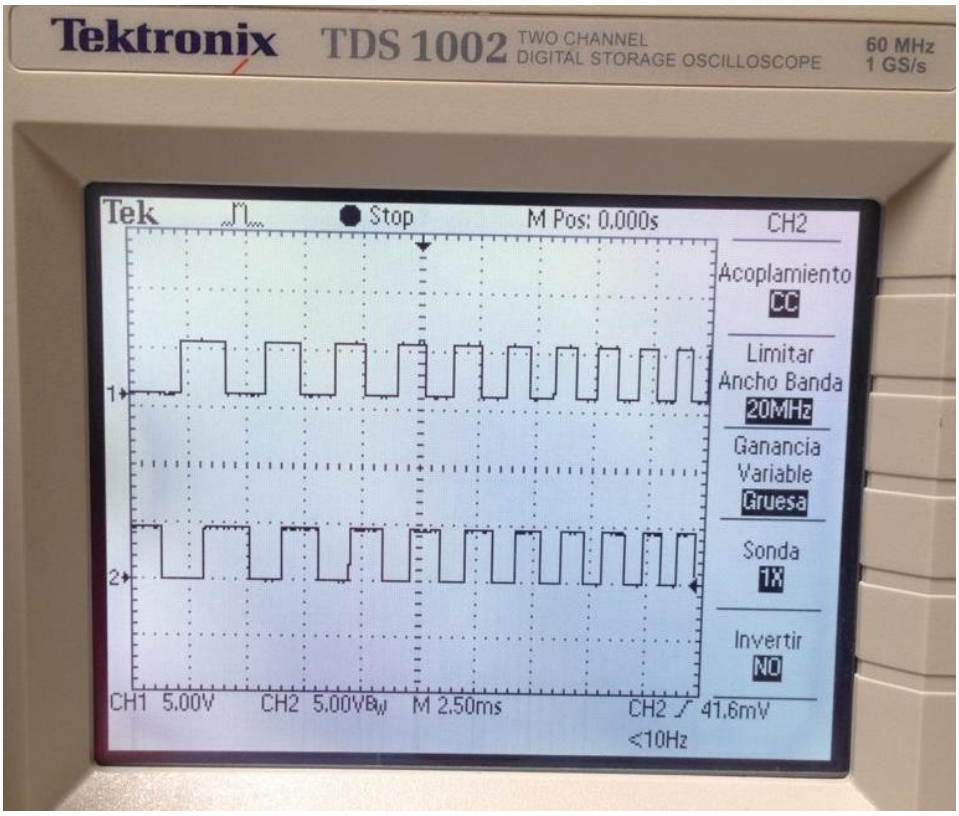

Figura 33: Señal de cuadratura del encoder

Una vez el correcto funcionamiento del *encoder* ha sido comprobado, procedemos a la conexión de los cables de alimentación y comunicación. Esperamos a que los LEDs se apaguen y realizaremos dos experiencias.

La primera experiencia es automática, para obtener las fuerzas medidas por el acelerómetro. Como hemos indicado en el apartado de comunicación, el motor recibirá una señal senoidal de referencia que hemos utilizado para comprobar el funcionamiento del filtro.

La segunda experiencia es manual, hemos desactivado toda la parte del código que se encarga de dar una acción de control al experimento. Hemos movido la plataforma con la mano siguiendo una trayectoria similar a la de la función senoidal para recrear más o menos el mismo funcionamiento.

Cuando el LED rojo se apague, conectamos el cable de comunicación. En el momento que el LED verde se encienda empezamos a mover la plataforma y el experimento comienza. Pasado un tiempo empezaremos a recibir en la terminal de TeraTerm los datos que estamos transmitiendo a través del cable USB.

Los datos que se transmitan por TeraTerm se guardan en el lugar elegido para el almacenamiento de datos. Estos datos luego se transforman de un archivo separado por comas a un .xlsx mediante el uso del programa Excel, esto permite que luego el archivo pueda ser leído por Matlab.

En el archivo .xlsx tenemos la oportunidad de ver el tipo de interrupción que ha provocado la medición, el tiempo en microsegundos en el que se ha tomado la medición y la posición de la plataforma en micrómetros.

Encoder	29954783	13997
Encoder	29955333	14034
Encoder	29955885	14072
Encoder	29956434	14109
Encoder	29956978	14146
Encoder	29957518	14183
Encoder	29958054	14220
Timer	29958306	14234
Encoder	29958591	14258
Encoder	29959119	14295
Encoder	29959646	14332
Encoder	29960171	14369
Encoder	29960693	14407
Encoder	29961215	14444

Tabla 4: Extracto de datos obtenidos del encoder

5. Análisis de los resultados.

Como hemos mencionado anteriormente, hemos recabado información en la forma de un documento Excel listo para procesar a través de Matlab. Como hemos indicado en el apartado 3.2, ahora implementaremos un filtro de Kalman para estimar la velocidad y la aceleración. Utilizamos matrices de 3x3 ya que el vector de estado que estamos utilizando necesitará de variable posición, velocidad y aceleración para poder llevar a cabo una estimación correcta.

En cuanto al acelerómetro, la intención era comprobar los resultados, no obstante, el acelerómetro posee una cantidad de ruido que imposibilita la verificación de los datos de manera absoluta.

