Proyecto Fin de Grado

Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Análisis y Diseño de un Control Bio-inspirado de un Robot para la aproximación a un obstáculo

Autor: Francisco Miguel Martín Barbero

Tutor: José Ángel Acosta Rodríguez

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

Sevilla, 2019

Proyecto Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Análisis y Diseño de un Control Bio-inspirado de un Robot para la aproximación a un obstáculo

Autor:
Francisco Miguel Martín Barbero

Tutor:
José Ángel Acosta Rodríguez
Profesor titular

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2018

Proyecto Fin de Grado: Análisis y Diseño de un Control Bio-inspirado de un Robot para la aproximación a un obstáculo

Autor: Francisco Miguel Martín Barbero

Tutor: José Ángel Acosta Rodríguez

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2018

El Secretario del Tribunal

A mi familia

A mis maestros

A mis compañeros

A mis amigos

Agradecimientos

En primer lugar, me gustaría agradecerle a mi familia todos estos años, en los que me han estado apoyando sin importar mis elecciones sino mis actos. Por todo lo que han hecho para que pudiera seguir adelante y acabar llegando hasta aquí.

También, agradecer a todas las personas que han conseguido crearme este interés por el conocimiento y que me inculcaron su amor por la ciencia, ya fueran profesores o no. Además, de a todos los profesores que me han ayudado a crecer tanto en conocimiento como en persona.

Por último, a todos aquellos amigos que me han ayudado a seguir adelante en todo momento, a los que siguen y a los que se fueron.

Francisco Miguel Martín Barbero

Sevilla, 2019

Resumen

En este proyecto se tratará de imitar el comportamiento animal, en concreto de aves, sobre como realizan la parada ante el deseo de detenerse en un lugar específico o detenerse antes de impactar contra un obstáculo, e intentar implementar este comportamiento en un sistema real.

Para lograr este propósito se recopilará información sobre estas trayectorias seguidas por los seres vivos. La teoría que ha tratado de documentar y dar forma a estas trayectorias ha sido denominada: Teoría τ.

A través del estudio de esta teoría, se realizará un análisis previo para lograr conseguir un control que implemente una trayectoria seguida por el robot similar a la proporcionada por todos estos años de evolución.

Una vez realizada una recopilación de información se procederá a implementar una simulación del control con ayuda de MATLAB® y SIMULINK®.

Tras conseguir unos resultados satisfactorios en las simulaciones, se pasará al ámbito real con la construcción de un robot móvil, sobre el cual, se implementará un control similar al simulado en MATLAB®.

Para la construcción del robot, se utilizará como base de procesamiento de los datos y ejecución del control una placa **Raspberry Pi,** acompañado de sensórica que ayudará a medir distancia restante hasta el obstáculo y la velocidad del robot.

Abstract

This project will try to imitate animal behavior, in particular birds, and how they perform their detection in the desire to stop at a specific place or stop before impacting an obstacle.

To achieve this purpose, information will be gathered about these trajectories followed by living beings, the theory that has tried to document and shape these trajectories has been called: Theory τ.

Through the study of this theory, a previous analysis will be carried out to achieve a control that implements a path followed by the robot similar to that provided by all these years of evolution.

When the collection of information has been carried out, control simulation will be implemented with the help of MATLAB® and SIMULINK®.

After achieving satisfactory results in the simulations, it will be moved to the real world with the construction of a mobile robot, on which a control similar to MATLAB® simulations will be implemented.

For the construction of the robot, a **Raspberry Pi** will be used as a base for processing data and running the control, helped by sensorics that will measure the remaining distance to the obstacle and the speed of the robot.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

$\tau(t)$	Tiempo instantáneo
$\tau'(t)$	Tasa de cambio de τ respecto al tiempo
d(t)	Distancia hasta el punto de detección
a(t)	Aceleración durante la maniobra de detección
v(t)	Velocidad durante la maniobra de detección
x(t)	Posición durante la maniobra de detección
K	Constante de acoplamiento entre movimiento y guía
g	Aceleración de la gravedad (m/s^2)
sen	Función seno
tg	Función tangente
arctg	Función arco tangente
DC	Corriente continua
GPIO	General Purpose Input Output, en este caso para la Raspberry
CLK	Señal de reloj
GND	Referencia del voltaje
ADC	Convertidor Analógico a Digital
DAC	Convertidor Digital a Analógico
mAh	Miliamperios hora (capacidad de almacenamiento en baterías)
SNR	Signal-to-noise ratio
MSE	Minimum square error
Rpm	Revoluciones por minuto
LED	light-emitting diode
ToF	Time of flight

1 INTRODUCCIÓN

«No es la más fuerte de las especies la que sobrevive y tampoco la más inteligente. Sobrevive aquella que más se adapta al cambio».

- Charles Darwin -

Durante las últimas décadas, se han producido un gran número de avances tecnológicos. Concretamente en este documento nos centraremos en los relacionados con el control, cuyo inicio se remonta a tiempos muy lejanos, y a otro ámbito mucho más reciente en comparación, la robótica. Con la unión de estos dos campos, además de otros, se ha realizado la siguiente investigación, a modo de sumarse a la aplicación de estas nuevas investigaciones.

Así como no se nota un grano de arena en una playa, muchas veces no nos damos cuenta lo complejo que sería realizar la mayoría de acciones cotidianas, si nuestro cerebro no nos ayudara transformando una orden en un conjunto de señales hacia nuestros músculos. Es por esto, que suele pasar por desapercibido todos los años de evolución que lleva el planeta tierra y de todas las habilidades cognitivas desarrolladas por los seres vivos que contiene.

Basándonos en lo anterior, a la hora de buscar una forma eficaz y sencilla de comprender por nuestro cerebro, nos ha llevado a darnos cuenta en algo que lleva con nosotros desde el principio, la evolución de las especies. Si tras todos estos años de evolución se pueden ver tantos detalles en la naturaleza, sería un desperdicio no recurrir a ella cuando queremos encontrar la solución a un problema. De modo que cuando se quiso encontrar una solución para el problema de aterrizar un dron de manera eficiente y segura, se pensó en un organismo volador que lleva con nosotros mucho tiempo. Y es que las aves desarrollan esta habilidad para finalizar el vuelo con bastante facilidad, claro que a la hora de descifrar todo lo que pasa por sus mentes cada vez que lo hacen, tendremos que dejar el mundo real y verlo todo de una manera más compleja, con las matemáticas.

Sin embargo, aunque nosotros seamos capaces de realizar estas acciones con facilidad, ya se ha llegado a ver claramente durante esta etapa, de "evolución" de la robótica, que cualquier tarea tan simple como subir un escalón puede llevar un mundo para un robot. A medida que la ingeniería avanza en este ámbito, se logran avances. Estos avances, sin embargo, se logran lentamente, ya que intentar replicar todos los cálculos que hace nuestro cerebro de manera interna, como por ejemplo cálculo de momentos de inercias, están a un subnivel en el cual no accedemos a la hora de realizar la acción. De hecho, se ha podido ver cómo la empresa *Boston Dynamics* consiguió hace unos años lograr que un robot subiera un escalón con bastante dificultad, y hasta hace poco no se ha visto como esta empresa ha conseguido desarrollar robots con capacidades motoras muy equiparables a las de seres vivos, logrando hacer saltos e incluso bailar. Desde hace pocos años se ha visto que la manera óptima de resolver problemas simples para nosotros en nuestra vida diaria pero complejo para un robot como, por ejemplo, problemas de lanzamientos de objetos, tema que se podría asemejar también al tratado en este proyecto, se están realizando, implementando redes neuronales como solución al problema.

Estos desarrollos de la robótica, nos puede llevar a la conclusión de que implementar un control basado en una habilidad cognitiva de un conjunto de seres vivos, puede llegar a ser bastante complejo. De manera que, tendremos que simplificar estas habilidades hasta que se alcance un punto que cumpla con correcto funcionamiento para el fin con el que queremos usarlo, y que cumpla unas condiciones de complejidad que puedan ser manejadas. En este caso por un robot tipo dron cuyas capacidades de cálculo y procesamiento están en parte más limitadas que un robot terrestre. En resumen, trataremos de imitar esta habilidad cognitiva y no de intentar crearla o de que el robot la aprenda a base del machine learning.

1.1 Motivación del Proyecto

Con la reciente aparición de Drones ha crecido el interés de encontrar la mejor forma para el control de estos, y para no desaprovechar lo que la evolución ha podido ir perfeccionando durante años, en este Proyecto se intentará abordar una solución al problema de control basado en la respuesta que tienen las aves hacia el problema de detener el vuelo.

Diseñar un control basado en la forma de afrontar el problema por parte de las aves, a parte ayudarnos a entender de una mejor forma cómo funcionan los seres vivos, nos adentrara en el camino de investigar por qué la evolución ha considerado que esta forma de solucionar este problema es la más adecuada.

1.2 Objetivos del Proyecto

El principal objetivo de este proyecto, es comprobar la posible implementación de un mecanismo de control bio-inspirado para el aterrizaje de un dron en una superficie estable. Para ello, se partirá de un objetivo menor, obtener información de la Teoría τ. Esta teoría, estudia como las aves logran realizar una maniobra de detención satisfactoria y que movimiento desarrollan a lo largo de esta maniobra. En este estudio se parte de varias experiencias de pilotos al aprender unas habilidades similares a las que desarrollan las aves.

Una vez obtenida suficiente información sobre la maniobra seguida por los seres vivos, así como del estudio de esta teoría que la analiza, se continuara implementando un control basado en la teoría de forma experimental en MATLAB® y SIMULINK®. Dichas pruebas en software se harán con la finalidad de alcanzar el último objetivo de este proyecto, que será la implementación de un sistema robótico automatizado que pueda imitar la habilidad cognitiva de las aves de detener su movimiento, en este caso ante un obstáculo.

2 TEORÍA TAU

«De vez en cuando vale la pena salirse del camino, sumergirse en un bosque. Encontrarás cosas que nunca habías visto.»

- Alexander Graham Bell -

La teoría de tau, más conocida como: Teoría τ, es el intento de replicar los frutos que dieron años de evolución, en concreto de cómo los seres vivos se detienen ante el encuentro de un obstáculo o como inician un vuelo. Principalmente es un estudio basado en animales con la capacidad de volar, pero también se puede ver cómo los humanos han adquirido esta capacidad. Este estudio, revela la trayectoria seguida por estos animales al iniciar y detener el vuelo e intenta replicarlo en esta teoría, la Teoría τ.

A continuación, se presentarán los conceptos más importantes de la Teoría τ, los cuales he usado para el desarrollo de este proyecto y que he adquirido basándome en los artículos: [1], [2], [3], [4], [5], [6] y [7].

2.1 Conceptos básicos

Tras la realización de un estudio sobre cómo los animales se comportaban ante del deseo de llegar a una posición concreta, y con la ayuda de varios pilotos se estimó que: para detenerse de manera segura antes del obstáculo, se deben saber varias cosas antes de iniciar la maniobra.

En primer lugar, saber cuándo iniciar la desaceleración y luego regular esta durante el acercamiento. Es decir, siendo **d**, la distancia al punto de parada, **v**, la velocidad y **a**, la aceleración de la aeronave. Todas en función del tiempo, en cualquier punto durante la desaceleración, la distancia instantánea para detener puede obtenerse de manera simple:

$$d = \frac{v^2}{2a}$$

A partir de la ecuación anterior, para una velocidad y aceleración instantáneas conocidas, si la distancia resultante de la fórmula anterior es más grande que la distancia al obstáculo, el piloto tiene que aumentar la desaceleración y, por el contrario, si es más pequeña, el piloto sabe que puede reducir la desaceleración.

Por otra parte, aunque la ecuación parece simple es difícil implementarla en un control, ya que contiene elementos propensos a errores como el cuadrado y una división, además de la carga computacional que estas operaciones conllevan. En busca de hallar una forma más simple de control, se procedió, transformando la ecuación anterior en una más general, de modo que quedo de la forma:

$$x = \frac{x'^2}{2x''}$$

Ya partir de la anterior podríamos expresarla de una forma más simple si queremos ver lo necesario para evitar la colisión:

$$\frac{x * x''}{x'^2} > 0.5$$

2.2 Introducción

Centrándonos en lo explicado anteriormente, introduciremos lo que es llamado como: *Teoría τ*.

Se ha tomado como norma en la Teoría τ considerar la posición(**x**) como negativa, por lo que, el espacio (negativo), se está acercando a cero, con velocidad positiva y aceleración negativa.

Durante esta desaceleración, el tiempo instantáneo restante para la detención es llamado Tau(τ) y se puede escribir de la siguiente forma:

$$\tau = \frac{x}{x'}$$

Esta teoría recibe el nombre de Tau, debido a que, a partir de este tiempo se realizan los demás cálculos para que la detección del sistema se realice exitosamente, y de la manera más perfecta posible. Sin embargo, controlar un sistema con τ es complicado al contener una división en la cual el denominador deseamos que llegue a 0, y además existen muchos valores diferentes para la posición y la velocidad que nos darían τ iguales. Es por esto que necesitamos desarrollar un sistema de control más complejo como se verá posteriormente.

Sabiendo que esta Tau es crucial para la correcta parada del sistema antes del impacto, se descubre que la derivada de la misma (τ') es también útil para ello. Su fórmula es la siguiente:

$$\tau' = 1 - \frac{x * x''}{x'^2} \qquad or \quad \tau' + 1 = \frac{x * x''}{x'^2}$$

La hipótesis fundamental de la Teoría τ es que el sistema recoge τ y τ' como las entradas básicas para el sistema de control. Manteniendo τ' constante durante la desaceleración se puede interpretar como mantener un τ, o lo que es lo mismo un tiempo constante en la detección del sistema, lo que es llamado gap closure o intervalo de detección. Debido a esta relación entre tau y su derivada, podemos hablar de una constante de acoplamiento, que dependiendo de su valor el sistema seguirá una trayectoria u otra en su detección:

$$\tau = K * \tau'$$

La constante de acoplamiento: K = 1 – τ' puede estar entre 0 y 1 y dependiendo de su valor, el sistema actuara de la siguiente forma:

k = 0; τ' = 1; x'' = 0: El helicóptero se aproximará y colisionará con el obstáculo con velocidad constante.

k = 0.5; τ' = 0.5: El helicóptero se aproximará y se detendrá con deceleración constante

k = 1; τ' = 0: El helicóptero se aproximará exponencialmente al obstáculo, y en teoría nunca llegará.

De esta manera τ' pasara a estar implícita en K una constante que elegiremos nosotros de forma arbitraria según necesitemos para nuestro sistema y situación.

En resumen, τ' nos indicaría como de rápido se detiene el sistema indicando como varia el tiempo de parada(τ) y como no nos daría más información que la anterior y es un parámetro deseable para el diseño seria sustituida por esta constante que elegiremos según sea conveniente.

También destacar, que K = 1 - τ' por simple convenio para que k tenga los valores contrarios a τ' e ir de menor desaceleración a mayor desaceleración.

2.3 Movimientos acoplados

Con el desarrollo anterior, alcanzamos el concepto de acoplamientos de movimientos, concretamente en desplazamiento y velocidad. Sin embargo, el desarrollo de τ ha sido basado en acciones en las que es necesario unir dos movimientos para logar un resultado satisfactorio. En muchas ocasiones, la acción interceptiva que realiza un animal no está seguida de movimientos extrínsecos, sino que depende de esquemas de movimientos ya aprendidos que se correspondes con el resultado que se desea obtener. Estas "guías mentales", han sido "programadas" en el cerebro, al adquirir una nueva habilidad psicomotora como por ejemplo aprender a volar en caso de algunos animales o manejar un helicóptero en caso humano. De esta manera, estas guías tienen que ser biológicamente simples para poder adaptarse a una gran variedad de seres vivos, que es lo que ocurre con la Teoría τ. Esta teoría, propone el seguimiento del siguiente movimiento como resolución al problema, y a continuación veremos cómo desarrollar este movimiento, así como la evolución de la teoría misma.

Para echar un vistazo más de cerca a la Teoría τ, como ya se dijo anteriormente, la mayoría de movimientos naturales están basados en dos movimientos acoplados de la forma:

$\tau_x = K * \tau_y$, siendo $\mathbf{x}$ e $\mathbf{y}$ dos variables acopladas según tau, también se pueden mostrar cómo: $\boldsymbol{x = C * y^{\frac{1}{K}}}$. Donde C es una constante arbitraria y $0 \leq K \leq 1$ es una relación de leyes de potencia entre la dilatación o expansión del espacio y que se ajusta a varias relaciones sensoriales naturales, como ya se habló antes de ella. Los movimientos de los que parte la explicación que viene a continuación, se basan en movimientos dependientes o relacionados directamente con la gravedad. Los movimientos, pueden ser una caída libre en el caso de un movimiento $\mathbf{y}$ acelerado (aceleración de la gravedad), o a un aterrizaje en el caso de $\mathbf{y}$ desacelerado. De este modo, y considerando la variable $\mathbf{y}$ como la altura o eje vertical y la $\mathbf{x}$ como el eje horizontal, se suele sustituir la llamada τ_{xy} como τ_g debido a que el segundo movimiento ($\mathbf{y}$) está directamente relacionado con la gravedad.

Dado que la gravedad es un factor de influencia importante en el movimiento del animal, τ_g (t) es una función basada en la ecología de los animales. τ_g (t) se puede derivar de la ecuación dinámica del movimiento de caída libre bajo la aceleración de la gravedad **g**:

$$x_g(t) = \frac{1}{2} * g * t_d^2 - \frac{1}{2} * g * t^2$$

$$x'_g(t) = -g * t$$

$$\tau_g(t) = \frac{x_g(t)}{x'_g(t)} = \frac{1}{2} * \left(t - \frac{t_d^2}{t} \right)$$

A continuación, se muestras las fórmulas de la teoría τ para dos variables acopladas **x** e **y** designadas como **g**. A la izquierda veremos las ecuaciones para movimiento desacelerado y a la izquierda para movimiento acelerado.

$$a_g = a_{g0}$$
$$v_g = v_{g0} + a_{g0}\, t$$
$$x_g = x_{g0} + v_{g0}\, t + \frac{a_{g0}}{2} t^2$$
$$v_g(T) = 0$$
$$x_g = x_{g0}(1-\hat{t})^2$$
$$v_g = -\frac{2x_{g0}}{T}(1-\hat{t}) \quad v_{g0} = -\frac{2x_{g0}}{T}$$
$$a_{g0} = \frac{2x_{g0}}{T^2}$$
$$\tau_g = -\frac{1}{2}(T-t) = \tau_{g0} + \frac{t}{2}$$
$$\hat{\tau}_g = -\frac{1}{2}(1-\hat{t})$$
$$\tau'_g = \hat{\tau}_\gamma = 1$$

$$a_g = a_{g0}$$
$$v_g = v_{g0} + a_{g0}\, t$$
$$x_g = x_{g0} + v_{g0}\, t + \frac{a_{g0}}{2} t^2$$
$$v_{g0} = 0$$
$$x_g = x_{g0}(1 - \hat{t}^2)$$
$$v_g = -\frac{2x_{g0}}{T}\,\hat{t}$$
$$a_g = a_{g0} = -\frac{2x_{g0}}{T^2}$$
$$\tau_g = -\frac{T}{2}\left(\frac{1}{\hat{t}} - \hat{t}\right)$$
$$\hat{\tau}_g = -\frac{1}{2}\left(\frac{1}{\hat{t}} - \hat{t}\right)$$
$$\tau'_g = \hat{\tau}_g = \frac{1}{2}\left(1 + \frac{1}{\hat{t}^2}\right)$$

A través de estas ecuaciones, se puede describir por completo la trayectoria seguida en el caso anterior por los pilotos que han ayudado a crear la teoría, pero en definitiva de las aves, a las que pretendemos imitar. Luego, una vez conocidas estas ecuaciones podremos construir la trayectoria en cualquier punto.