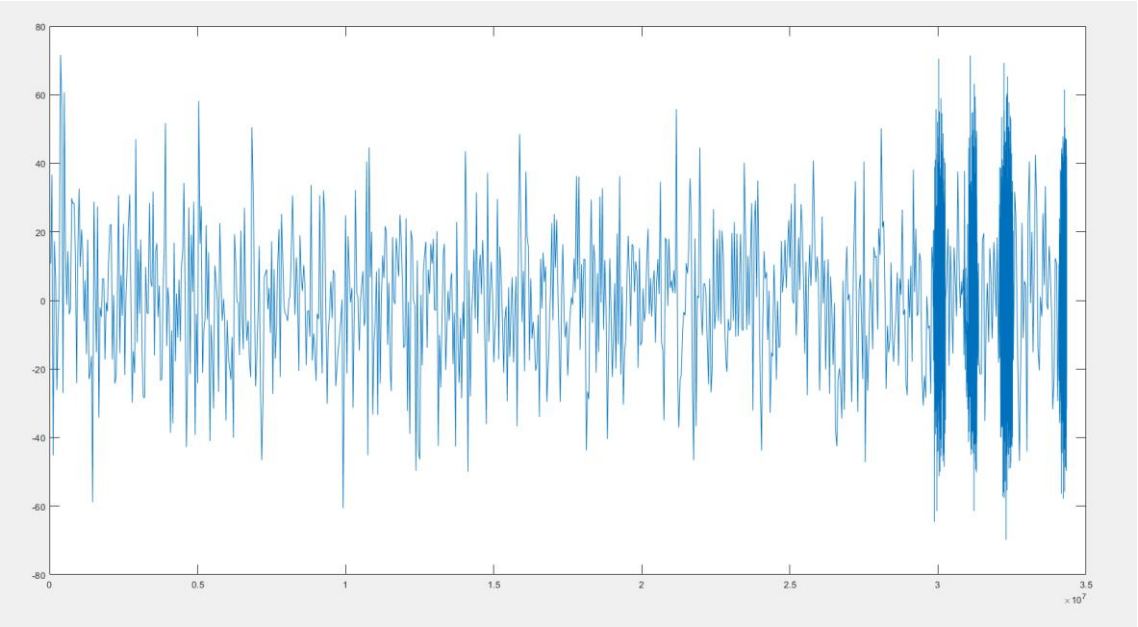

Figura 34: Ruido del acelerómetro

Hemos establecido los parámetros del filtro de Kalman tal y como los hemos definido en el apartado teórico. Introduciremos el vector de tiempos calculado y el vector de posiciones calculado, y, en base a las matrices de covarianza y de estado que han sido calculadas llevaremos a cabo el filtrado. De este modo obtendremos unas gráficas y tablas que representan los valores obtenidos de la aceleración y de la velocidad.

Hemos definido todas las variables en base al documento de Excel generado. Sabemos que la posición inicial es 1000 micro metros, pero de cara a probar el funcionamiento del filtro de Kalman le hemos dado una posición un poco distinta, de 0 micro metros, con el fin de probar su funcionamiento.

```
cuadrante=1;
p_enc=0;
%pos2=p_enc;
n_periodos=0;
tiempo_anterior=0;
%tiempo=0;
tiempo_enc=0;
tiempo_accel=0;
tiempo_accel_ant=0;
incrementa_encoder=1;
var_a=0.1;
var_a=1;
var_a=100;

%var_a=0.0001;

% Posicion, velocidad y aceleración inicial
p=0;
pos_vect=0;
v=w;
a=0;
r1=1;
%w1=.1
%w2=10;
betha=1;
```

Figura 35: Condiciones iniciales de Kalman

Después hemos definido las matrices de estado y de covarianza que permitirán que el filtro de Kalman funcione conforme a nuestro sistema. Es importante utilizar un sistema con matrices 3x3, por lo normal el filtro de Kalman utiliza matrices de 2x2, en nuestro caso es necesario el primer caso debido a la intención de estimar posición, velocidad y aceleración.

```
W1=[0 0 w1]';
%W=W1*W1';
%W=zeros(4,4);

W=diag(W1)

A=[0 1 0; 0 0 1; 0 0 0];
B=[0 0 0]';
C=[1 0 0 ];
D=0;
Gamma=[0 0 1]';
%M=[1 0 0 0;0 0 0 0;0 0 0 0;0 0 0 0];
M=eye(3);
X=[0 0 0]';   % Valor inicial para el predictor;
P= eye(3);      % Matriz de covarianzas
X_corector=[0 0 0]';
Kvect=[];
X_vect=[];
a_vect=[];
v_vect=[];
%res_enc=0.000001;
%cte=(res_enc/10)^2/3;
pos2=Data(:,3);
N=length(Time_t);
p_vect_Kalman=[];
```

Figura 36: Matrices del filtro de Kalman

Una vez todas las matrices y variables se encuentran definidas, hemos introducido las actualizaciones del predictor y del corrector, que son la parte principal del filtro de Kalman. Se realizan actualizaciones en cada iteración, primero se predice la posición, y después, en base al error generado, se corrige, con el fin de crear un filtro con errores mínimos y predicciones precisas.

```
for i=1:N-1

   Y=p_timer(i)';
   Ts=dt(i);
   Qd=c2dnoise(A,Gamma,w1,Ts);

   Ad=expm(A*Ts);
   X_predictor=Ad*X_corector;
   P=Ad*P*Ad'+Qd;


   R=cte;
    K=P*C'*inv([R+C*P*C']);
 X_corector=X_predictor+K*(Y-C*X_predictor);
   P=(eye(3)-K*C)*P;
   pos=X_corector(1);
      %pause
    %
    p_vect_Kalman=[ p_vect_Kalman;pos];
     pos_vect=[pos_vect;pos];
    Kvect=[Kvect,K'];
    X_vect=[X_vect;X_corector'];
     a_vect=[a_vect,X_corector(3)];
     v_vect=[v_vect,X_corector(2)];

end
```

Figura 37: Corrector y predictor de Kalman

Los valores de la estimación tanto de la velocidad como de la aceleración se almacenan en sus correspondientes vectores y luego se representan gráficamente para comprobar si el funcionamiento es correcto. Primero se han representado la posición medida contra la posición estimada y después hemos comparado las gráficas de velocidad y aceleración estimadas con las gráficas de velocidad y aceleración obtenidas mediante la doble derivada. En el caso de la aceleración hemos incluido la gráfica del acelerómetro para comprobar la precisión del filtro. Se llevaron a cabo dos pruebas con dos movimientos distintos.