2.4 Construcción de la Trayectoria τ

A partir de estas ecuaciones de movimientos acoplados se podría desacoplar la componente **x** a partir de la τ_g,y esta seguiría lo llamado trayectoria τ. Debido a que la componente **y**, sería un movimiento constantemente acelerado o desacelerado, se podría tratar de forma aparte. Además, este intento de replicar una trayectoria según la Teoría τ, está orientada a un sistema volador, en el caso de este trabajo los resultados están orientados a la construcción de un robot terrestre, aunque como para su futura continuación sería interesante la implementación a un robot volador, es útil intentar replicar este movimiento.

$$\hat{x} = -(1-\hat{t})^{2/k}$$

$$\hat{x}' = \frac{2}{k}(1-\hat{t})^{(\frac{2}{k}-1)}$$

$$\hat{x}'' = -\frac{2}{k}(\frac{2}{k}-1)(1-\hat{t})^{2(\frac{1}{k}-1)}$$

$$\hat{\tau}_{\hat{x}} = -\frac{k}{2}(1-\hat{t})$$

$$\hat{\tau}'_{\hat{x}} = \dot{\tau}_x = \frac{k}{2}$$

$$\hat{x}''(0) = -\frac{2}{k}(\frac{2}{k}-1)$$

$$T = -\frac{2\tau_{x0}}{k}$$

$$\hat{x} = -(1-\hat{t}^2)^{1/k}$$

$$\hat{x}' = \frac{2\hat{t}}{k}(1-\hat{t}^2)^{(\frac{1}{k}-1)}$$

$$\hat{x}'' = -\frac{2}{k}\left(\left(\frac{2}{k}-1\right)\hat{t}^2 - 1\right)(1-\hat{t}^2)^{(\frac{1}{k}-2)}$$

$$\hat{\tau}_{\hat{x}} = \frac{k}{2}\left(\hat{t} - \left(\frac{1}{\hat{t}}\right)\right)$$

$$\hat{\tau}_{\hat{x}} = \frac{k}{2}\left(1 + \left(\frac{1}{\hat{t}}\right)^2\right)$$

$$\hat{x}''(0) = \frac{2}{k}$$

$$T = -\sqrt{\frac{2x(0)}{k\,\ddot{x}(0)}}$$

Conforme a la temática de este proyecto, nuestro interés se centrará en las fórmulas de la izquierda correspondientes al movimiento desacelerado. Estas fórmulas, serán implementadas posteriormente en la parte práctica de este proyecto, de manera que generarán la trayectoria deseada a seguir.

2.5 La Teoría en una vista más real

Las siguientes figuras ilustran las variaciones de los movimientos y τ para varios valores de la constante de acoplamiento k.

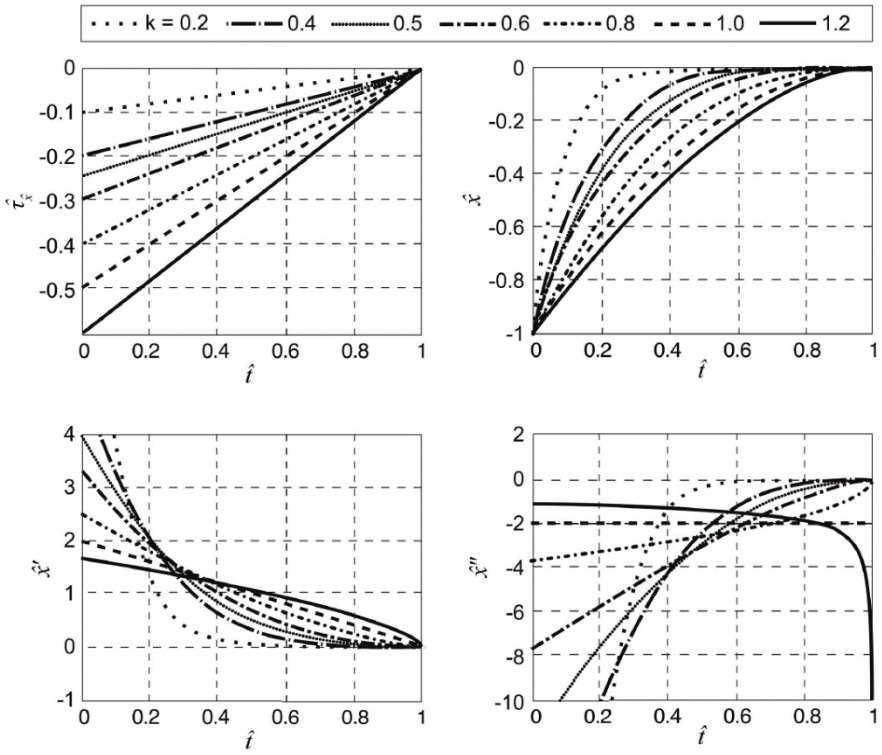


Figura 2:1 Representación de Tau, posición, velocidad y aceleración, durante el tiempo de maniobra para las diferentes K

El tiempo y las distancias se han normalizado por el tiempo de maniobra y la distancia, T y **x**. Algunas observaciones directas de estas relaciones pueden ser útiles en este punto.

Las trayectorias anteriores mostradas en la Figura 2:1, nos enseñan como varía la posición del sistema para conseguir la detección correcta del sistema, y a partir de ellas podemos ver con mayor como se comportaría el sistema esta teoría, aun así, y de una forma más concreta, repasaremos las características de estas trayectorias para el movimiento desacelerado:

- Los movimientos comienzan con una desaceleración brusca, que disminuye gradualmente a medida que se termina el movimiento. Cuanto mayor sea el acoplamiento k, más se acerca el perfil al caso límite de desaceleración constante (k = 1). En la práctica, la desaceleración inicial tardaría un tiempo en ser alcanzada, de la misma forma la τ tardara un tiempo en alcanzar el valor constante deseado.
- En teoría valores para K>1 son posibles, pero significan una desaceleración al final, lo que significaría una colisión en la práctica, al no poder suplir la desaceleración pedida en la mayoría de casos.
- En el momento en que se inicia la desaceleración (t = 0), el tiempo total requerido para detener el sistema es de: $T = -2 * \tau_x(0)/K$. Por ejemplo, si k = 1, entonces el tiempo de maniobra será el doble del τ percibido en t = 0.

Para el caso acelerado existen otras características a tener en cuenta, pero para el caso de desaceleración, el cual vamos a tratar es innecesario su conocimiento.

En las prácticas, esperaríamos un seguimiento cercano a esta guía dada por la teoría τ, tanto en aceleración como en desaceleración, con las ecuaciones vistas anteriormente. Sin embargo, esta teoría no permitiría que el objeto se moviera, o en su defecto, una gran perturbación hacia el vuelo del animal o la aeronave que consiga desviarlo de la trayectoria, es por esto que se aplicara su uso como un generador de trayectorias y un posterior control robusto que corrija posibles errores o perturbaciones para el seguimiento de la trayectoria τ de la mejor manera posible. Cabe destacar, que la base del control será la teoría τ al proyectar una trayectoria bio-inspirada, que posiblemente sea, la que mejor se adapte tanto a la seguridad del modelo como a la simplicidad a la hora de ejecutar el control. Por otra parte, estas correcciones pueden, en principio, hacerse mediante ajustes en k o incluso en el tiempo de maniobra T, aunque, como se observa con los movimientos que sigue la trayectoria τ de desaceleración constante, la propia T se puede predecir a través del τ instantáneo, lo que limitaría el posible ajuste de este parámetro. Por ejemplo, si es necesario una fase de velocidad constante, debido a grandes turbulencias o porque es requerido, se podría dejar aparte la trayectoria τ y mantener $\tau = 1$ durante ese periodo de tiempo y después continuar con la trayectoria.

3 METODOLOGÍA DEL CONTROL Y SIMULACIONES

«Donde los sentidos nos fallan, la razón debe intervenir.».

- Galileo -

Una vez realizada la recopilación de información sobre cómo se podría simular la trayectoria seguida por las aves, que como se ha visto en el capítulo anterior, está recogido en la Teoría τ, se va a proceder a intentar replicar el movimiento a través de algunas simulaciones en MATLAB® y concretamente utilizando SIMULINK®.

Como ya se ha podido ver en el apartado anterior, la Teoría τ, nos indica la trayectoria a seguir si queremos realizar este movimiento bio-inspirado, por lo que a la hora de afrontar el problema lo haremos partiendo de un seguidor de trayectoria *(Tau-guidance)* acompañado de un controlador que obligue al sistema a realizar el seguimiento de la trayectoria deseada.

También, destacar que estas simulaciones estarán enfocadas para la construcción del sistema robot móvil, por lo que se intentará simular un sistema terrestre más que uno volador. Sin embargo, no conocemos el modelo con certeza, por lo que para simular el modelo se usara una función de segundo orden intentando imitar a un motor, bastaría una de primer orden, pero queremos orientar el control de manera que sea lo más robusto posible.

3.1 Diseño del entorno de simulación

A la hora de enfocar el control, se usará un seguidor de trayectoria junto con un control tipo PID. Una de las principales partes del entorno de simulación, será el generador de trayectoria *(Tau-guidance)*. Este bloque contendrá una función en Matlab en la que se generarán las trayectorias bio-inspiradas, ya definidas en la Teoría τ. Las entradas del mismo será la posición actual del Sistema y el tiempo, este último es solo para saber cuándo debe comenzar a actuar. Las salidas serán la posición velocidad y aceleración deseadas para que se siga el movimiento bio-inspirado en cada momento. También destacar, que, para el desarrollo correcto de la simulación, además de lo anterior, se le pasaran un par de valores que le indican la distancia a la que se desea que se detenga y la K para determinar el movimiento.

3.1.1 Bloques que forman el entorno de simulación

A continuación, se hará un resumen de los bloques usados en Simulink. En la Figura 3:1, podemos encontrar una visualización de todos los bloques de simulación. Estos se pueden dividir en varios bloques, el primero lo podemos encontrar en la Figura 3:2 aumentado, este bloque es el que contiene el código que se comentará a continuación y que generará los valores deseados para que el robot los siga en todo momento. Este bloque está colocado en la esquina inferior izquierda. La parte central de la simulación es el modelo y su control PID, que se pueden visualizar de una mejor forma en la Figura 3:3. Por último, tenemos el bloque que se usará para proporcionar al sistema los datos necesarios en cada momento, este bloque recibe el nombre de switch y se puede encontrar en la Figura 3:4. Este bloque recibe los datos tanto de tiempo actual de la simulación y tiempo en el que se inicia el control, para saber cuándo cambiar los datos que le envía al control. El switch antes de que se cumpla la condición de tiempo le envía al control una velocidad de referencia indicada por el usuario, para colocar el robot en unas condiciones iniciales, en este caso de velocidad, y cuando la simulación empieza cambia, para comenzar a enviar los datos que genera el bloque Tau_Guidance. Una vez han sido definidos los bloques pasaremos a explicar el conjunto de la simulación.

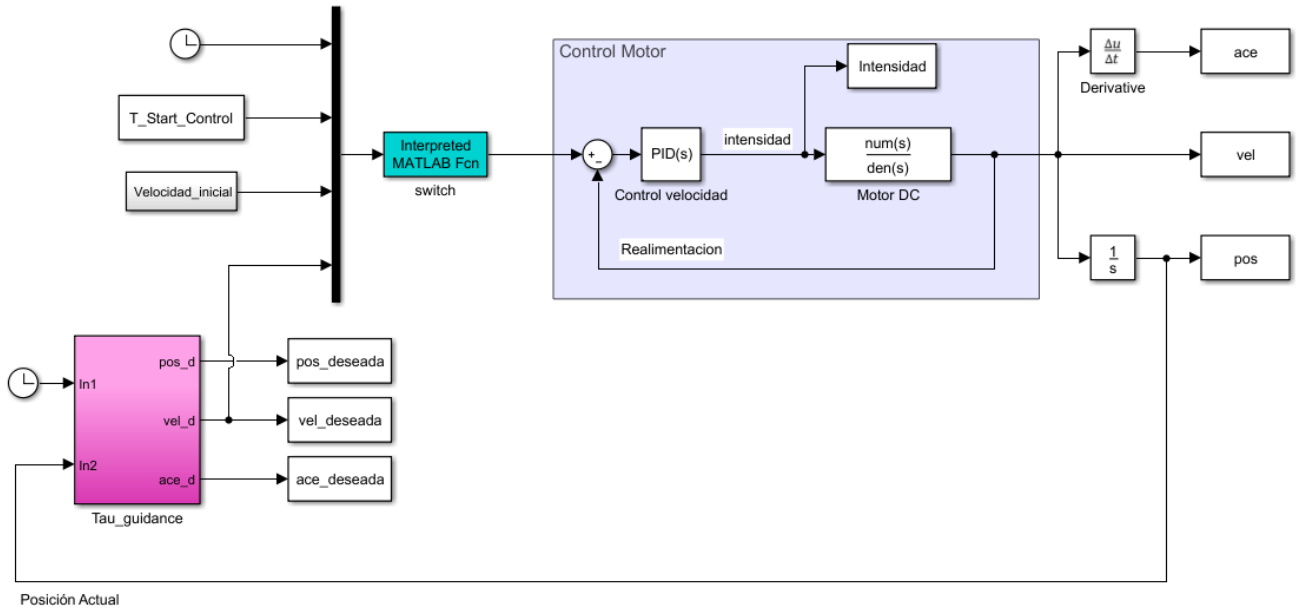


Figura 3:1 Entorno de simulación

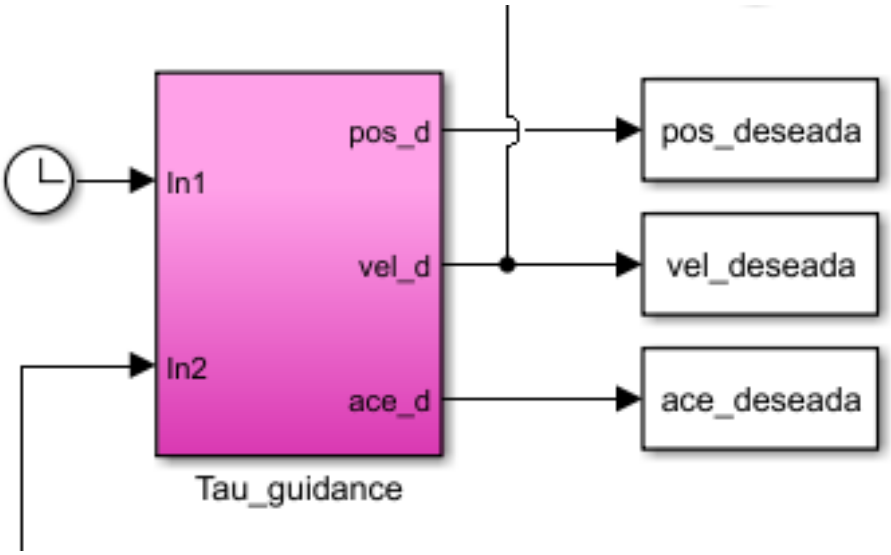


Figura 3:2 Bloque Generador de trayectoria

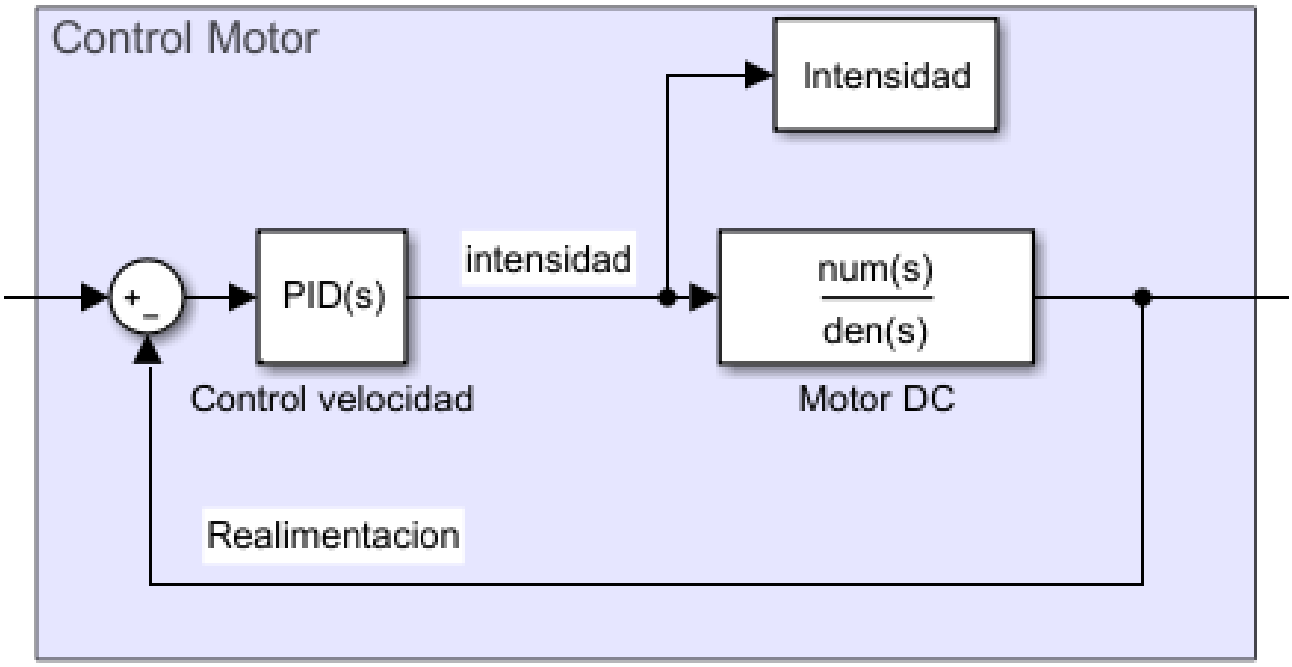


Figura 3:3 Bloque Control del Sistema

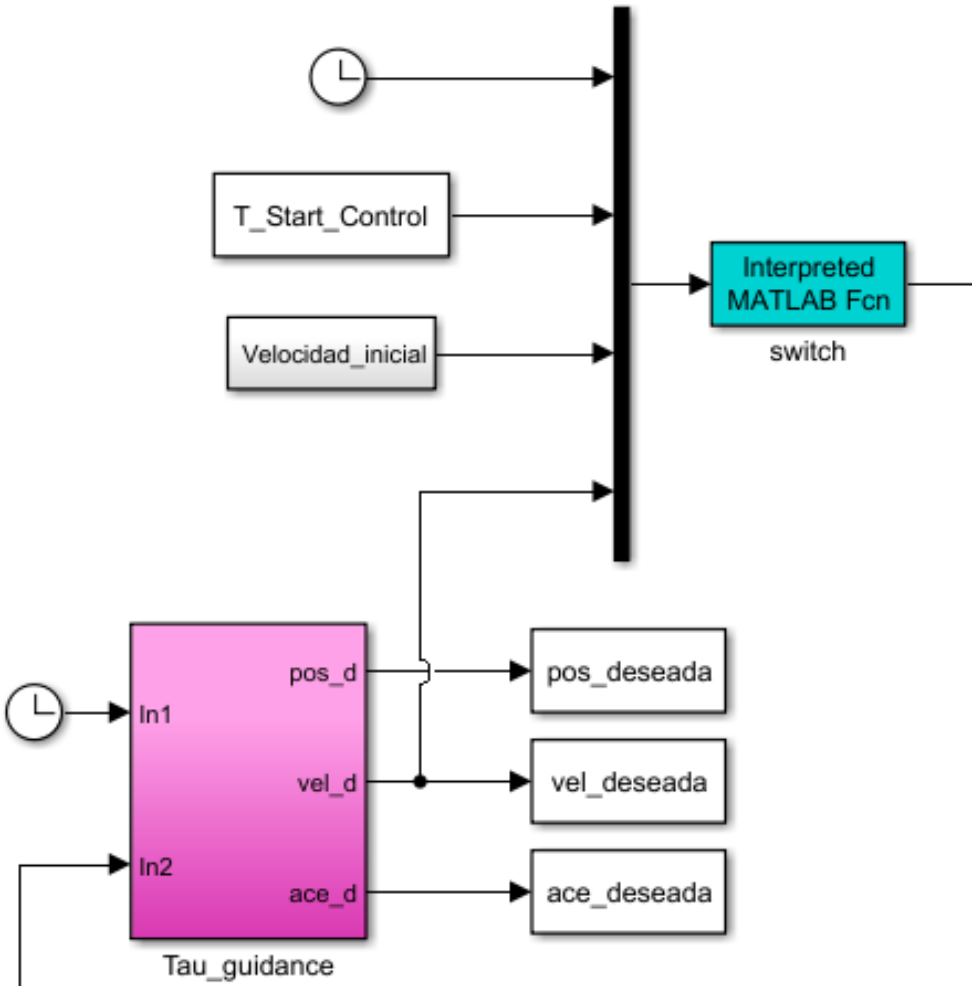


Figura 3:4 Bloque Switch y su entrada

3.1.2 Generador de la trayectoria

A continuación, se comentará de una forma más detallada el código incluido en el bloque que genera la trayectoria tau y que se ha fundamentado en las fórmulas vistas en el apartado anterior.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%Definición de variables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%variables con memoria
persistent tiempo_ant
persistent pos_ant %valor inicial 0
persistent vel_ant %valor inicial 0
persistent ace_ant %valor inicial 0
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Globales para poder observarlas al finalizar
global Tiempo_maniobra
global pos_inicial_control
global vel_inicial_control
global ace_inicial_control