EL

La primera prueba:

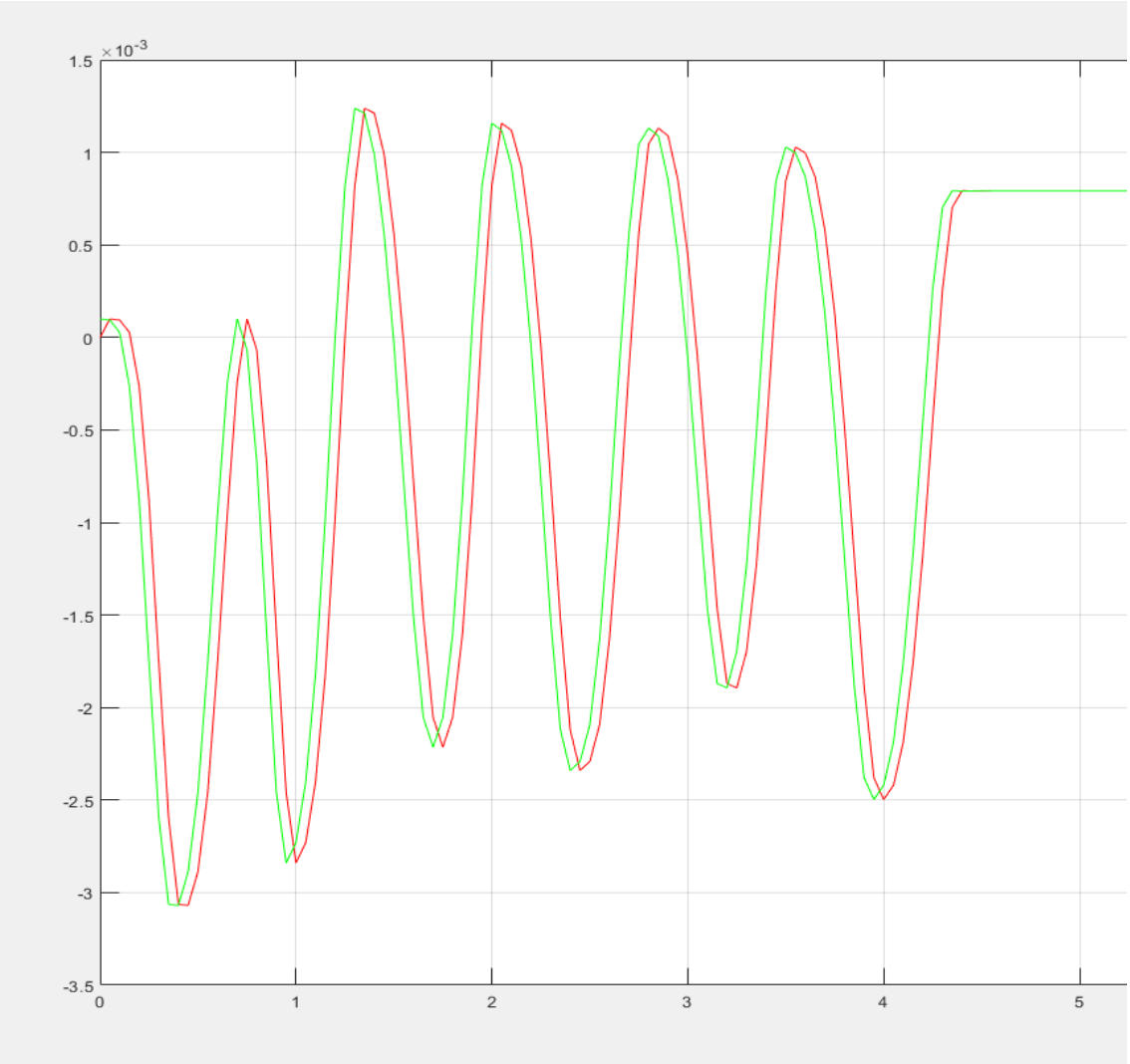

Figura 3835: Prueba 1 Posición medida (rojo) y posición de Kalman (verde)

Cómo se puede observar en la figura 36, el modelo de Kalman es correcto a la hora de predecir y corregir sus resultados, obteniendo una figura muy similar a la posición medida a través del encoder.