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Datos necesarios para el generador
global Start_control %Instante en el que empieza
global K %Constante para Tau
global D_objeto %Distancia a la que se encuentra el objeto

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Entradas del módulo
tiempo=inp(1);
pos=inp(2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%Valores iniciales
vel=0;
ace=0;
%%%
tau_nominal=0;
pos_nominal=0;
vel_nominal=0;
ace_nominal=0;
%%%
pos_deseada=0;
vel_deseada=0;
ace_deseada=0;
tau_deseada=0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%Inicio del código
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if(tiempo<1e-4) %Cuando el tiempo es 0
    tiempo_ant=tiempo;
    pos_ant=0;
    vel_ant=0;
    ace_ant=0;
elseif(tiempo>(Start_control+Tiempo_maniobra))
        %Fin del control
        %El tiempo actual es mayor al del comienzo más el tiempo que
        %dura la maniobra

        %Una vez acabada la maniobra se fijan los
        %valores de referencia finales
        pos_deseada=pos_inicial_control+D_objeto;
        vel_deseada=0;
        ace_deseada=0;
        %debería acabar con valores próximos a 0
        %depende del valor de la K usada
```

```
else
    %%%%%%%%%%%%%%%%%%%%%%%%%
    %Generador de trayectoria
    %%%%%%%%%%%%%%%%%%%%%%%%%

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Calculo de datos necesarios
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Tiempo transcurrido desde la iteración anterior (incremento)
    inc_tiempo=tiempo-tiempo_ant;
    %velocidad y aceleración actuales
    %A partir de la entrada de la posición
    vel=(pos-pos_ant)/inc_tiempo;
    ace=(vel-vel_ant)/inc_tiempo;

    if(tiempo<=(Start_control))
    %Se espera hasta que se alcance el tiempo deseado
    %El control empezaría en la siguiente iteración
    %En la primera se calculan valores necesarios para su desarrollo

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%En primer lugar necesitamos la tau inicial
        %tau= distancia/velocidad %en T=0 distancia total
        tau_zero=-D_objeto/vel;
        %tau es negativa según la referencia escogida

        %T es (2*tau_zero) /K siendo K una constante elegida
        %El tiempo de maniobra será el tiempo total en desarrollarse
        %la aproximación
        Tiempo_maniobra=-(2*tau_zero)/K;
        %el signo - es para que sea positivo al elegir tau negativa

        %%%%%%%%%%%%%%%%%%%%%%%
        %valores inicial de la posición, velocidad y aceleración
        pos_inicial_control=pos;
        vel_inicial_control=vel;
        ace_inicial_control=ace;
        %%%%%%%%
        %como el control no se inició aun, pero el módulo necesita
        %valores de salida se escogen como referencia los
        %valores actuales, en este caso mantenemos la velocidad
        pos_deseada=0;
        vel_deseada=vel;
        ace_deseada=0;

    else
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %Ejecución del algoritmo que genera la trayectoria
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %Las variables se calculan normalizadas
        %Posteriormente se calculan las reales según cada caso
        %%%%%%%%%%%%%%%%
```

```
        %Porcentaje del tiempo de maniobra trascurrido:
        Tm=(tiempo-Start_control)/Tiempo_maniobra; %menor de 1

        %Calculo valores normalizados
        %los signos - dependen de la referencia escogida
        pos_nominal=-(1-Tm)^(2/K);
        vel_nominal=(2/K)*(1-Tm)^((2/K)-1);
        ace_nominal=-(2/K)*((2/K)-1)*(1-Tm)^(2*((1/K)-1));
        %en definitiva es derivar y se podría optimizar mucho más el
        %calculo pero así queda más claro, se podrían optimizar

        %tau y derivada de tau
        %No es necesario su calculo
        tau_nominal=-(K/2)*(1-Tm);
        tau_nominal_1=K/2;

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %A continuación se pasará a los valores deseados
        %para las diferentes salidas
        %(multiplicar para quitar el nominal)
        pos_deseada=pos_inicial_control+pos_nominal*D_objeto+D_objeto
        %Se le suma D_objet porque va de -1 a 0

        vel_deseada=vel_nominal*(D_objeto/Tiempo_maniobra);
        ace_deseada=ace_nominal*(D_objeto/(Tiempo_maniobra^2));
        tau_deseada=tau_nominal*D_objeto;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Guardando los valores para la siguiente ejecución
    tiempo_ant=tiempo;
    pos_ant=pos;
    vel_ant=vel;
    ace_ant=ace;

end

%salida
out(1)=tau_nominal;
out(2)=pos_nominal;
out(3)=vel_nominal;
out(4)=ace_nominal;
out(5)=pos_deseada;
out(6)=vel_deseada;
out(7)=ace_deseada;
out(8)=tau_deseada;
%En este caso se colocan como salida todos los parámetros calculados,
%pero bastaría con sacar solo la velocidad deseada
```

3.1.3 Modelo del motor y PID

Por otro lado, en la simulación tenemos un control tipo PID junto con un modelo de motor, simulando lo que sería el sistema móvil. También destacar que para estas pruebas se ha optado controlar la velocidad para simplificar las simulaciones, pero se podría hacer perfectamente con la posición y en la parte final del proyecto es con ella con la que se trabajará.

A la hora de elegir la base para un modelo de motor DC se ha usado como base la información de la página [8]. A partir de esta página web, se han podido sacar las diferentes ecuaciones que rigen este sistema y así poder sacar un modelo bastante acertado.

Dinámica del sistema:

$$\sum T_m = J_m \frac{d\theta_{m(t)}^{\ 2}}{d^2t}$$

$$J_m = J_a + (\frac{N_1}{N_2})^2 J_L, \quad D_m = D_a + (\frac{N_1}{N_2})^2 D_L$$

$$t_{m(t)} = \left(J_a + (\frac{N_1}{N_2})^2 J_L\right)\frac{d\theta_{m(t)}^{\ 2}}{d^2t} + (D_a + (\frac{N_1}{N_2})^2 D_L)\frac{d\theta_{m(t)}}{dt} \quad (1)$$

$$e_{a(t)} = R_a i_{a(t)} + e_{m(t)} \quad (2)$$

$$e_{m(t)} = k_b \frac{d\theta_{m(t)}}{dt} \quad (3), \quad t_{m(t)} = k_i i_{a(t)} \quad (4), \quad \theta_{L(t)} = (\frac{N_1}{N_2})\theta_{m(t)} \quad (5)$$

Transformada de Laplace:

$$T_{m(s)} = (\left(J_a + \left(\frac{N_1}{N_2}\right)^2 J_L\right)s + D_a + \left(\frac{N_1}{N_2}\right)^2 D_L)s\theta_{m(s)}$$

$$E_{a(s)} = R_a I_{a(s)} + E_{m(s)}$$

$$E_{m(s)} = k_b s\theta_{m(s)}$$

$$T_{m(s)} = k_i I_{a(s)}$$

$$\theta_{L(s)} = (\frac{N_1}{N_2})\theta_{m(s)}$$

Función de transferencia:

$$I_{a(s)} = \frac{T_{m(s)}}{k_i}$$

$$E_{a(s)} = R_a I_{a(s)} + E_{m(s)} = R_a I_{a(s)} + k_b s\theta_{m(s)} = R_a(\frac{T_{m(s)}}{k_i}) + k_b s\theta_{m(s)}$$

Y como resultado, obtenemos una función de transferencia con al que despejar con los datos de nuestro motor:

$$\frac{\theta_{L(s)}}{E_{a(s)}} = \frac{(\frac{N_1}{N_2})(\frac{k_i}{R_a})}{s((J_a + (\frac{N_1}{N_2})^2 J_L)s + D_a + (\frac{N_1}{N_2})^2 D_L + \frac{k_i k_b}{R_a})}$$

Como para las simulaciones aún no se sabía el modelo a usar se decidió a usar un modelo estándar, en concreto uno de los proporcionados por esta página web. Este modelo es de segundo orden.

$$\frac{\mathbf{0.0417}}{\boldsymbol{s(s+1.1667)}}$$

Una vez con el modelo del motor elegido y con ayuda de la herramienta de Matlab PIDTuner, calcularíamos los parámetros del PID que más se ajustaran a nuestros requisitos. De esta manera se elegio un PID más agresivo de lo normal para que se pudieran suplir de manera efectiva los grandes cambios de aceleración en los momentos iniciales del movimiento. Estos datos se pueden encontrar en la Tabla 3:1.

Siendo el PID de la forma:

$$P\left(1 + I\frac{1}{s} + D\frac{N}{1 + N\frac{1}{s}}\right)$$

Tabla 3:1 Características del PID.

P	1.16
I	0.26
D	0.49
N	28
Rise time	0.5 segundos
Closed-loop stability	stable

3.2 Simulaciones Generador de trayectoria

A fin de intentar replicar el movimiento seguido por la trayectoria de la Teoría τ, se comenzará imitando el movimiento normalizado ya comentado en el punto 2.5. Para esto se utilizará el entorno de simulación ya explicado y cuyos resultados nos darían las siguientes gráficas:

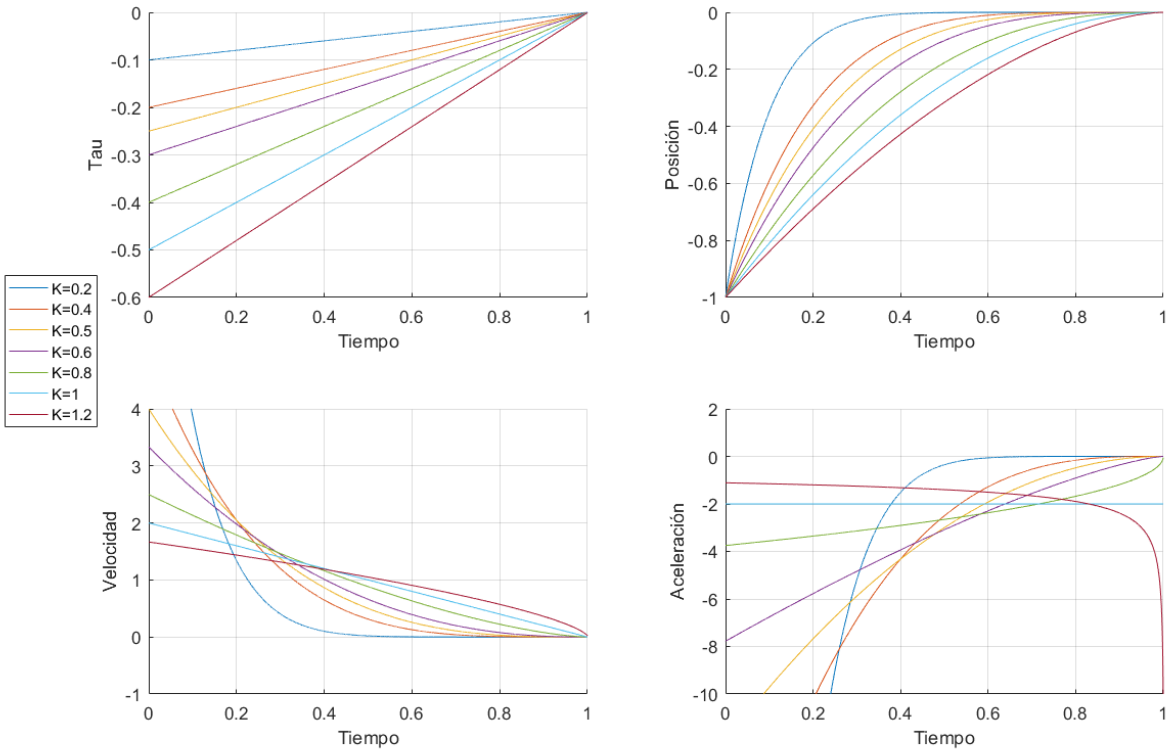


Figura 3:5 Representación de Tau, posición, velocidad y aceleración, durante el tiempo de maniobra para las diferentes K. Simulación MATLAB

Como se puede ver en la Figura 3:5, los resultados de la simulación, son iguales a los de la Teoría τ comentada anteriormente. Cabe destacar que el tiempo esta normalizado como ya se vio en la gráfica sacada de los artículos.

Para su mejor visualización se dejan a continuación cada grafica por separado:

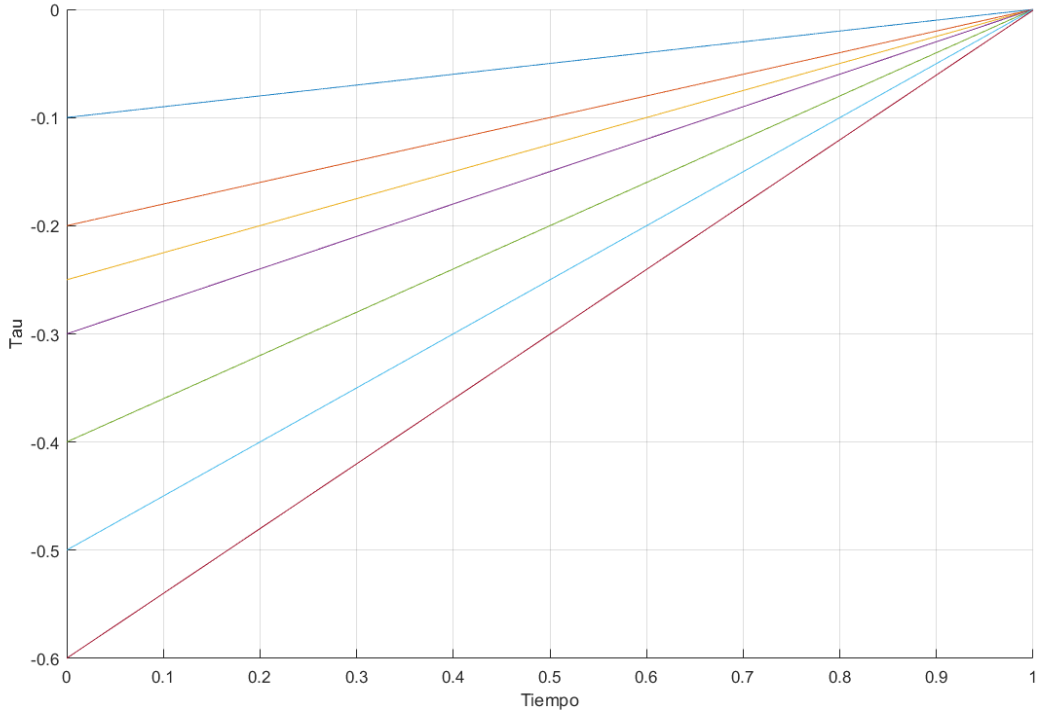


Figura 3:6 Representación de Tau según K

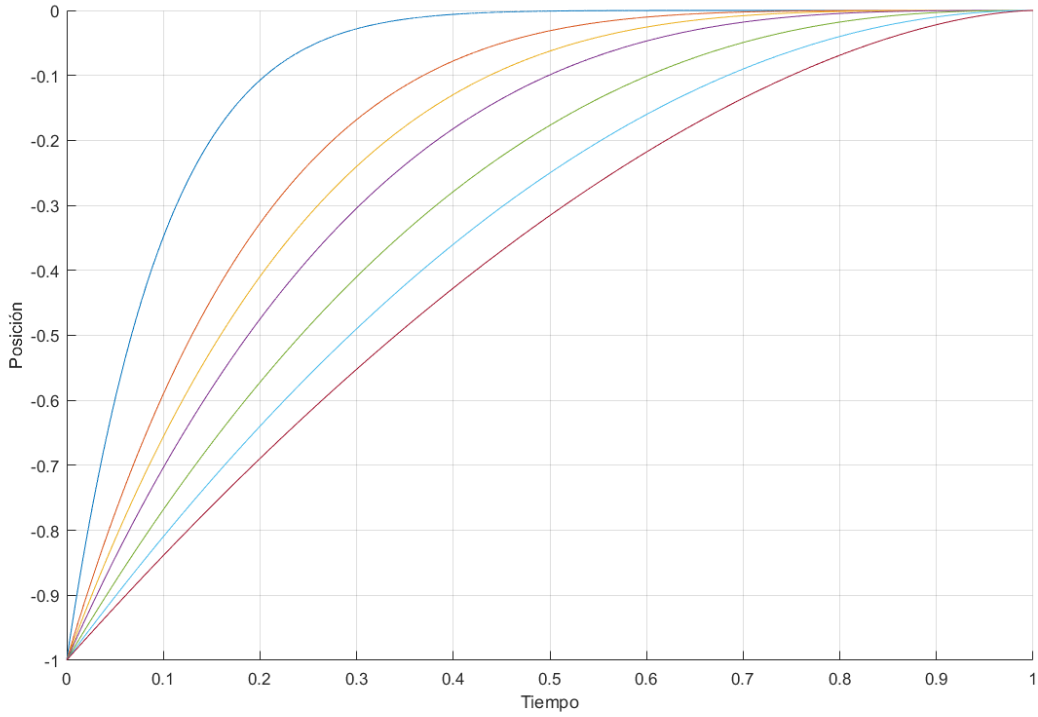


Figura 3:7 Representación de la posición según K

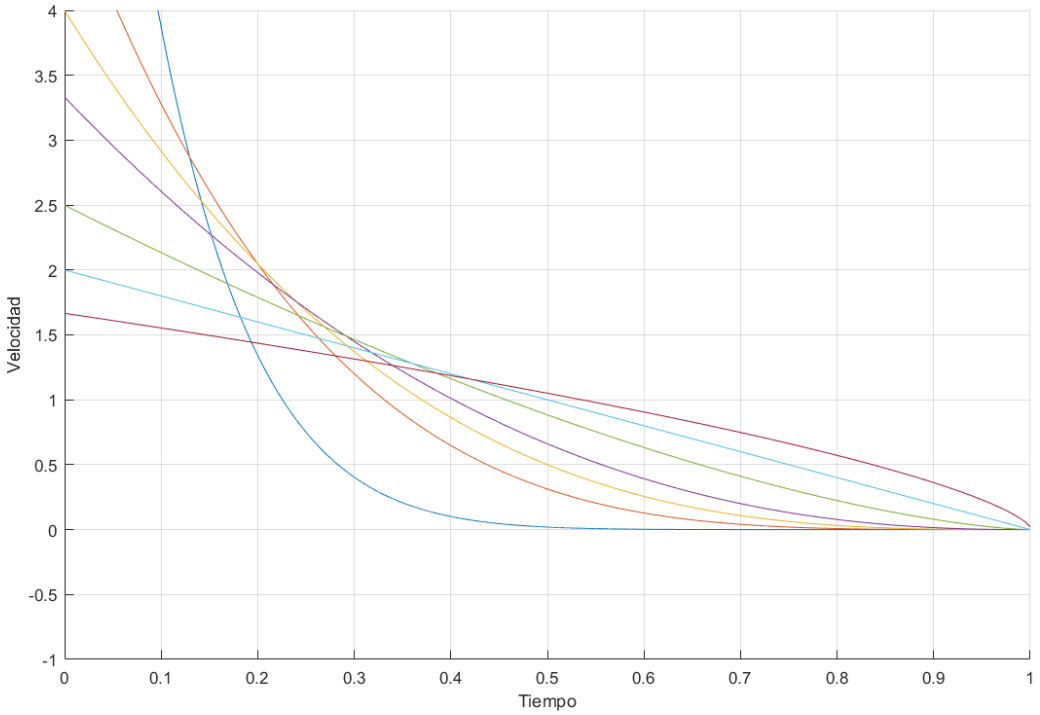


Figura 3:8 Representación de la velocidad según K

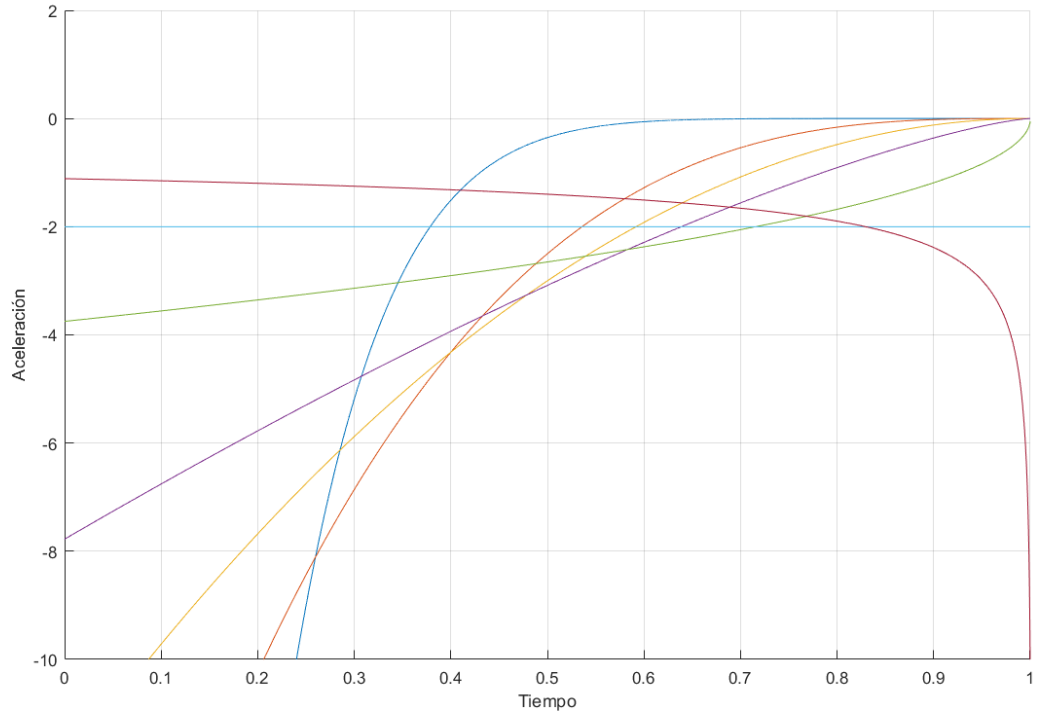


Figura 3:9 Representación de la aceleración según K

Aunque las gráficas vistas en la Figura 3:5 directamente parezca que no aportan mucho, nos indican que posición velocidad y aceleración deberá tener el sistema en cada momento, por lo que empezaremos a simular el comportamiento de nuestro sistema al intentar obligarle a seguir una trayectoria. Al ser un sistema irreal nos permitiría introducir casi cualquier trayectoria, pero para enfocar el sistema a una K real empezaremos con una K=0.5, que es la de color naranja en las gráficas anteriores. Se ha elegido esta K porque en el estudio las K entre 0.4 y 0.6 son las que se comportan mejor en sistemas reales, y aunque el nuestro aun no lo sea el siguiente paso será hacer simulaciones en tiempos discretos y ver cómo podemos modificar la K según lo necesitemos. Una vez aclarado esto y simulando para una K de 0.5, una velocidad inicial de 5 unidades de distancia por tiempo y una distancia al objeto de 10 unidades de distancia los resultados obtenidos son bastante prometedores.

Como se puede ver a simple vista en la Figura 3:10, el control usado para el modelo cumple bastante bien su función luego para estas simulaciones bastará. También destacar que el seguimiento tanto de posición y aceleración es muy acertado pese a controlar solo en aceleración, ya que el sistema es bastante simple.

Para ver los errores de cerca, aumentaremos las gráficas un poco. Sin embargo, para este caso no hace falta aumentar mucho las gráficas ya que los posibles errores son bastante pequeños. En la Figura 3:11, se puede ver que el error de posición es bastante pequeño. En el caso de la aceleración, vista en la Figura 3:13. El error es al comienzo al pasar de una referencia de 0 a una de -2 pero se adapta en menos de 1 segundo.

(En las gráficas se pueden ver dos líneas discontinuas de color magenta, estas indican el inicio y el final del Gap o dicho de otra forma el tiempo en el que el control bio-inspirado está en funcionamiento.)

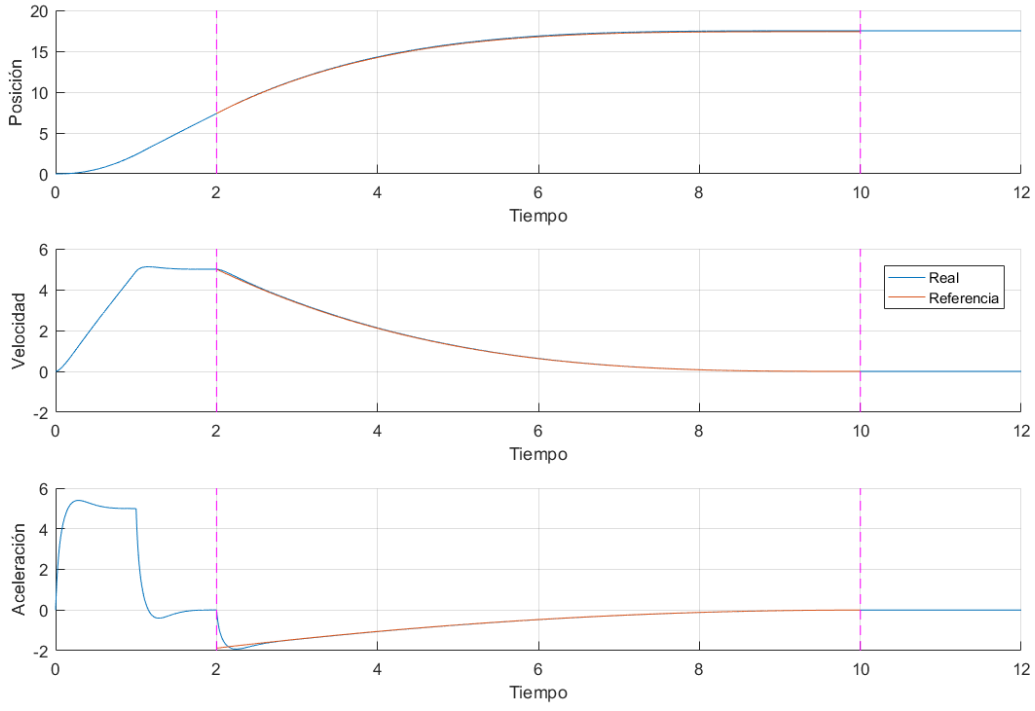


Figura 3:10 Simulación k=0.5

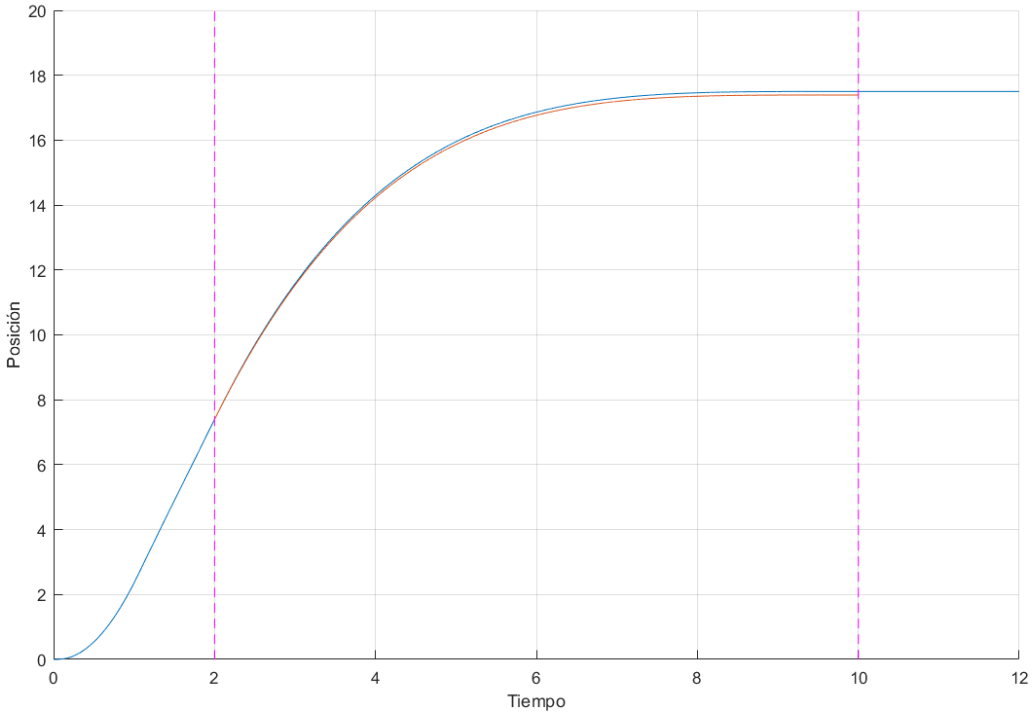


Figura 3:11 Posición para la simulación de K=0.5

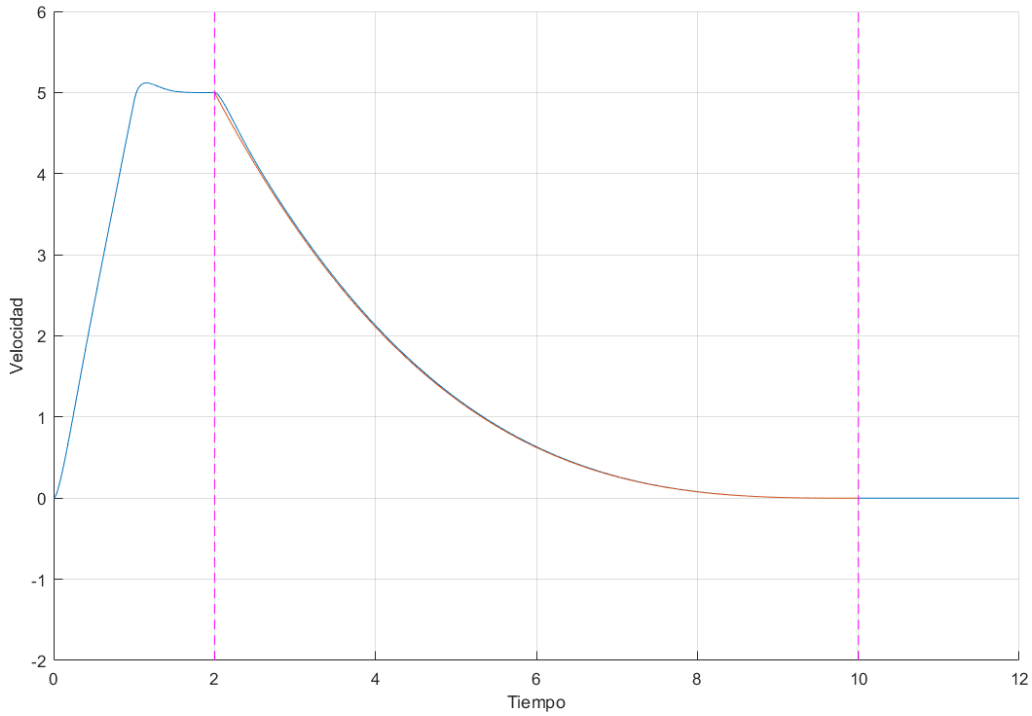


Figura 3:12 Velocidad para la simulación de K=0.5

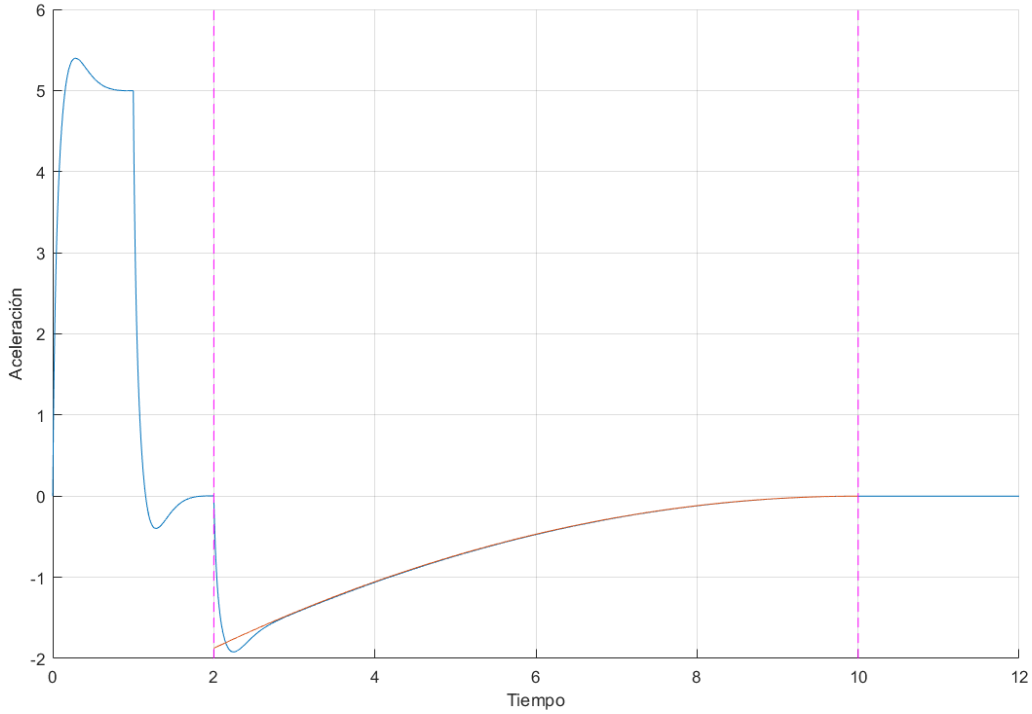


Figura 3:13 Aceleración para la simulación de K=0.5

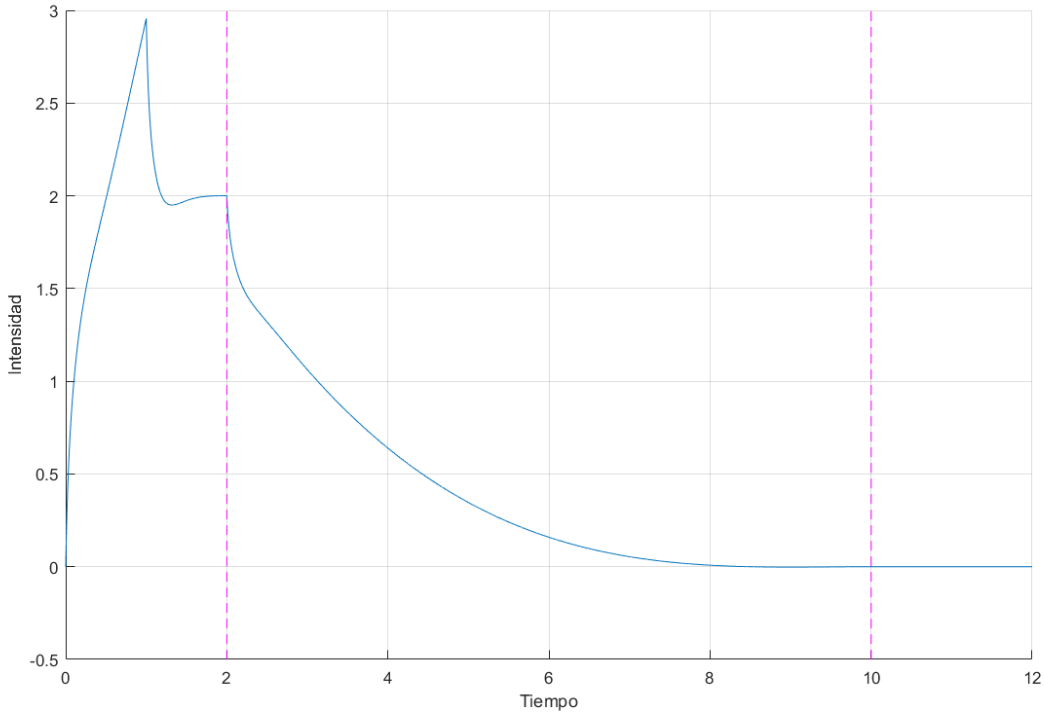


Figura 3:14 Intensidad para la simulación de K=0.5

Por otro lado, podemos ver la intensidad en la Figura 3:14. En un caso de K=0.5, la intensidad siempre es mayor a 0, lo que en un caso real simplifica las cosas, ya que cambiar de signo la intensidad de un motor real puede producir errores a la hora de seguir una trayectoria precisa. Si cambiamos la K a una un poco más grande, podemos encontrarnos que al final de la trayectoria debemos frenar tan bruscamente que necesitamos intensidades negativas al final, como se puede apreciar en la Figura 3:15 para la cual se ha utilizado una K mayor. Sin embargo, para K=0.5 o menor encontramos que la fuerza necesaria para detenernos por completo va disminuyendo de manera uniforme hasta 0. Por otro lado, si disminuimos mucho la K obtenemos un movimiento mucho más suavizado, pero mucho más lento de realizar, por lo que tampoco queremos que la detección dure muchísimo al estar relacionado inversamente el tiempo de maniobra con la K escogida. Un ejemplo de una K más pequeña, K=0.3 la podremos ver en la Figura 3:16. En esta figura se apreciará mejor que implica un movimiento con una K menor.

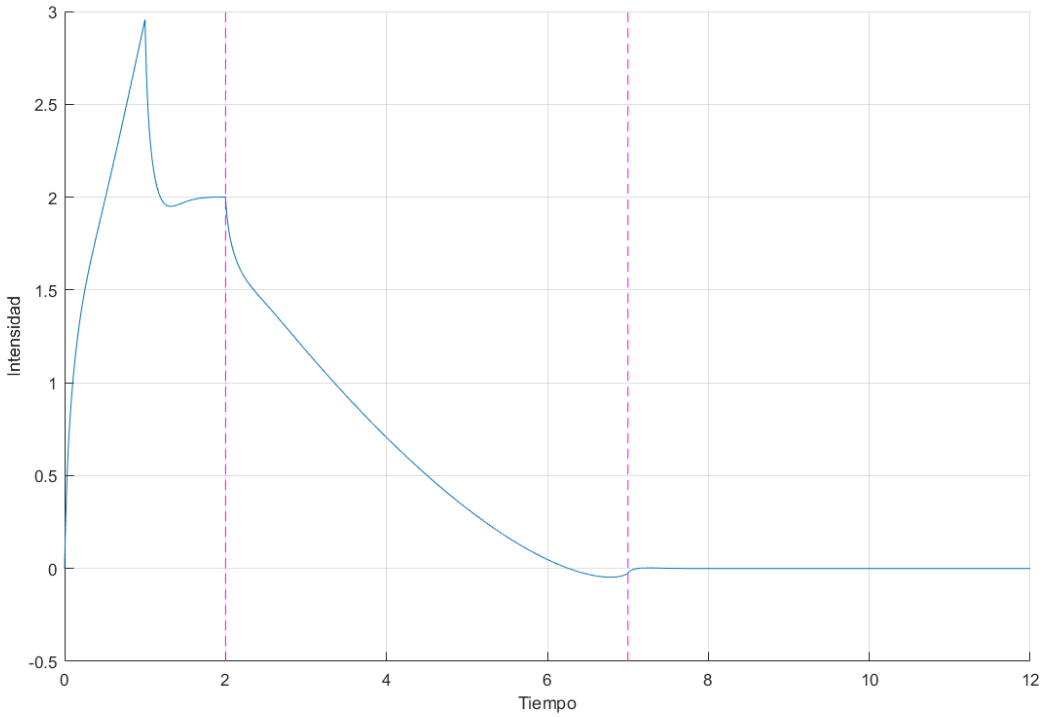


Figura 3:15 Intensidad para la simulación de K>0.6

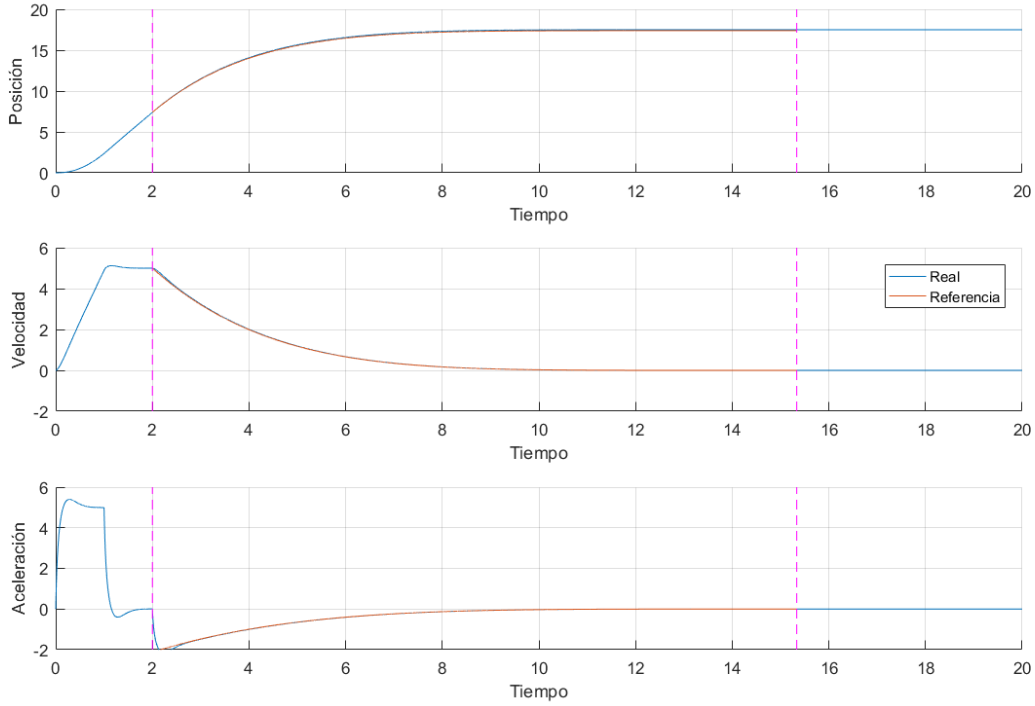


Figura 3:16 Simulación K=0.3

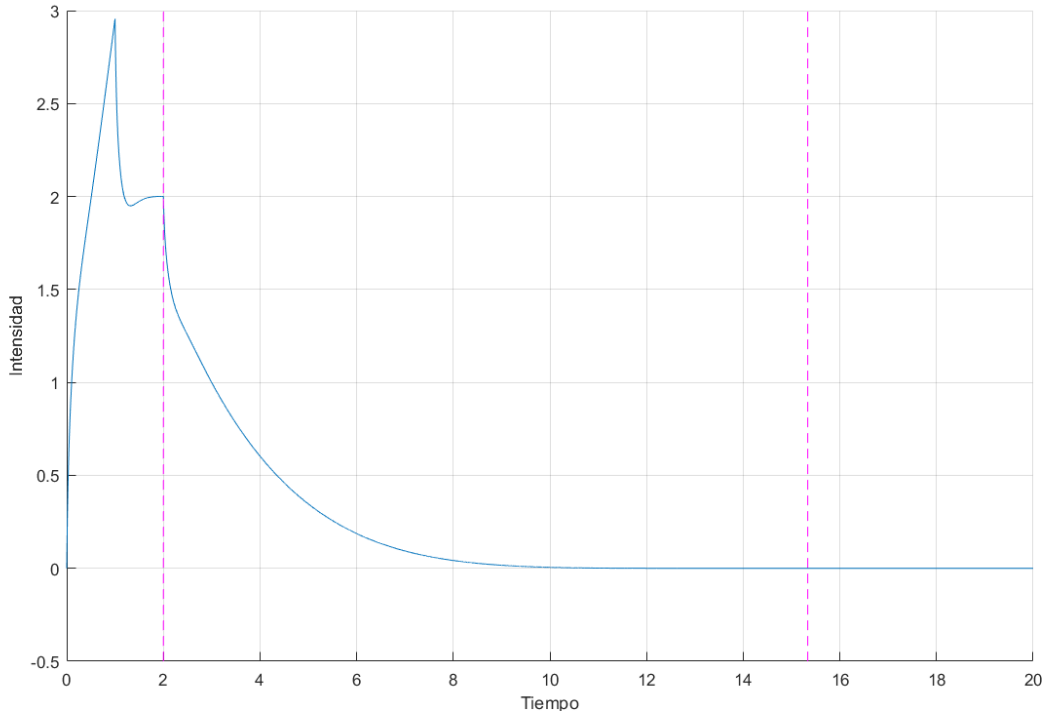


Figura 3:17 Intensidad para K=0.3

Como se puede ver en la Figura 3:16, el movimiento es incluso más suavizado como ay se había comentado, pero la velocidad hasta alcanzar el objetivo a la mitad del movimiento es tan pequeña que carece de sentido escoger una K tan pequeña, ya que el tiempo sobrante seria desperdiciado en un sistema móvil terrestre, que no necesita precisión milimétrica. Por otro lado, a la hora de implementarlo en un sistema que necesita una precisión mayor, una K más pequeña puede ser una mejor opción. La intensidad de esta K=0.3 como podemos ver en la Figura 3:17, alcanza el 0 mucho antes.

3.2.1 Conclusiones, K elegida

A partir de todas estas simulaciones, podemos ver de una mejor forma, que el movimiento propuesto por la Teoría τ, es un movimiento genérico que deberá ser adaptado según la situación. En nuestro caso, ya que queremos un movimiento suave, nuestra K deberá ser menor o igual que 0.6 para evitar una frenada agresiva en los últimos instantes y por otro lado para que la maniobra no sea muy duradera en los últimos instantes y puesto que no necesitamos una gran precisión en la posición final al no ser un vehículo aéreo, para nuestro robot una K superior o igual a 0.4 es más que suficiente para alcanzar el punto objetivo sin errores notables.

Luego, en resumen, nuestra K será: $0.4 \leq K \leq 0.6$.

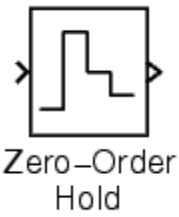
3.3 Simulaciones más realistas.