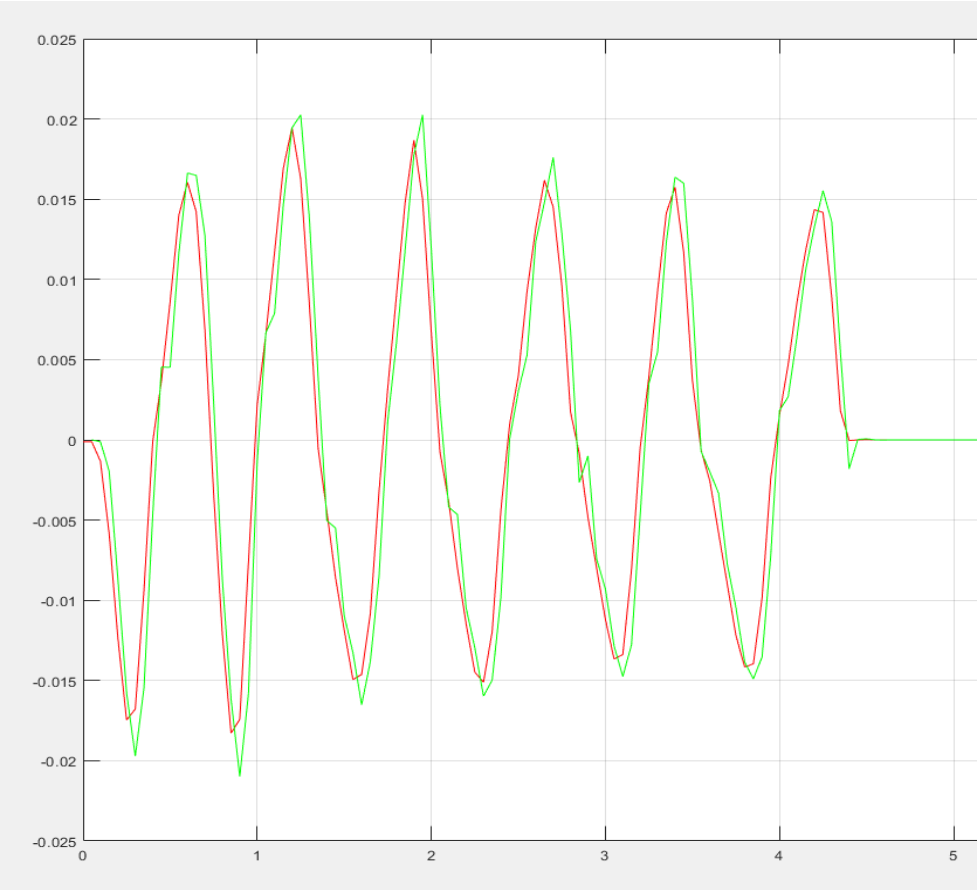

Figura 39: Prueba 1 Velocidad derivada (rojo) y velocidad de Kalman (verde)

En la figura 37 hemos observado que el modelo de la velocidad derivada y la obtenida a partir de Kalman son similares, no obstante, si nos dirigimos a la figura 38 podemos observar que los picos de velocidad de la doble derivada consisten en el muestreo constante mientras que en los datos de velocidad estimada existe una tarea de ajuste a los datos muestreados.