Una vez comprobadas las diferentes características de la trayectoria sugerida por esta teoría, es hora de implementar una simulación un poco más avanzada y que nos acerque más al mundo real, es por esto que a continuación se incluirán las dos características más comunes que nos encontramos en la realidad, el tiempo de muestreo y los errores de medición.

3.3.1 Tiempo de muestreo

El tiempo elegido para el intervalo en el que se actualizara el control es de 0.1seg, el cual luego en el sistema real será cambiado por otro debido a limitaciones en los sensores (Ver punto 5.1.1). Se ha elegido este tiempo porque será mayor al tomado en el robot.

Para la discretización del sistema se usará el bloque:



Este bloque implementa la decodificación más simple a la hora de convertir señal analógica a digital, pero, sin embargo, es el que estará presente en el sistema real de nuestra Raspberry, por lo que será suficiente para la simulación.

Una vez colocado el bloque en la simulación anterior, el resultado se puede ver en la Figura 3:18. Se puede ver como el control no cambia mucho sin siquiera implementarse ningún tipo de filtro. Los resultados son simulares, luego se puede suponer que el robot real podrá afrontar esta dificultad.

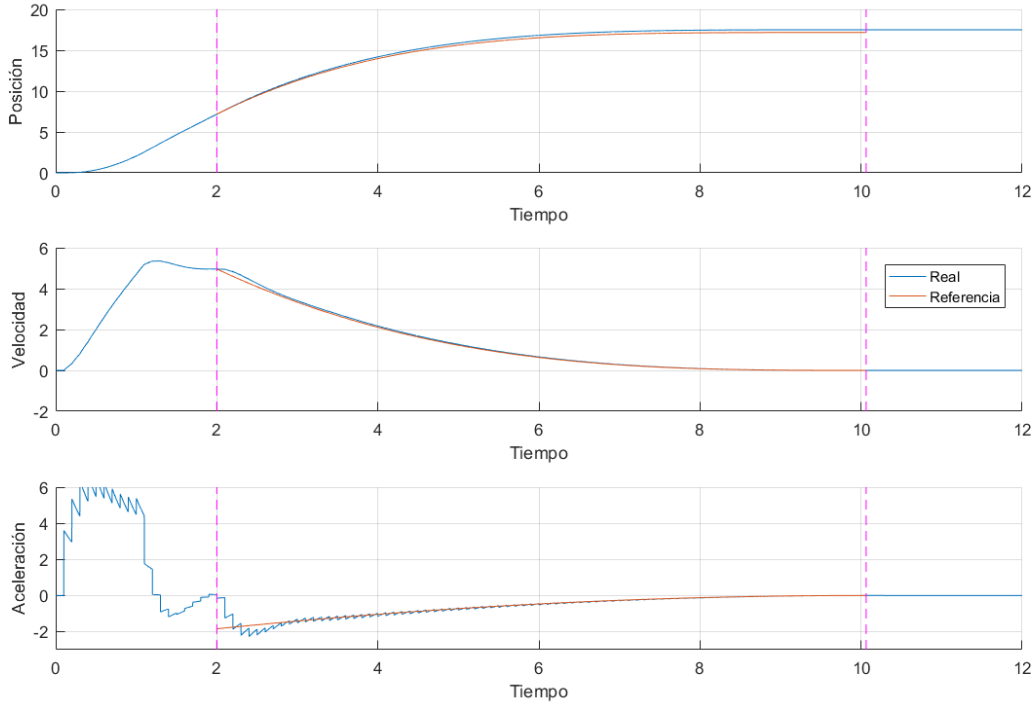
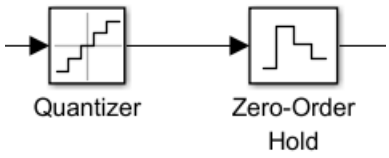


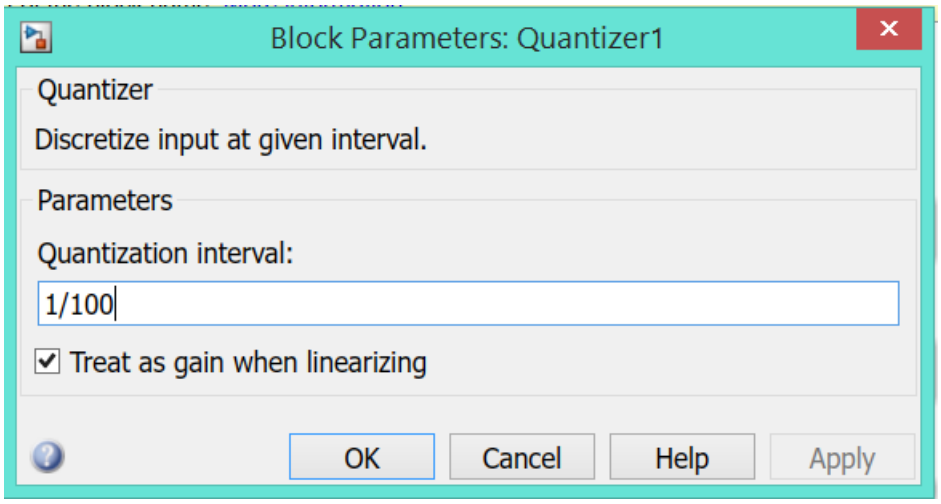
Figura 3:18 Simulación con tiempo de muestreo de 0.1 segundo

3.3.2 Errores en las medidas

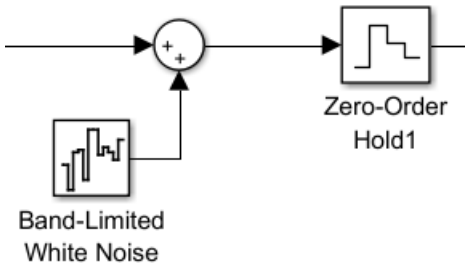
Por otro lado, para incluir un error en las señales de salida, se le introducirá un error a la función que genera. Para la posición, y simulando un encoder un error tipo cuantizador ya que se tomarán medidas a través de un encoder en el motor y a la salida un zero-order hold anterior simulando la toma de medidas de cada intervalo:



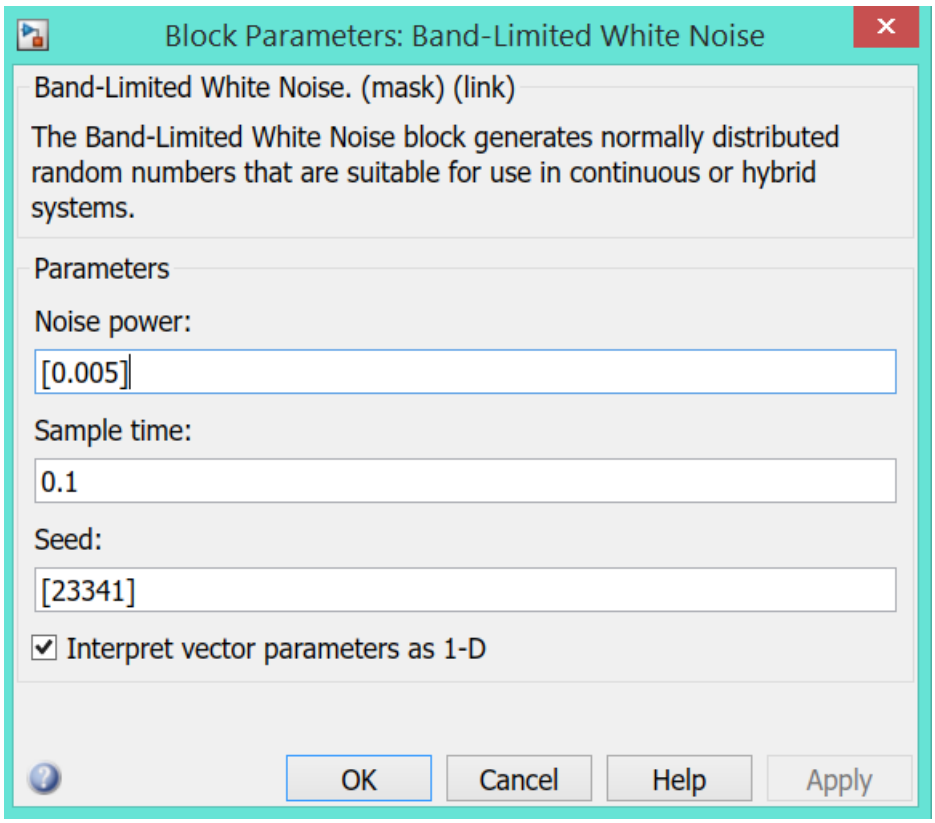
Dentro del bloque cuantizador se colocan los siguientes valores:



Para la velocidad un error blanco, ya que se obtendrán las medidas de una relación entre el encoder y otro sensor tipo Time of flight y también el bloque de zero-order hold al final:



Dentro del bloque ruido blanco se colocan los siguientes valores:



Para la aceleración no se han colocado errores ya que no afecta al control, es un parámetro que sirve solo para dar información. Los errores tanto de posición como de velocidad, se han escogido de manera que sean superiores a los sensores realmente usados. Tanto el encoder como los otro sensores y sus características se verán en el apartado de construcción del robot móvil. Una vez implementadas estos errores y simulando, obtenemos que el controlador es tan robusto que sin implementar ningún sistema de filtro funciona correctamente. Los resultados se pueden ver en la Figura 3:19 y posteriores.

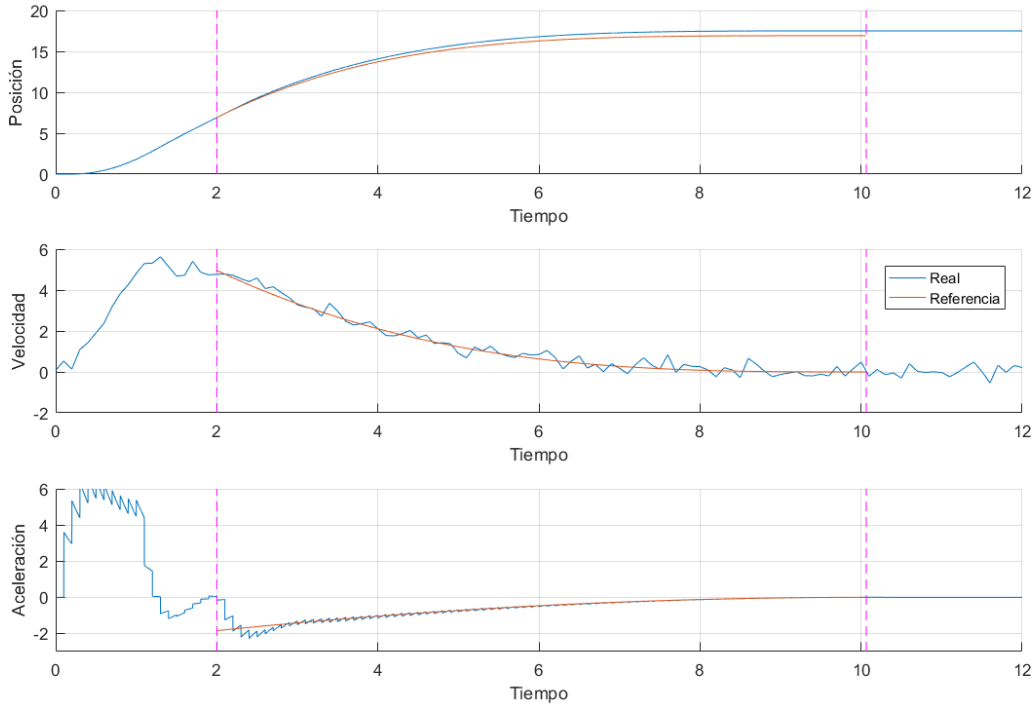


Figura 3:19 Simulación con la implementación de los errores en las medidas

A continuación, se muestran las gráficas en mayor tamaño para apreciar con más detalle sus diferentes características:

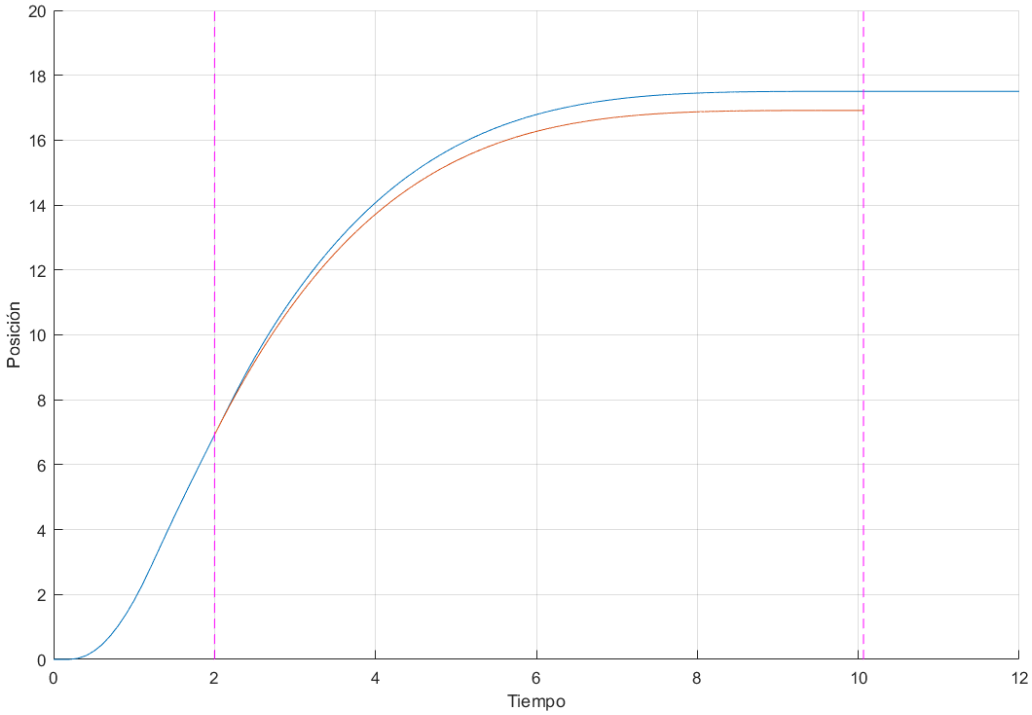


Figura 3:20 Posición para la simulación con la implementación de errores en las medidas.

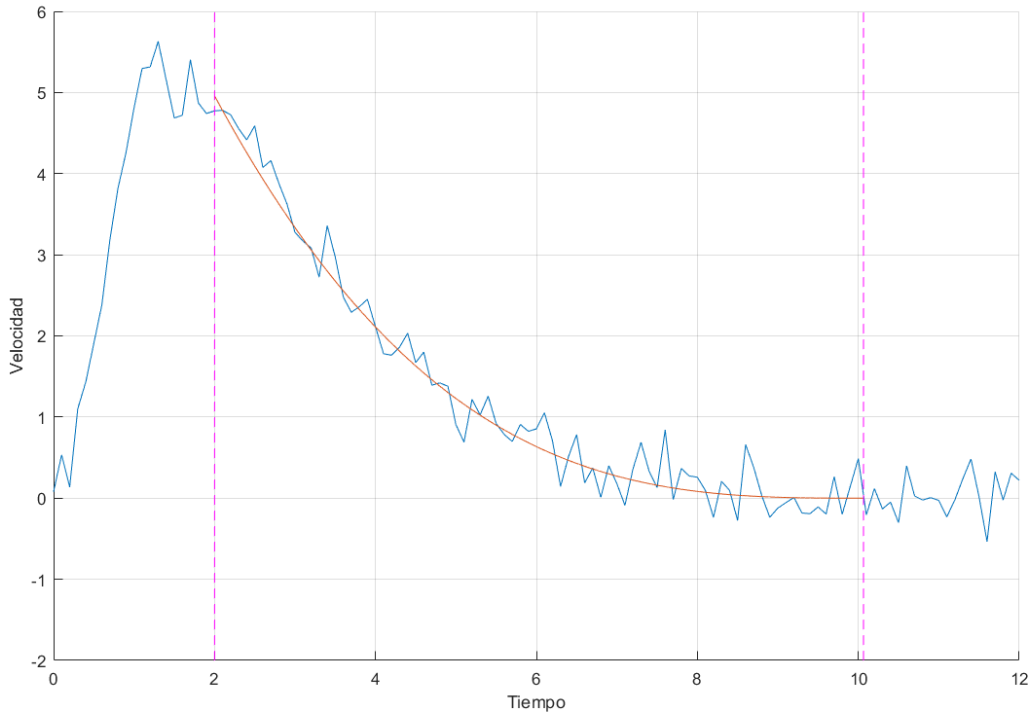


Figura 3:21 Velocidad para la simulación con la implementación de errores en las medidas.

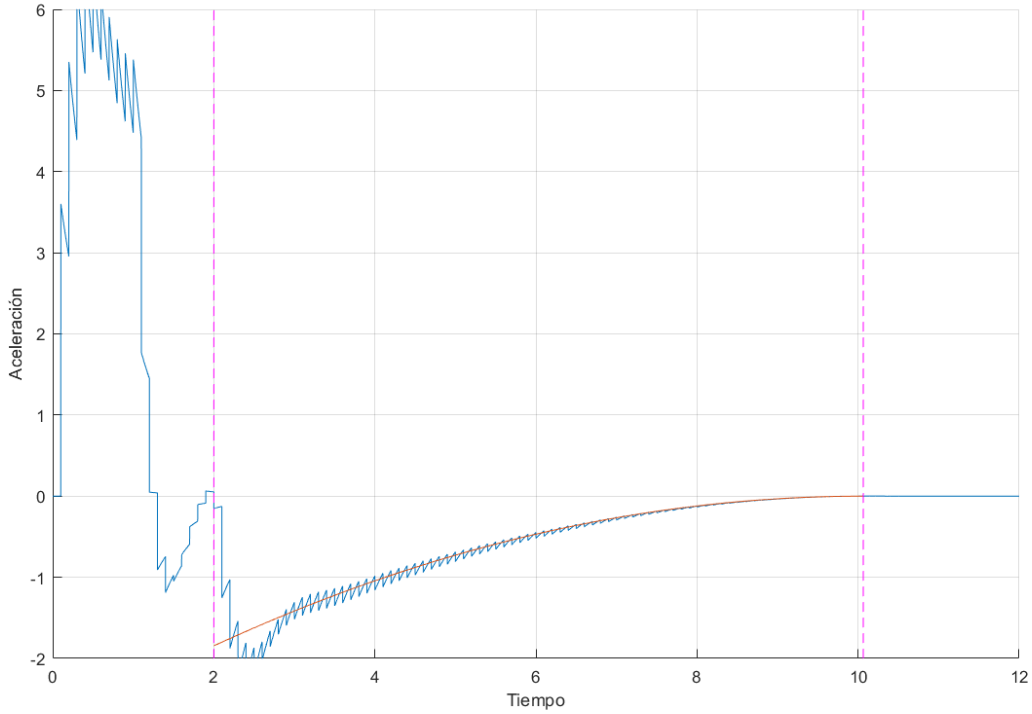


Figura 3:22 Aceleración para la simulación con la implementación de errores en las medidas.

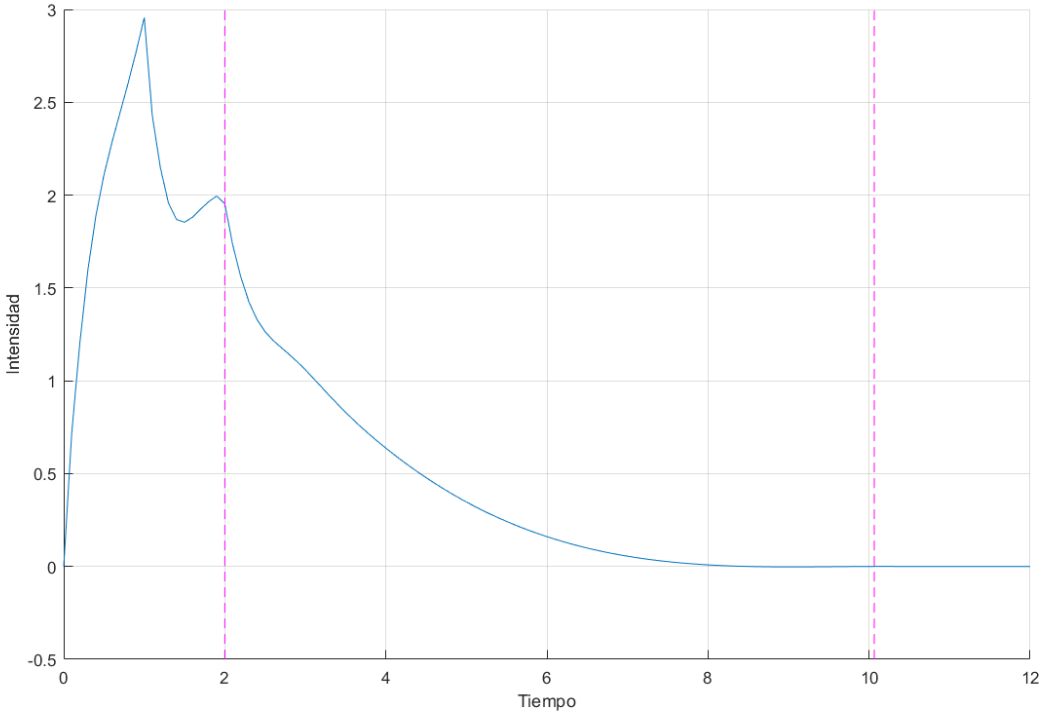


Figura 3:23 Intensidad para la simulación con la implementación de errores en las medidas.

El único error destacable es el relacionado con la posición, pero realmente este fallo es debido al error introducido en el encoder. El error aun así no es muy importante, ya que el error total es aproximadamente de una unidad de distancia. A la hora de la prueba real la posición estará realimentada no solo con los datos del encoder que estará más destinado a medir la velocidad, sino que estará más directamente relacionada con el sensor de distancia laser, el cual gana precisión conforme más cerca está el objetivo, llegando a eliminar los errores el final del recorrido. Estos detalles sobre el sensor se verán en el apartado 4.1.2, se verán las especificaciones de compra, y más adelante en el 5.1.2, se verán los detalles del funcionamiento, y por último en el Apéndice 2 se verá una comparación entre este sensor y el de ultrasonidos en el tema de errores en las medidas sobre todo.

4 CONSTRUCCIÓN DEL ROBOT MÓVIL

«La ciencia instrumental, o mecánica, es la más noble y la más útil, por encima de todas».

- Leonardo da Vinci -

Para continuar con el Proyecto y pasar de las simulaciones anteriores al plano real, es necesario la creación de un robot móvil. En este robot, se podrá introducir lo equivalente al código visto anteriormente, y comprobar, si realmente se puede llevar a cabo la implementación de un movimiento bio-inspirado. Es por esto, que en este apartado se relatará la construcción del robot, así como sus diferentes sensores y materiales. Además, de por qué se han escogido estos frente a otros.

En la parte final de este apartado, se podrán ver fotos del resultado obtenido tras montar todos los componentes. También, se podrá ver, un esquema sobre cómo se ha conectado todo, intentando que quede lo más claro posible el procedimiento seguido para su construcción.

4.1 Elementos usados

A la hora de construir el robot, se debía pensar en la aplicación futura, es por eso, que al principio de todo y pretendiendo enfocar la velocidad como base a la hora de ejecutar un control, una forma bastante precisa de conocer la velocidad, sería algo bastante importante. Por otra parte, debía existir una manera de conseguir una lectura de la distancia a la que se encuentre el objeto, ya que es un dato indispensable a la hora de ejecutar la Teoría τ. Por último, obviando los elementos mecánicos como chasis y ruedas, es la placa que administraría todas las entradas de datos y actuaría en el motor para que el robot cumpliera la trayectoria deseada, para la que se ha elegido la placa Raspberry Pi.

4.1.1 Motor con encoder

El motor escogido es un motor DC con encoder incluido y Relación del engranaje de 70:1 que le hace alcanzar 157rpm a la máxima alimentación. Se puede encontrar su vista en la Figura 4:1. Este motor trabaja a 12V como máximo y con un voltaje mínimo del 10% de este para comenzar el movimiento, esto limitará el rango útil del control, pero no será un problema. Fuente de la información en [9]

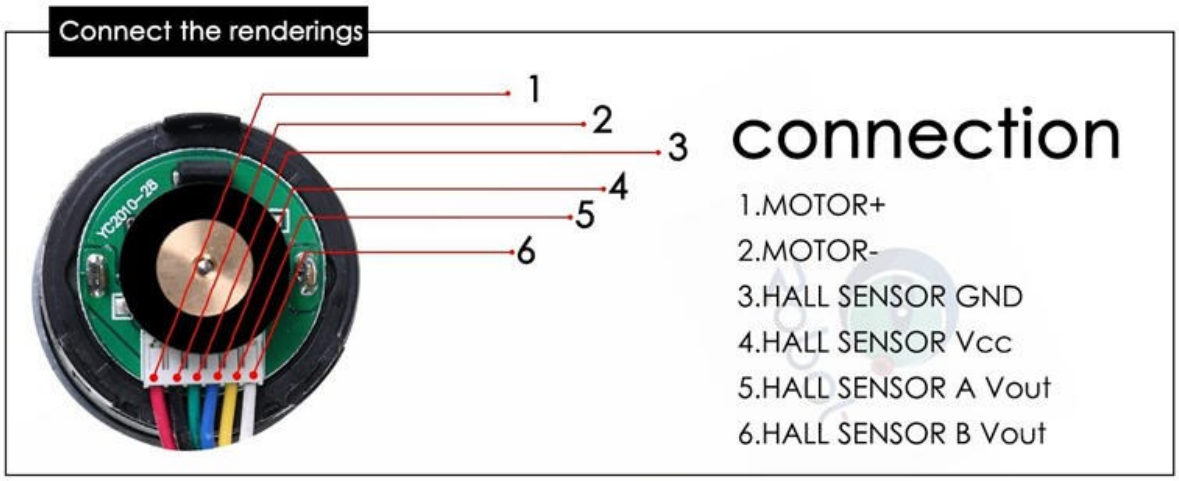
Figura 4:1 Motor usado.

Este motor contiene un encoder del tipo Hall, los tipos de encoder y su funcionamiento serán vistos más detenidamente en el Apéndice 1. Este encoder cuenta con dos salidas, que están desfasadas 90º, aunque en nuestro caso no vamos a utilizar este desfase, si se utilizaran las dos salidas de este para la recogida de datos, pero eso es algo que se verá más adelante, con la creación del código de implementación del control en el robot real.

Algunos datos adicionales del motor:

- Sin Carga Velocidad: 6V - 78RPM / 12V - 157RPM (+ / - 10%)
- Sin Carga Corriente: 6V - 120mA / 12V - 250mA
- Rotor Bloqueado Corriente: 6V - 3.0A / 12V - 6.5A
- Rotor Bloqueado Par: 6V - 14Kg.cm / 12V - 28Kg.cm
- Nominal Par: 6V - 3.5Kg.cm / 12V - 7Kg.cm

Sin embargo, en la siguiente Figura 4:2 que hemos encontrado en la referencia del motor, comentada anteriormente, se puede ver el encoder:



Specifications	Code	Test Conditions	Small	Basic	Big	UNITS
Input Voltage	VCC	---	4.5	---	24	V
Output current	IC	Vcc=12V,GAUSS<-170	---	<0.1	20	MA

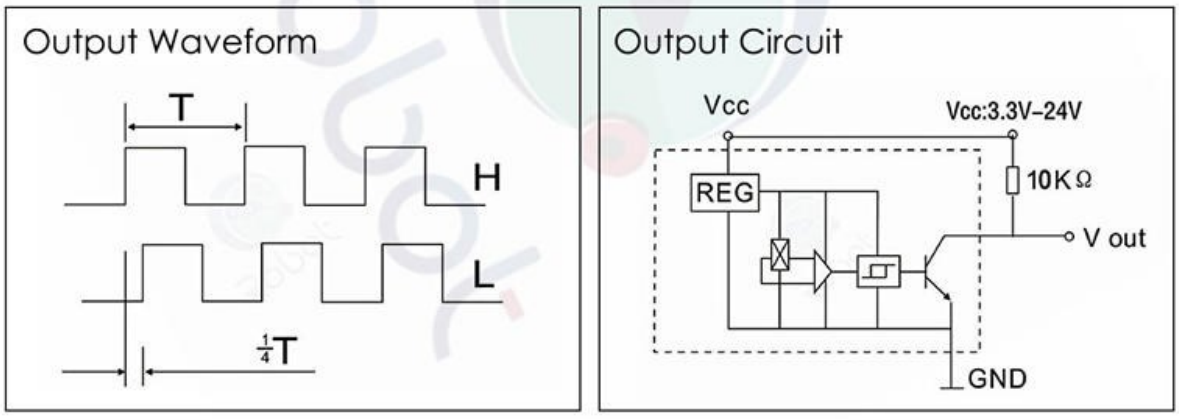


Figura 4:2 Características del encoder.

Al tener una Relación del engranaje de 70:1 y 16 pulsos por ciclo, el encoder genera: 70*16=1120 pulsos por vuelta, lo que nos dará una aproximación bastante buena de la velocidad.

Como ya se ha comentado, el motor trabaja a 12V, pero la alimentación del robot se realizará con baterías que alimentan a 5V para ello se necesitará un convertidor DC/DC de 5V a 12V, que se muestra en la Figura 4:3. Como elección para convertidor DC/DC elevador de bajo coste las opciones eran pocas, así que se optó por un convertidor usado para Amazon Echo Spot y modificándolo debidamente para que encajara en el proyecto.

Sus características más destacables son, su ligero peso, pesa solo 25 g, lo que te permite llevarlo fácilmente y no afecta al peso del robot comparado con otros componentes. Por otro lado, sus dimensiones también son pequeñas al ser un convertidor estático, su tamaño es de: de 15,2 x 7,6 x 1,27 cm.

Figura 4:3 Convertidor DC/DC para la alimentación del motor DC.

Además, para controlar el motor que funciona a un voltaje diferente al de la Raspberry, ya que esta solo puede dar como máximo 3.3V de salida, se necesita la utilización de un módulo que genere una señal de salida al motor de 0V a 12V. Para esto, se enviará una señal desde la Raspberry a través de un PWM que actuará sobre este módulo. El circuito elegido está basado en un puente H y es usado comúnmente en robots móviles controlados por microcontroladores. El módulo se llama Neuftech L298N Dual H y se puede encontrar información sobre él en [10].

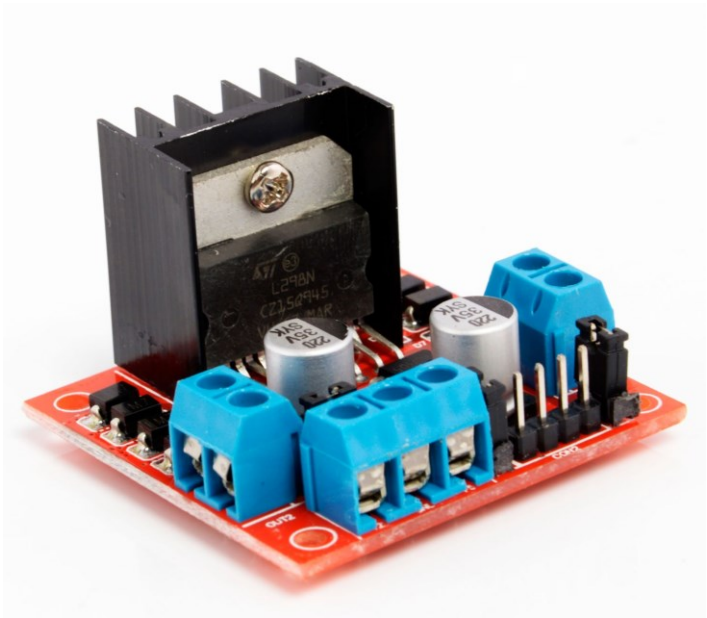


Figura 4:4 Modulo Neuftech L298N Dual H.

4.1.2 Sensor de distancia

A la hora de enfrentar este proyecto unos de los elementos más importantes es como medir la posición actual del robot y cuanta es la distancia hacia el objeto, es por esto, que la elección de un sensor que sirva de una manera correcta para este fin debe ser algo que se elija con cuidado.

Como primer método para medir la posición relativa del robot respecto a la posición objetivo, se pensó en el sensor de ultrasonidos HC-SR04 que se puede ver en la Figura 4:5, [11]. Este sensor, es típicamente usado en proyectos con Arduinos o microcontroladores.

Figura 4:5 Sensor Ultrasonidos.

Sin embargo, este sensor, además de ser muy lento en lectura, tiene un alcance máximo de 0.3 metros a una precisión aceptable para nuestro propósito que es generar una trayectoria concreta, se puede ver la referencia al estudio en el Apéndice 2. Es por esto, que se le intentó buscar un sustituto, y se llegó al encuentro de un sensor laser del tipo Time of Flight llamado VL53L0X que se puede ver en la Figura 4:6. Su datasheet, [12] , ya daba resultados más prometedores que el anterior y tras la recogida de información que se comentara en el apéndice, como ya se ha dicho, se optó por escoger este sensor. Sin embargo, la placa de adaptación escogida para este sensor es la copia GY-VL53L0XV2, que, aunque en prestaciones es similar, a la hora de comunicarlo con la placa Raspberry pi ha hecho falta la modificación de la librería [13] que esta implementada en Python, el lenguaje que hablara la Raspberry para este proyecto. Esta librería, ha sido la base para la modificación al estar basada en Python, que con ayudad de la librería de Arduino que, si incluye la modificación para este sensor, [14], se ha conseguido el correcto funcionamiento de este sensor.

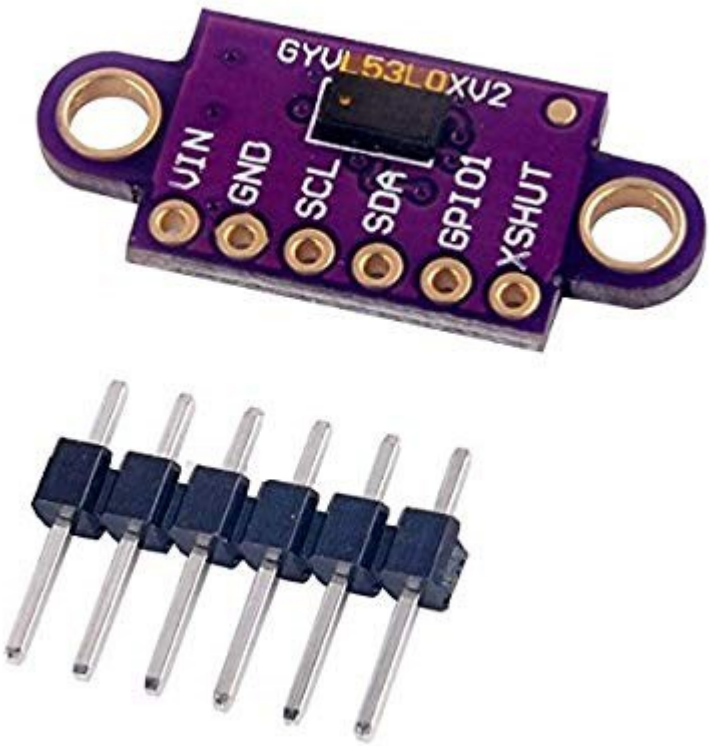


Figura 4:6 Time-of-Flight Distance Sensor GY-VL53L0XV2.

Las características de este sensor, aunque ya vienen detalladas en su datasheet serán reveladas durante el procedimiento de este trabajo según correspondan. Aunque, a continuación, se comentarán los diferentes pines que este posee y para que se usa cada uno, con el fin de ayudar en la explicación de la posterior explicación cableado.

Los 6 pines contenidos en el sensor, tienen el funcionamiento:

Tabla 4:1 Pineado del sensor GY-VL53L0XV2

Nombre Pin	**Tipo señal**	**Descripción**
VIN	Suministro	Conecta el sensor a la alimentación.
GND	Referencia	Conecta el sensor a la referencia de tierra o masa.
SCL	Entrada Digital	Entrada de la señal de reloj del I2C.
SDA	Entrada/Salida Digital	Conexión de datos del I2C.
GPIO1	Salida Digital	Salida de interrupciones.
XSHUT	Entrada Digital	Pin de encendido/apagado. Desactiva el sensor a nivel bajo.

4.1.3 Raspberry Pi

Como microcontrolador que ejecute el control se ha pensado en una Raspberry pi, ya que, aunque no destaque por su gran manejo en la GPIO como pueden ser de los microcontroladores, en tema de comunicaciones y potencia de cálculo es bastante fuerte. Debido a esto es mucho más fácil enviar y recibir información a través de una Raspberry Pi que con otro controlador.

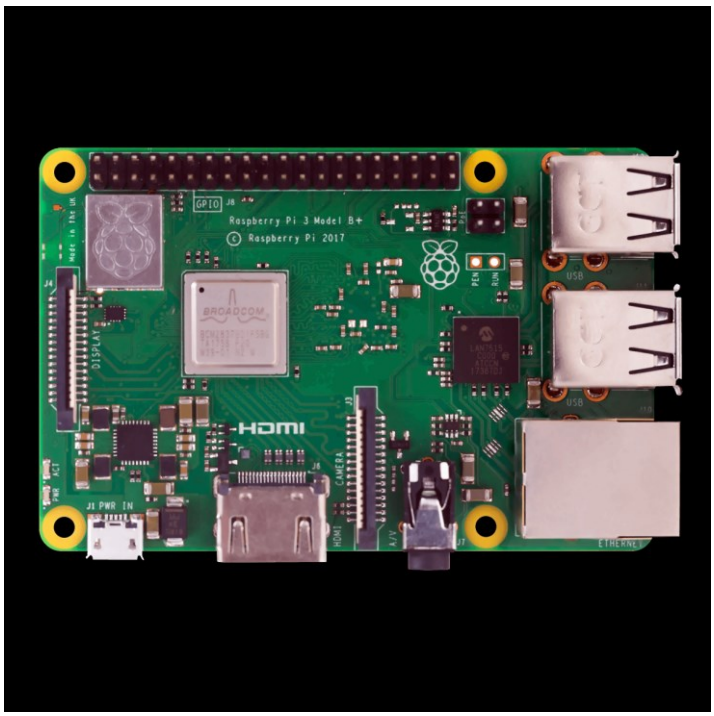


Figura 4:7 Microcomputador Raspberry Pi 3 Model B.

Basándonos en la información ya recogida sobre la historia de Raspberry en [15], podemos ver cómo ha mejorado y las características que ha adquirido hasta su último modelo, que son las siguientes:

- CPU + GPU: Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
- RAM: 1GB LPDDR2 SDRAM
- Wi-Fi + Bluetooth: 2.4GHz y 5GHz IEEE 802.11.b/g/n/ac, Bluetooth 4.2, BLE
- Ethernet: Gigabit Ethernet sobre USB 2.0 (300 Mbps)
- GPIO de 40 pines
- HDMI
- puertos USB 2.0
- Puerto CSI para conectar una cámara.
- Puerto DSI para conectar una pantalla táctil
- Salida de audio estéreo y vídeo compuesto
- Micro-SD
- Power-over-Ethernet (PoE)

Aunque la mayoría de estas características no les vallamos a sacar provecho es interesante tenerlas en cuenta, ya que algunas si serán usadas y en el caso de una continuación del proyecto pueden resultar útiles.

Otro detalle de este miniordenador es que tiene un GPIO como si de un microcontrolador o Arduino se tratase, incluso uno de los más complejos y sencillos de usar. Su distribución es el siguiente:

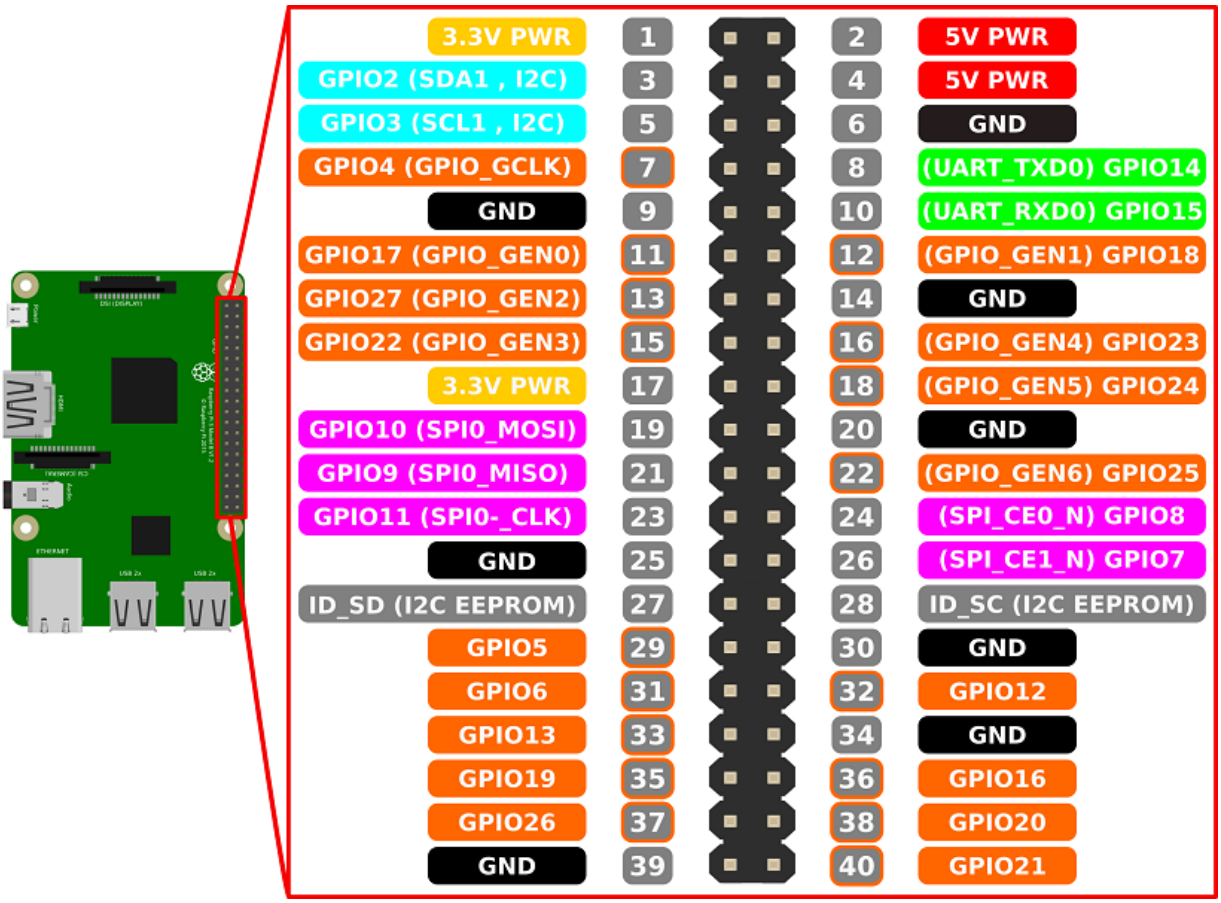


Figura 4:8 GPIO Raspberry Pi 3 Model B.

4.1.4 Alimentación

A la hora de pensar en cómo se podría alimentar el robot móvil, uno de los requisitos que se considera indispensable es que fuera aislado de la red de tensión eléctrica, ya que se deseaban evitar cables externos que pudieran limitar la movilidad del coche. De igual forma que una de las características por la que se ha elegido la Raspberry frente a otro dispositivo es su fácil conexión a internet, o al menos a una red LAN, sin el uso de hardware externo.

Tras toparme con la necesidad de alimentar el robot de una manera que fuera capaz de estar aislado, la mejor solución fue optar por una batería portátil, en este caso de 5V que es la más común y a la que se alimenta la raspberry.

Además de haber usado un convertidor DC/DC para el motor como se ha comentado en el apartado sobre el mismo, no ha sido el único inconveniente encontrado. Durante la construcción del robot, se encontró el problema de que, al ir conectados a la misma fuente, el motor y la Raspberry, el motor demanda una intensidad muy alta para la batería que, al ir conectada también a la Raspberry, interfiere en su funcionamiento. Esta interferencia, llega al punto de reiniciar la Raspberry o inutilizarla en una exposición continua a este fenómeno. Debido a este problema, se han tenido que usar dos baterías portátiles separadas, pero con las tierras unificadas para conseguir la misma referencia en ambas y poder ejecutar el control del motor correctamente desde la Raspberry en el puente H.

4.1.5 Chasis y otros elementos

Para la construcción del robot se ha tomado como base para el chasis una estructura de madera plastificada para obtener flexibilidad y resistencia. A partir de esta barata estructura, se han imprimido en 3D algunas piezas. Estas piezas en 3D, así como el proceso y más detalle sobre este método se podrá encontrar en el Apéndice 3.

Tras colocar las diferentes piezas en sus lugares correspondientes, nos quedaría en la parte inferior del robot las ruedas y el motor unido a las ruedas delanteras mediante un engranaje, cuyos dientes no son rectos sino helicoidales cómo se puede ver en la Figura 4:9.

Figura 4:9 Engranajes usados para la transmisión de potencia entre el motor y el eje del robot móvil.

A la hora por decantarme por este tipo de dientes ha entrado en rigor las diferentes características de cada uno, como se podría ver por ejemplo en [16] los engranajes con cierto ángulo en los dientes tienen considerables ventajas respecto a dientes rectos.

- Una de las principales ventajas es que al tener siempre en contacto un diente o más, la transmisión de fuerza es constante y no se producen picos, esta es una de las razones principales por la que lo hemos escogido intentando priorizar la suavidad del movimiento y que no haya limitantes por medio del hardware.
- Otras ventajas menores en nuestro caso, es que se reduce el ruido producido por estos al trabajar y que pueden llegar a transmitir mayores potencias.

Sin embargo, estos tipos de dientes también generan algunos inconvenientes, los cuales pese a ser graves en un entorno industrial, en nuestro caso han sido fáciles de evitar o solventar.

- En primer lugar, estos engranajes poseen un coste de producción bastante más alto que los de dientes rectos, pero, al estar impreso en 3D, el coste tanto de material como de tiempo ha sido el mismo que si se hubiera usado un engranaje con dientes rectos.
- Otras desventajas pueden ser el alto desgaste de estos engranajes con el tiempo que en nuestro caso tampoco afectaría, luego su funcionamiento será de un tiempo bastante limitado y por último tienen menor eficiencia, pero no es un dato destacable y en nuestro caso no es reseñable.

Para elegir las ruedas, el principal factor que influye es, evitar el deslizamiento, para ello se han elegido unas ruedas usadas comúnmente en coches RC, hechas de goma e imitando el dibujo y por tanto la adhesión de una rueda de coche, estas se pueden ver en la Figura 4:10. Como dato adicional, las ruedas tienen un diámetro de 6,8 cm y de altura son de 2,8 cm.

Figura 4:10 Foto de las ruedas que se incluirán en el robot móvil.

Además de todo lo anterior, y por último lugar, se le han añadido los dos LEDs que aparecen en la Figura 4:11. Estos, irán localizados en la parte trasera del robot móvil simulando luces de freno, las cuales se encenderán al comienzo de la maniobra y se apagarán con la finalización de la misma. Los leds incluidos en el robot de manera final se podrán ver en las fotos del siguiente apartado.

Figura 4:11 LEDs que serán incluidos en la parte trasera del robot.

4.2 Construcción y Pineado

4.2.1 Esquema de conexiones y Pineado

Tras la descripción de los componentes, el siguiente paso es el montaje, pero antes de eso, hay que saber las distintas conexiones de los elementos y como estarán realizadas.

En las Figura 4:12 y Figura 4:13 se mostrarán unos esquemas de conexionado, pero se resumirá a continuación el conexionado. Simplificando, dos baterías alimentando una al motor y otra a la Raspberry. La batería que alimenta al motor contiene un convertidor DC/DC de 5V a 12V y a continuación el circuito comentado anteriormente, puente H, y por último el motor. La otra batería alimenta a la Raspberry y esta se comunica mediante su GPIO al circuito de puente H que controla el motor, al sensor de distancia ToF y a los LEDs.