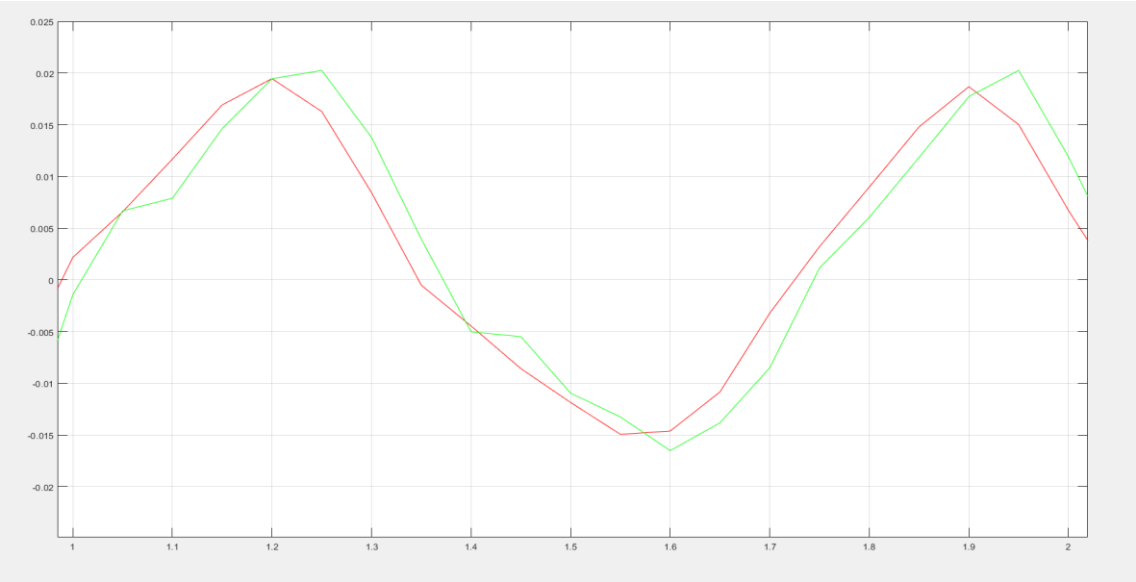

Figura 40: Ampliación de la figura 37

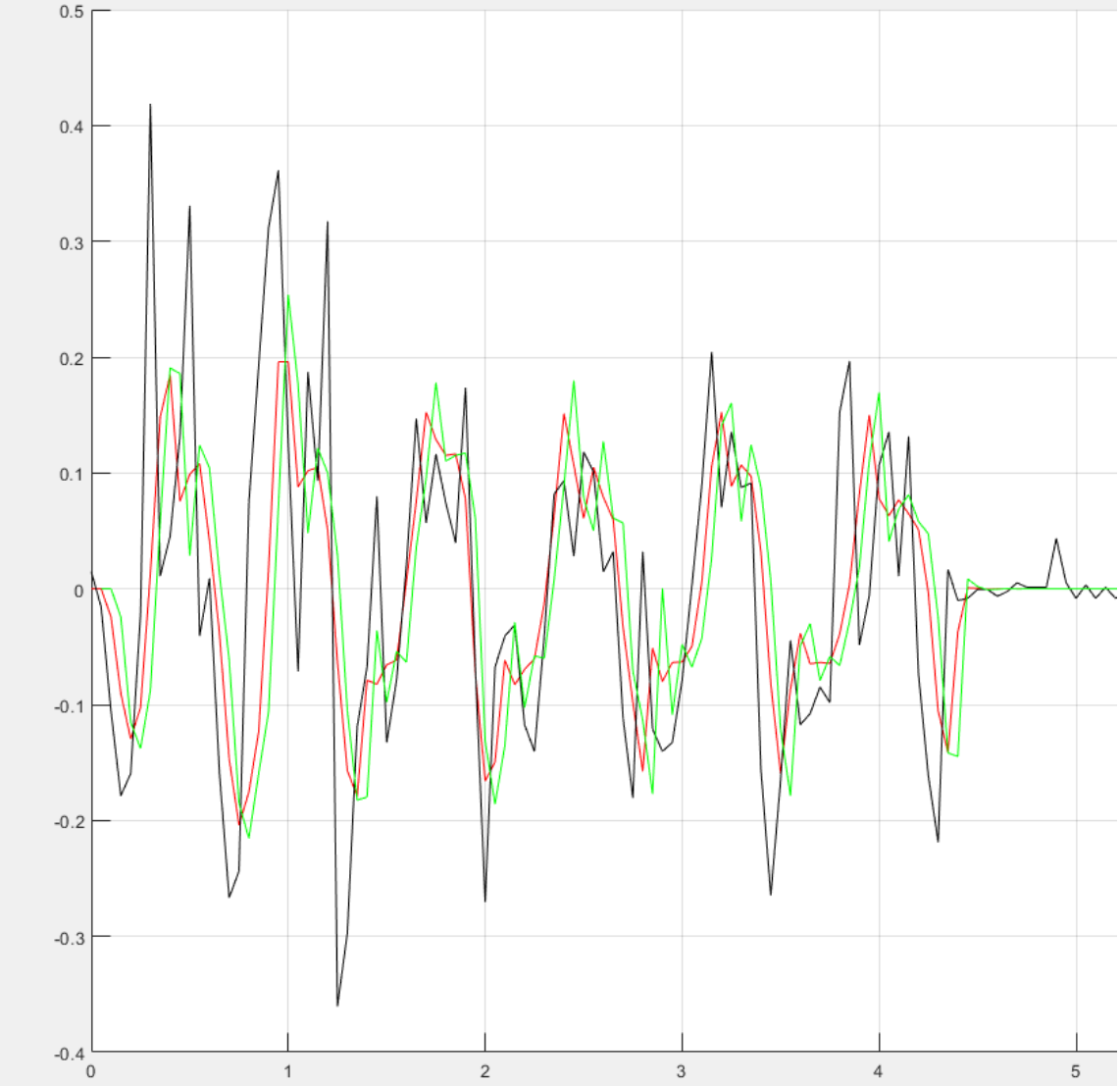

Figura 4136: Prueba 1 Aceleración doble derivada (rojo), aceleración de Kalman (verde) y medida acelerómetro (negro)

En la figura 39 podemos apreciar que la medida obtenida a partir del acelerómetro (negro) tiene una pequeña cantidad de ruido, pero también podemos apreciar que tanto la doble derivada (rojo) como el filtro de Kalman (verde) dan estimaciones con menor varianza. Como podemos observar en la figura 40, aunque el modelo de derivada se asemeje ligeramente a la medición del acelerómetro, podemos apreciar claramente como el filtro de Kalman realiza una corrección que coincide más con la medida obtenida por el acelerómetro.

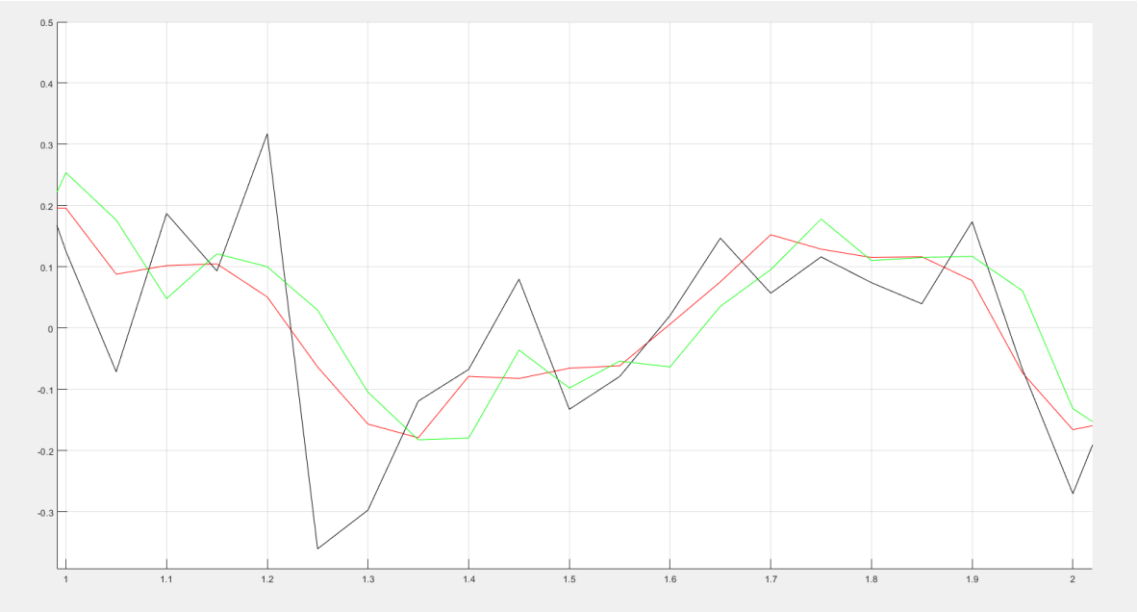

Figura 4237: Ampliación de la figura 39

En la segunda prueba se siguió un patrón de movimiento distinto, con más dinamismo en una menor cantidad de tiempo, para comprobar la respuesta de Kalman en distintas condiciones.