A modo de que la Figura 4:12 y Figura 4:13 queden mejor explicadas comentar:

- S1 y S2 son las diferentes salidas del encoder, y este va alimentado desde la Raspberry a 3.3V a través de un pin para activarlo o desactivamos si queremos.
- La señal PWM, IN1 e IN2 van al puente H. PWM indica el voltaje de salida, IN1 e IN2 indican si el motor va hacia delante o hacia atrás o si este está activo. Se puede encontrar más información sobre este funcionamiento en la referencia comentada anteriormente [10].

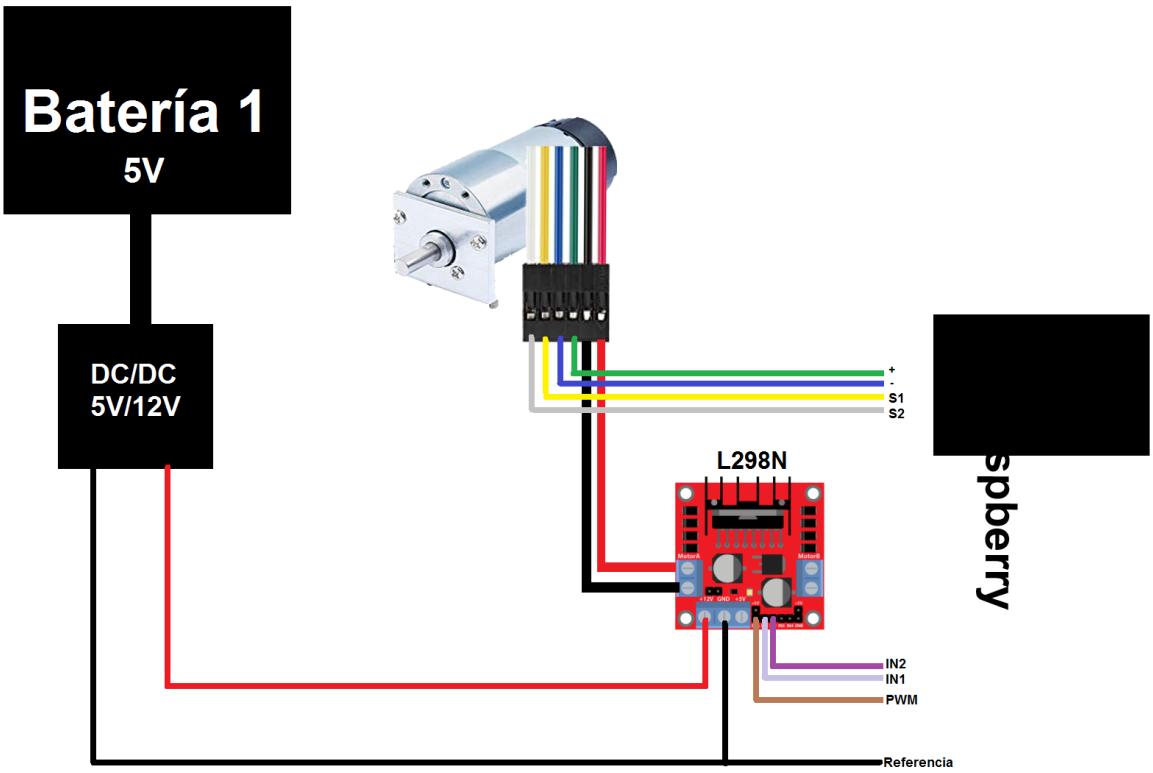


Figura 4:12 Conexionado. Parte 1. Motor y Puente H.

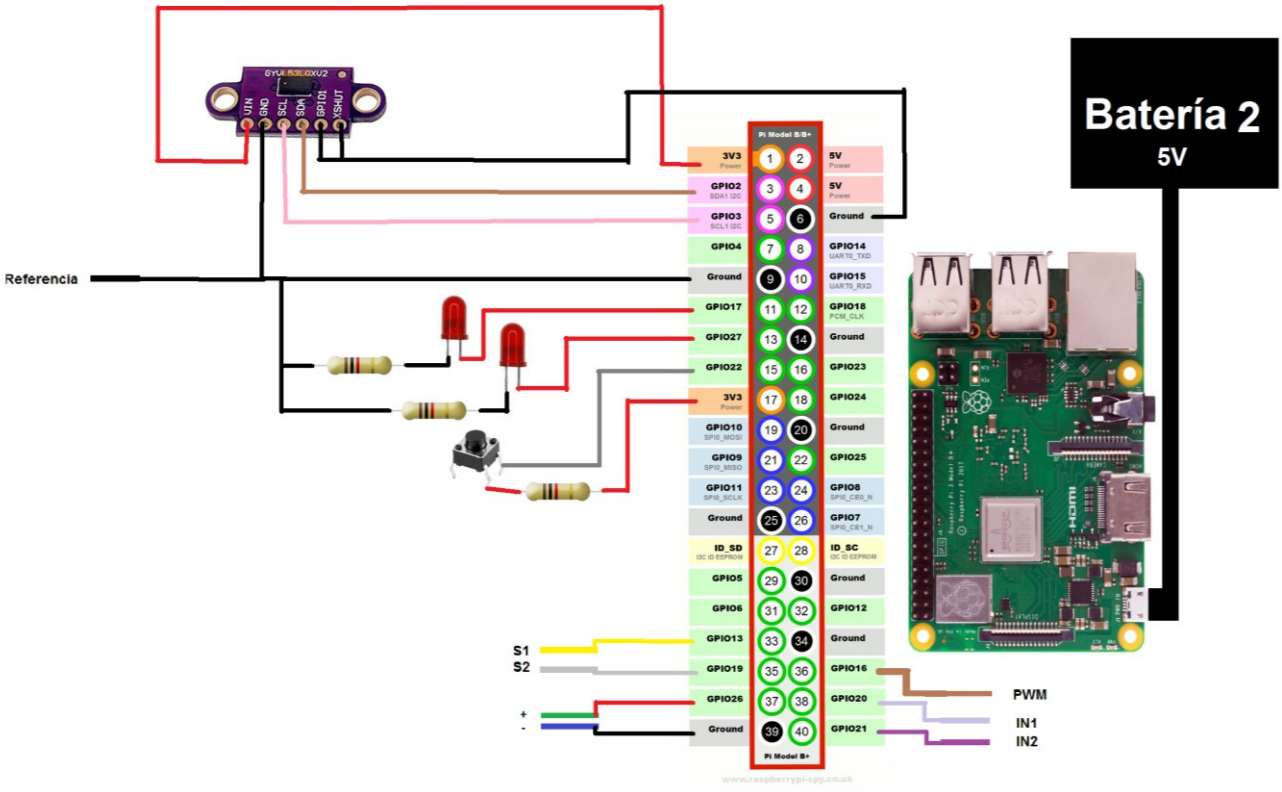


Figura 4:13 Conexionado. Parte 2. Raspberry, sensor laser y accesorios.

4.2.2 Resultado

Una vez conectado todo y montado los resultados se pueden ver en la Figura 4:14:

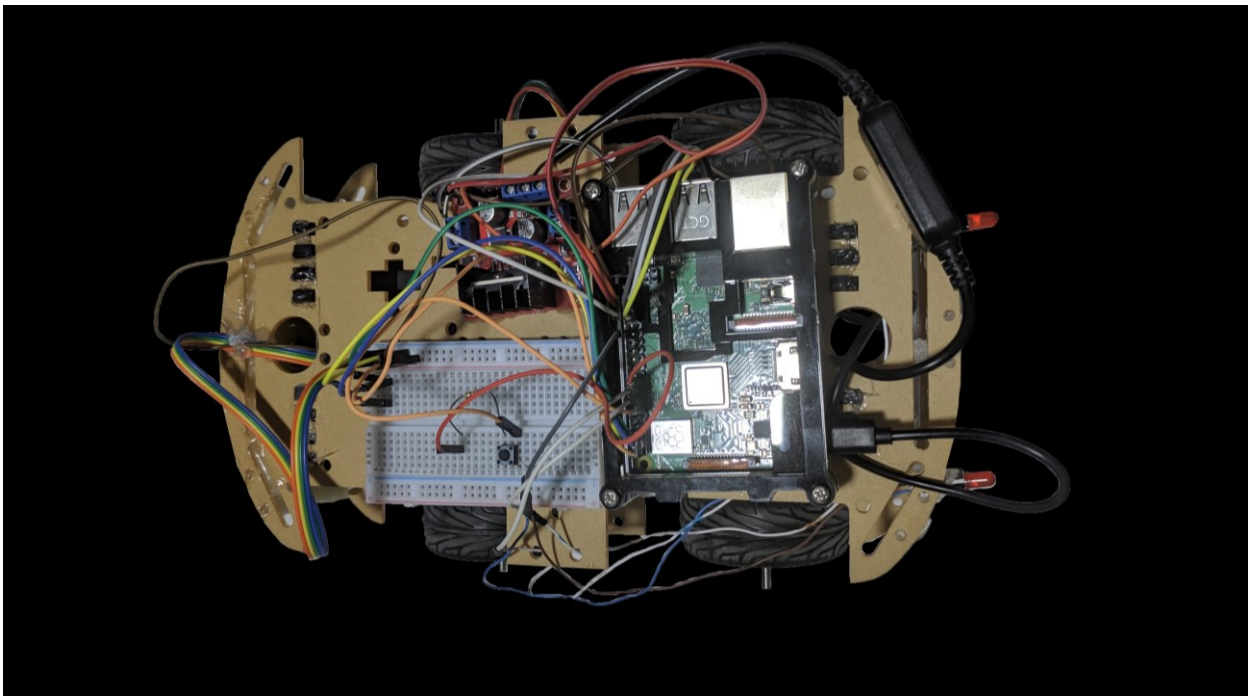


Figura 4:14 Robot terminado. Vista completa.

Aunque la Figura 4:14 anterior muestra la mayor parte del robot, a continuación, se mostrarán varias fotos más para ver con detalle los elementos más importantes.

En la Figura 4:15 se puede apreciar con detalle la colocación del sensor VL53L0X en un material no rígido para evitar daños si el robot choca contra el objeto por fallos o durante las pruebas de funcionamiento. Así como en la Figura 4:16 se puede ver cómo se han colocado las luces traseras que se mantendrán encendidas durante la maniobra.

Figura 4:15 Robot terminado. Vista frontal.

Figura 4:16 Robot terminado. Vista trasera.

Figura 4:17 Robot terminado. Vista interior.

Del mismo modo, en la Figura 4:17 se puede ver el compartimento interior en el que van colocadas las baterías ya que será el componente más pesado y para mantener el centro de gravedad bajo se han colocado en la parte inferior. También, colocar la mayor parte de la electrónica en la parte superior simplifica la visualización y el conexionado. Además de las baterías, el otro elemento más pesado es el motor que se ha colocado en la parte baja del robot para mantener aún más bajo el centro de gravedad y facilitar la transmisión de energía mediante engranajes con el eje que contiene las ruedas delanteras, como se podrá apreciar en la siguiente fotografía. Si nos centramos en la parte de abajo, en la Figura 4:18 se puede apreciar cómo van conectados los engranajes, una en el eje del motor y otro en el eje de las ruedas, su funcionamiento es simple de ver.

Figura 4:18 Robot terminado. Vista de los bajos.

Para finalizar con la muestra de lo que se ha obtenido durante la construcción, se puede ver en la Figura 4:19 una vista más cercana tanto del motor como del sistema de engranajes para su mejor visualización. Así de cómo encajan durante el funcionamiento estos dos engranajes helicoidales.

Figura 4:19 Motor, engranajes y eje.

5 PRUEBAS CON EL ROBOT REAL

«La robótica está empezando a cruzar esa línea que separa al movimiento absolutamente primitivo del movimiento que se asemeja al comportamiento animal o humano.».

- J. J. Abrams -

Habiendo terminado la construcción del robot móvil, así como de comprobar su correcto funcionamiento es hora de continuar con el enfoque este proyecto y comprobar la viabilidad de aplicar un controlador basado en la Teoría τ a un robot.

A continuación, se explicarán las diferentes consideraciones tomadas en cuenta a la hora de crear el código incluido en la Raspberry, que controlara el movimiento del robot, así como los pasos seguidos para crearlo y, por último, los resultados obtenidos con este proyecto. Durante la redacción de este apartado, se irán comentando fragmentos del código que se ha usado para que la Raspberry Pi ejecute en su interior y obtenga los resultados del experimento. El código completo se puede encontrar en el Apéndice 4.

5.1 Características del control

A continuación, se comentarán diferentes características del control que se han debido de tener en cuenta a la hora de construir tanto el código como de pensar en la implementación de un control en tiempo real, de manera funcional.

5.1.1 Elección del tiempo de ciclo

Para la creación el código que ejecutará la Raspberry, se ha basado en el ya implementado en Matlab, solo que, añadiendo la parte de lectura de sensores, así como la ejecución síncrona. Sin embargo, debido a las limitaciones del sensor *Time of flight* el tiempo de cada ciclo no será de 0.1 segundo, como ya se implementó en Matlab, sino que será directamente dependiente del modo de lectura del sensor. De esta manera, y debido a que tenemos que esperar cierto tiempo para recoger las medidas de este sensor, y este tiempo no es muy elevado, se ha decidido adaptar el control para que este sea su tiempo de ciclo y así leer del sensor sin problemas. La Tabla 5:1 se ha generado a partir de unos experimentos realizados con el sensor a una distancia fija de 500mm y sin movimiento, pero en el Apéndice 2, se puede apreciar otro experimento más voluminoso en muestras y variedad.

Tabla 5:1 Modos de funcionamiento del sensor GY-VL53L0XV2

Modo de funcionamiento	Tiempo entre lecturas	Error en la medida
GOOD ACCURACY	33 milisegundos	±15mm 3% a 500mm
BETTER_ACCURACY	66 milisegundos	±5mm 1% a 500mm
BEST_ACCURACY	200 milisegundos	±2mm <1% a 500mm
LONG_RANGE	33 milisegundos	No aplica
HIGH_SPEED	20 milisegundos	±25mm 5% a 500mm

La Tabla 5:1, ha sido generada a partir de la media de 1000 muestras con cada modo de funcionamiento y es aproximada, simplemente para ayudarnos a ver la precisión en cada modo, pero se ha realizado de forma estática y a una distancia fija, cosa que no será así durante los experimentos con el robot real.

Tras realizar una serie de pruebas con este sensor, se ha visto que el modo BETTER_ACCURACY, es el más óptimo en relación tiempo y calidad de medida, ya que a la hora de realizar el experimento se pueden considerar las situaciones casi ideales de una superficie blanca e interior este modo da bastante buenos resultados y no necesita de un tiempo tan largo como son 200ms para un control, y en preferible una mayor precisión al estar en movimiento y poder realizar solo una medida en ese punto. Es por esto que, como periodo para el control se ha seleccionado 66 milisegundos. A la hora de determinar el tiempo de cada ciclo se ha optado por una espera activa, por el hecho que en Python las señales POSIX no pueden ser activadas en un tiempo menor a 1 segundo.

5.1.2 Lectura e interpretación de los sensores

Como ya se ha podido ver en el apartado 4.1 se han colocado dos sensores en este robot móvil. Luego, durante la ejecución de la prueba, uno de los principales puntos críticos es la lectura de estos sensores para saber tanto la posición relativa restante para alcanzar el obstáculo como la velocidad en cada tiempo de ciclo.

En primer lugar, se leerá el sensor laser GY-VL53L0XV2 a través de I2C con ayuda de la librería ya comentada [13] y posteriormente la velocidad del motor con ayuda del encoder, midiendo el tiempo entre los pulsos del encoder y así poder calcular su velocidad. Sin embargo, para intentar estimar una posición más correcta durante el lapso de tiempo que dura el movimiento se ha usado un filtro de Kalman para estimar la posición con mayor precisión.

"Un filtro de Kalman es un algoritmo usado para estimar las variables de un sistema basándose en medidas con ruido. Lo que hace este algoritmo es calcular las diferentes probabilidades del estado del sistema, superponiéndolas posteriormente con las diferentes mediciones teniendo en cuenta su componente de ruido añadido. Es por ello que el filtro de Kalman es perfecto para usar sistemas embarcados donde los sensores, para recoger la información del entorno, tienen bastante ruido (sobre todo con sensores baratos)." [17]

Y puesto que nuestros sensores son de los más baratos del mercado, aunque con bastantes buenos resultados, se ha considerado que, esta técnica podría incluso mejorar más su precisión. Este filtro se ha aplicado sobre la posición, que es la variable sobre la cual, se ha ejecutado el control. Esta característica, es más estable que lo que sería la velocidad. Luego, se ha implementado un filtro de Kalman sobre ella, usando como predicción la velocidad.

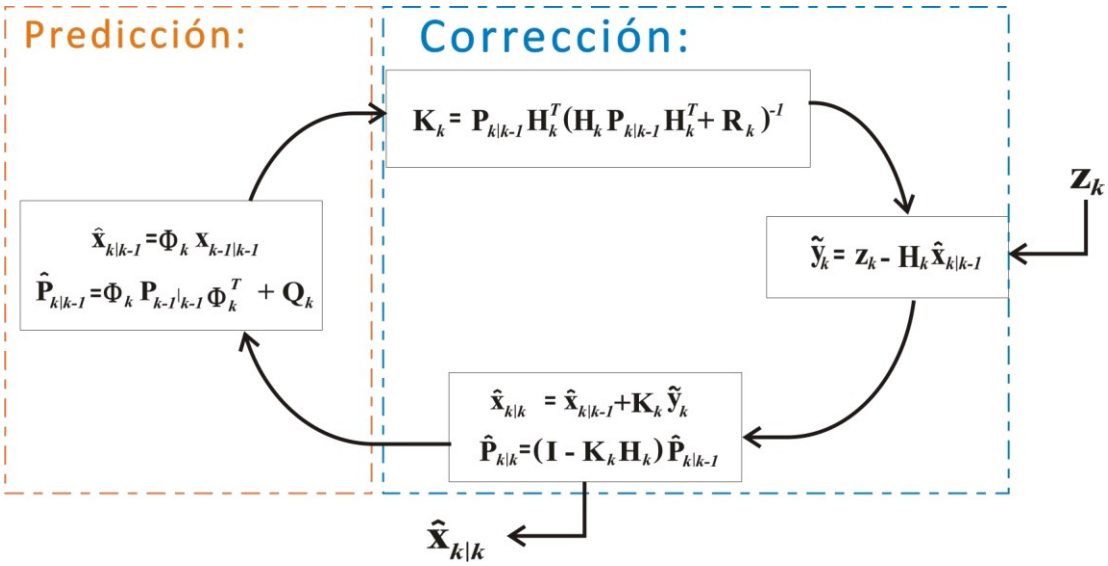


Figura 5:1 Ilustración Filtro de Kalman [18]

Siendo las ecuaciones del filtro de Kalman las que se pueden ver en la Figura 5:1. Para un sistema discreto como el nuestro y simplificando para 1 variable posteriormente, en principio quedarían unas ecuaciones de la forma:

Etapa de predicción:

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

$$P_k^- = AP_{k-1}A^T + Q$$

Etapa de actualización

$$K = P_k^- C^T [CP_k^- C^T + R]^{-1}$$

$$\hat{x}_k = \hat{x}_k^- + K(z - C\hat{x}_k^-)$$

$$P_k = (I - KC)P_k^-$$

Donde:
$A \in \mathbb{R}^{N\times N}$:= Matriz que contiene el modelo del sistema.
$\hat{x}_k^- \in \mathbb{R}^{N\times 1}$:= Estimación a priori de la medida en el instante actual.
$B \in \mathbb{R}^{N\times N}$:= Matriz que contiene el modelo de la entrada del sistema.
$u_{k-1} \in \mathbb{R}^{N\times 1}$:= Entrada del sistema en el instante k-1.
$P_k^- \in \mathbb{R}^{N\times N}$:= Estimación a priori de la Covarianza en el instante actual.

A la hora de construir estas matrices, siendo estas de 1x1 se pueden simplificar bastante, esto se vera de forma más detallada en la parte del código que refleja la aplicación de este filtro.