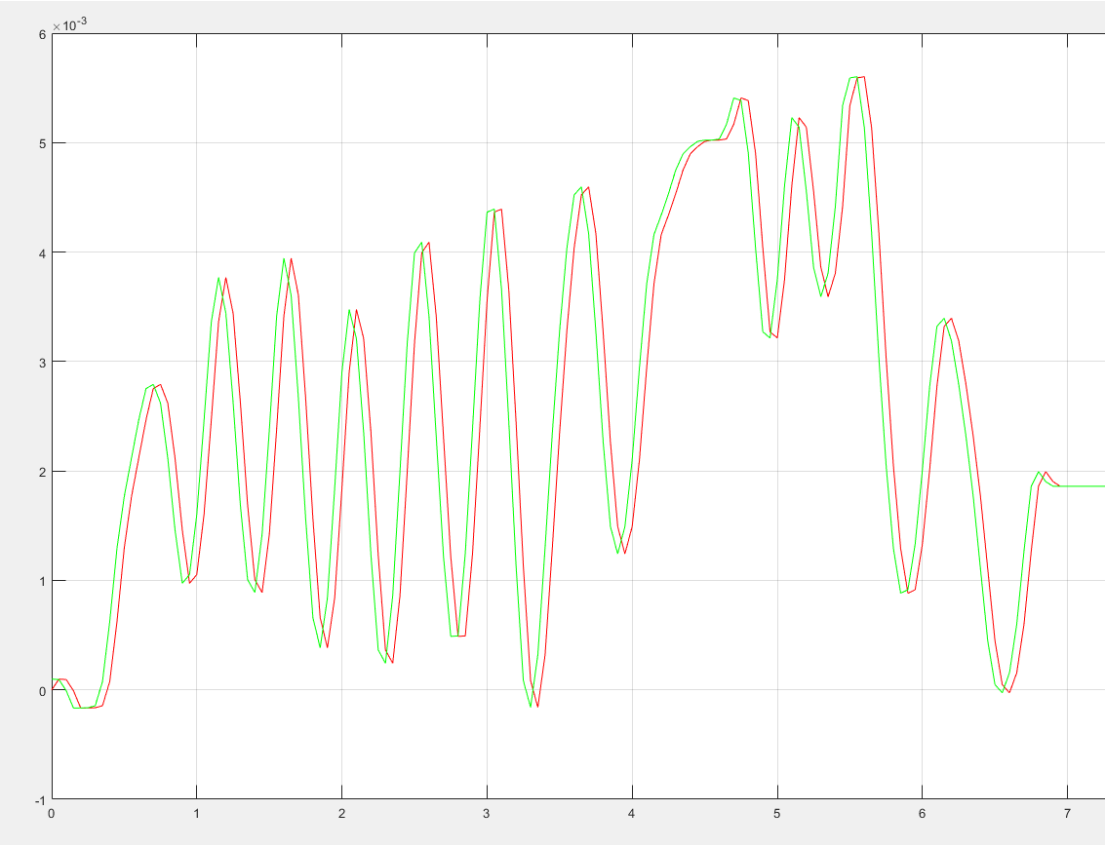

Figura 383: Prueba 2 Posición medida (rojo) y posición de Kalman (verde)

En la figura 41 podemos apreciar, de la misma manera que en la figura 36, que la estimación mediante *Kalman* produce un resultado fiable, que se corresponde con la posición medida.

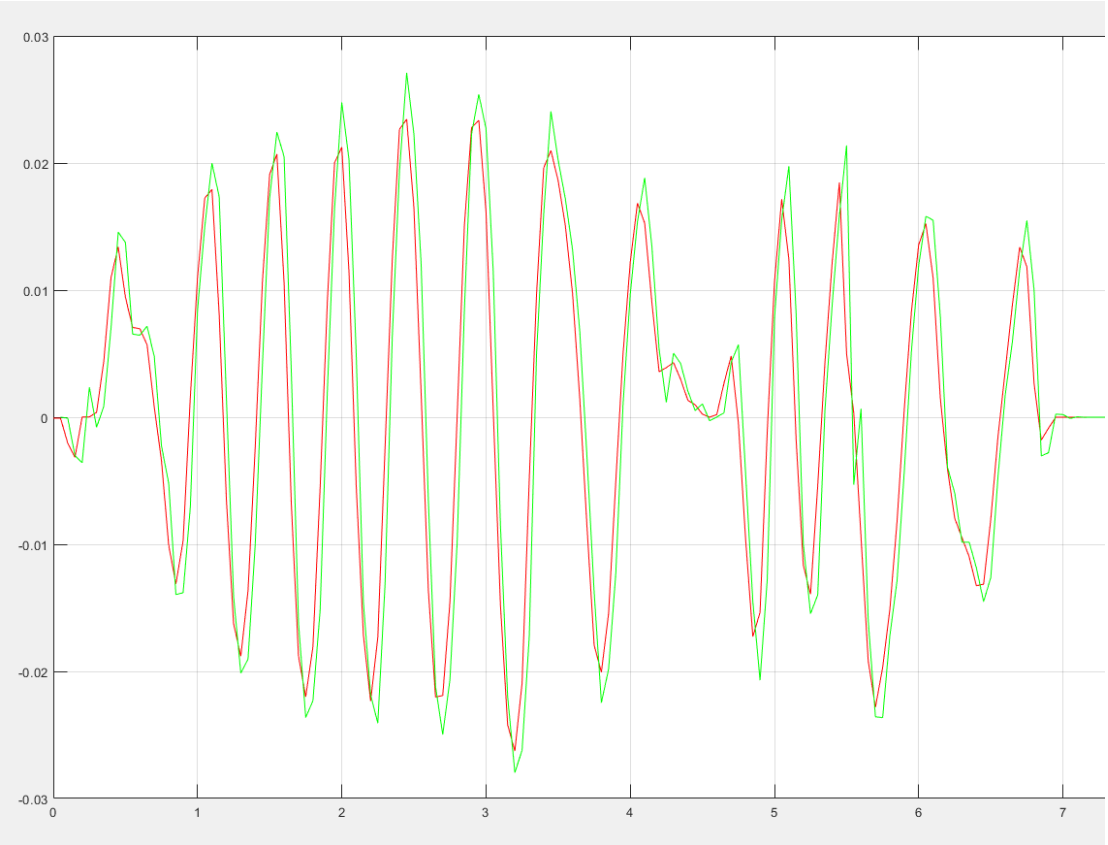

Figura 44: Prueba 1 Velocidad derivada (rojo) y velocidad de Kalman (verde)

Aunque el movimiento haya variado ligeramente, comprobamos que al igual que en la primera prueba, la estimación de la velocidad se realiza correctamente. En la figura 43 apreciamos, de la misma manera que en la figura 38, la tarea del filtro de Kalman a la hora de ajustar su estimación.

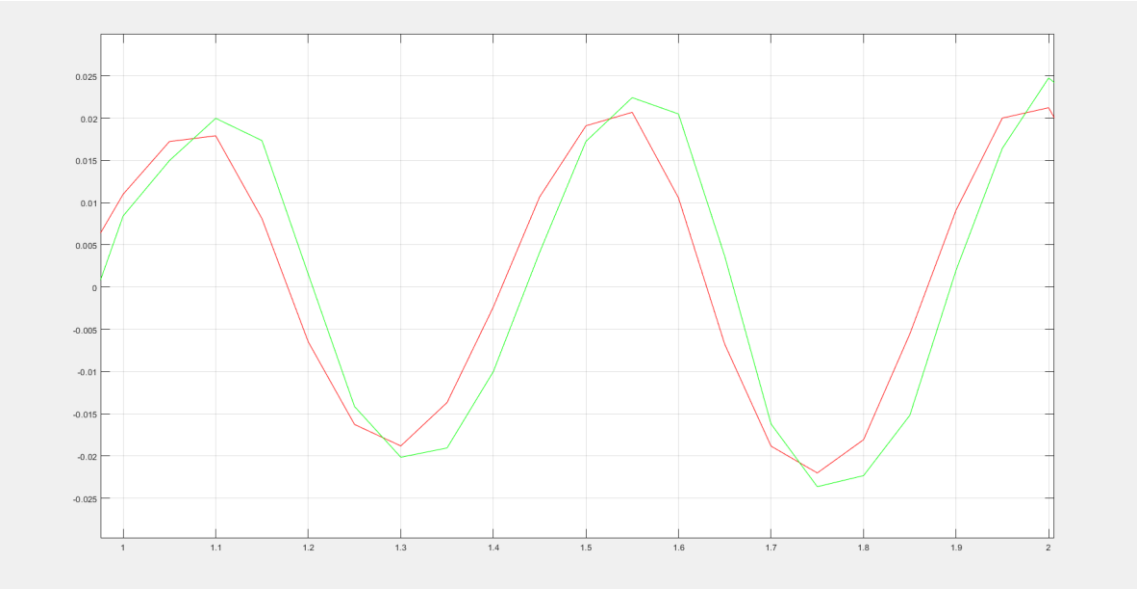

Figura 4539: Ampliación de la figura 42

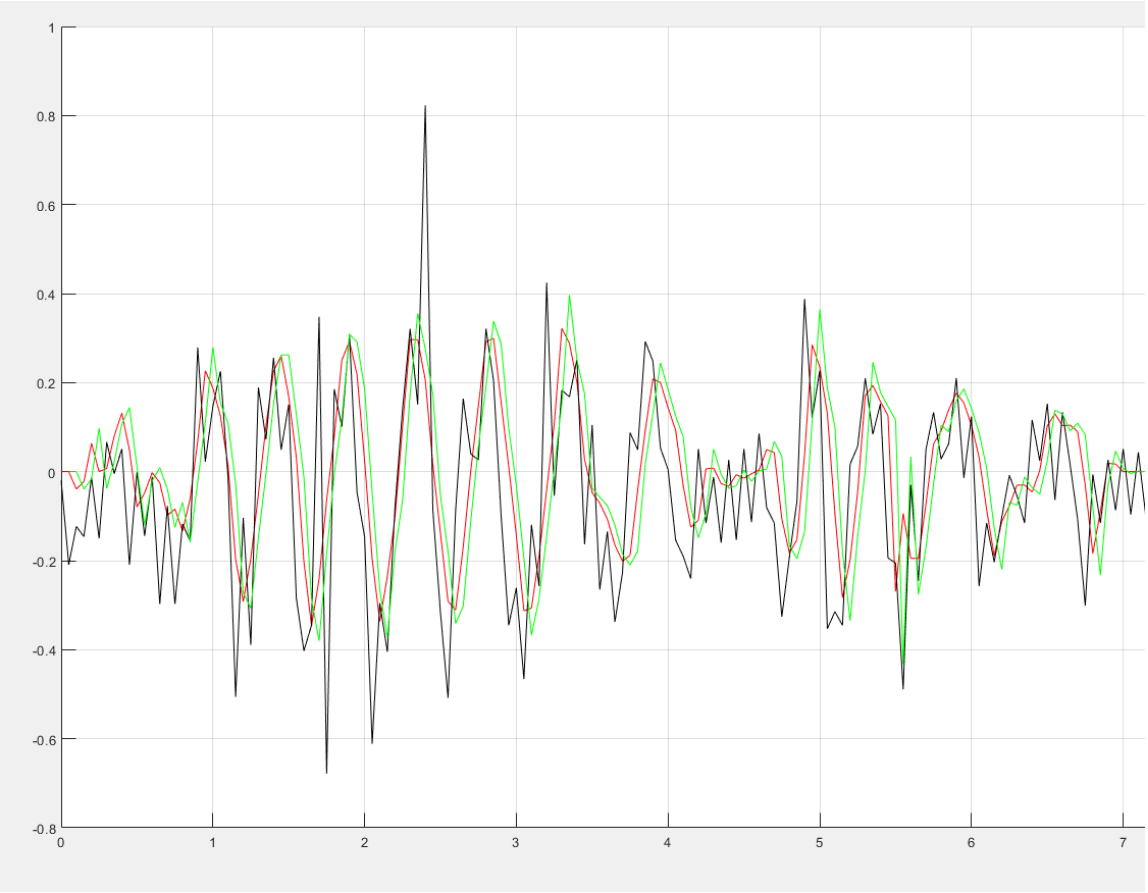

Figura 4640:Prueba 2 Aceleración doble derivada (rojo), aceleración de Kalman (verde) y medida acelerómetro (negro)

En la figura 44 podemos observar el comportamiento errático del sistema a lo largo de esta parte debido al movimiento más rápido que en la prueba anterior. Es difícil comprobar el funcionamiento del sistema en esta gráfica, pero siguiendo el código de colores establecido, hemos ampliado la figura en la número 45. En esta figura podemos apreciar lo indicado anteriormente, similares mediciones en la derivada, pero el filtro de Kalman establece un ritmo de predicción y corrección que se asemeja más al realizado por el acelerómetro.

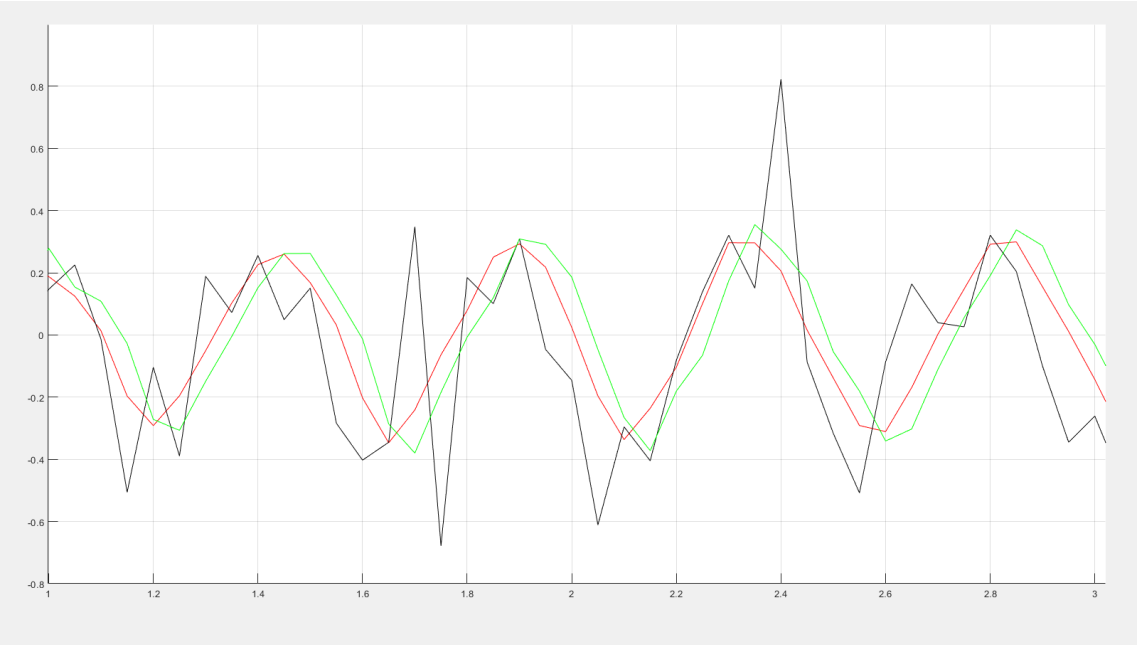

Figura 47: Ampliación de la figura 44

6. Conclusiones.

La obtención de los datos, junto a la programación y el montaje de la experiencia han probado ser las áreas más complicadas, la necesidad de precisión a la hora de obtener los datos y de lograr un correcto funcionamiento del sistema ha sido sin duda la parte que más problemas nos ha dado. La parte de análisis y proceso de los datos ha sido fructífera y más sencilla e intuitiva.

El uso del acelerómetro resulta ser una parte esencial, nos ha aportado datos relativamente fiables de cara a la estimación de las distintas variables. No obstante, es importante tener en cuenta que con un acelerómetro de mayor calidad se puede obtener mayor precisión a la hora de la comprobación.

El uso de un *encoder* de mayor precisión permitiría una predicción de la velocidad y la aceleración con menor error, era una condición asumible, pero el hecho de comprobarlo indica que nuestro sistema se corresponde con el funcionamiento real.

En definitiva, la obtención y análisis de datos, aunque compleja en naturaleza y cantidad de elementos para tener en cuenta, ha sido exitosa. Se han obtenido velocidades y aceleraciones que reflejan con mayor exactitud lo indicado por el acelerómetro, aunque no haya cumplido todas las premisas que quería resolver, ha sido exitoso en su mayor medida.

Aunque hubiese tenido más sentido implementar una señal del tipo senoidal para comparar con el movimiento de la mano (matriz B=0), el *encoder* con el que se realizaba la experiencia se quemó, imposibilitando esta situación, forzándonos a realizar el experimento mediante el movimiento manual.

Aunque el filtro de Kalman funciona correctamente, no existe manera de discernir si su funcionamiento es mejor que la doble derivada, debido a su similitud y la imposibilidad de verificar el funcionamiento con el acelerómetro. En el caso de que el *encoder* no se hubiese quemado habríamos podido llevar a cabo pruebas más exhaustivas con Kalman y mejorar su funcionamiento.

7. Bibliografía.

[1] Velocity and acceleration estimation for optical incremental encoders
R.J.E. Merry *, M.J.G. van de Molengraft, M. Steinbuch
Eindhoven University of Technology, Department of Mechanical Engineering, Control Systems Technology Group, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

[2] Venema, S. C. (1994). A Kalman filter calibration method for analog quadrature position encoders. Master's thesis, University of Washington, Seattle, Washington, USA.

[3] Hagiwara, N., Suzuki, Y., & Murase, H. (1992). A method of improving the resolution and accuracy of rotary encoders using a code compensation technique. IEEE Transactions on Instrumentation and Measurements, 41(1), 98–101.

[4] Mayer, J. R. R. (1994). High-resolution of rotary encodes analog quadrature signals. IEEE Transactions on Instrumentation and Measurements, 43(3), 494–498.

[5] Quick assembly two and three channel optical encoders, technical data heds-554x (2004) 15