```
    ##########################################################
    #Filtro de Kalman sobre la posición
       if(vel_act==0):
              FKalman_activo=0;

       #Si continua activo actuaremos de manera normal,
       #guardando el resultado de la posición en distancia
       #para que el código funcione de la misma forma si se
       #ejecuta el filtro como si no

       if(FKalman_activo==1):
              ##########################
              #Ganancia de Kalman
              kalman_K= kalman_cov* 1/(kalman_cov + Q_laser);
              #Estimación a posteriori
              distance=pos_kalman_pre+kalman_K*(distance-
              pos_kalman_pre);
              #Corrección de la covarianza
              kalman_cov = (1 - kalman_K) * kalman_cov;
              #Estimamos la siguiente posición,
              #Actual más la velocidad por el tiempo
              pos_kalman_pre=distance+vel_act*delta_time
              +Q_laser*Q_encoder;

       if(FKalman_activo==2):
              #Primera iteración, no hay predicción
              #Se colocan los valores iniciales y la predicción
              #para la siguiente iteración
              #Estimamos la siguiente posición actual más la
              #velocidad por el tiempo
              #Estimación del valor
              pos_kalman_pre=distance+vel_act*delta_time;
              #Estimación de la covarianza
              kalman_cov=Q_laser*Q_encoder;
              #Por último se devalúa la variable FKalman_activo,
              #para indicar que la siguiente iteración, se
              #realizará de manera normal.
              FKalman_activo=1;
```

5.1.3 Funcionamiento del robot y explicación del proceso.

Como ya se ha dicho anteriormente, al comienzo del bucle se leerá el sensor GY-VL53L0XV2, que es el factor limitante en función del tiempo y a continuación se leerá el encoder, que según el tiempo entre los pulsos nos indicará una velocidad media al realizar varias medidas. Una vez realizadas las medidas, se pasará a hacer un filtro de Kalman para intentar estimar la posición real del vehículo, a modo de mejorar lo máximo posible la entrada del control. Con la estimación de la velocidad y la posición se realizará la ejecución del controlador, en primer lugar, determinando la posición deseada según la Teoría τ y seguido la ejecución del controlador PID que determinará el voltaje aplicado al motor. Este es un breve resumen del procedimiento que seguirá el robot, pero se puede ver la versión más extensa con el código comentado en el Apéndice 4. Aun así, se verá de forma más detenida a continuación.

En cada iteración, lo primero que hacemos es guardar el tiempo de inicio para saber el tiempo total de la iteración, ya que es una espera active y necesitamos este dato. Posteriormente, leemos el sensor ToF, ya que este era el factor limitante.

```
#####################################
#Empieza el bucle
start_time = time.time();
while(fin_control==0):
    start_cicle_time = time.time()
    #####################################
    distance = tof.get_distance()
    #Después, procedemos a lo demás, que es menos sensible al tiempo
```

El siguiente Dato a medir es la velocidad dada por el encoder. Sin embargo, para evitar que el encoder no devuelva ningún valor, debido a que el motor no esté girando, es decir, velocidades muy bajas, se ha optado por eliminar este paso, si la diferencia entre las distancias actual y anteriores es menor a 4, número que se considera como el error máximo del sensor estando parado. Aunque durante el movimiento solo se debe alcanzar esta condición al final, durante las pruebas puede llegar a ocurrir que, en algún punto crítico se quedase bloqueado durante varios segundos o de forma infinita. Es por esta razón por la que se le han añadido condiciones de salida gracia a las señales POSIX, en este caso sigalarm.

El fragmento de código siguiente, comprueba que se active la señal del encoder y calcula el tiempo que tarda en Volver a activarse. Esta acción es realizada varias veces para evitar errores, aun así, se hacen varias operaciones para evitar posibles medidas erróneas. También destacar que se irán recogiendo todas las velocidades hasta que se inicia el movimiento de frenado, y así, conseguir una medida lo más precisa posible de la velocidad al inicio del movimiento.

A continuación, iría el filtro de Kalman, peor como ya ha sido comentado en el apartado anterior se pasará a la ejecución del control.

```
    if(abs(distance-distance_ans)>4):
        #si es mayor, miramos la velocidad con encoder
        ############################
        #Activamos sigalarm a 1 seg que es el tiempo mínimo permitido
        #en caso de que ocurriera algún error
        signal.alarm(1);
        ############################
        #Actuamos de forma normal dentro de un try por si ocurre
        #sigalarm, y tenemos que salir de él.
```

```
        try:
            #se realizarán 2 medidas y a continuación una estimación
            #para ver cuál de las dos es más fiable
            #Se podrían realizar más ya que no hay falta de tiempo
            #pero con dos medidas se ha visto que el error es mínimo
            #primera medida
            t_ini_vel=time.time();
            GPIO.wait_for_edge(Encoder_1, GPIO.RISING)
            GPIO.wait_for_edge(Encoder_1, GPIO.RISING)
            t_vel1=time.time()-t_ini_vel;
            #segunda medida
            t_ini_vel=time.time();
            GPIO.wait_for_edge(Encoder_1, GPIO.RISING)
            GPIO.wait_for_edge(Encoder_1, GPIO.RISING)
            t_vel2=(time.time()-t_ini_vel);

            #Para calcular la velocidad actual, hay que multiplicar
            #el tiempo por las diferentes características del robot
            velocidad_actual1=0.19074/t_vel1;
            velocidad_actual2=0.19074/t_vel2;
            #Nos quedamos con la medida adecuada:
            dif=abs(velocidad_actual1-velocidad_actual2)
            if(dif>0.5 ):
                vel_act=min(velocidad_actual2,velocidad_actual1);
            else:
                #media
                vel_act=(velocidad_actual1+velocidad_actual2)/2
        except  Time_Exception:
            #Si se ha tardado más de 1 seg en medir se considera 0
            vel_act=0;
        ############################
        #En el caso de que ocurra todo satisfactoriamente
        #se devuelve a 0 sigalarm
        signal.alarm(0);
    else:
        #En el caso que la diferencia entre la medida de la
        #distancia, no sea suficiente se considera 0, pero no
        #afectará a los cálculos posteriores.
        vel_act=0;

    ##########################################
    #Para conseguir la velocidad al inicio del movimiento lo mas
    #precisa posible, se realizará una media de las velocidades
    #durante el recorrido previo al inicio
    vel_media_inicio+=vel_act;
    contador_vel+=1;
```

Ahora, llegamos a la parte importante del código, en esta parte se ejecutará un control discreto de manera simple, en la que se actualizarán las variables y se comprobarán los errores de las mismas para introducirlas en el PID y así, hacer que el robot se adecue a la trayectoria. En primer lugar, se comprueba si la condición para activar el control se ha cumplido si este está detenido, y en caso contrario si ha excedido el tiempo que debía de durar el control y devuelve un fallo. Una vez que el control se está ejecutando funciona de manera similar al código comentado en el apartado 2, calcula la posición para el estado siguiente y calcula el error de la posición para darle valores a la salida del motor según adecue el PID.

```
    ############################################################
    #tras obtener las medidas, pasamos al control
    if (control_activo==1 and fin_control==0):
        #Si el control esta activo
        #Lo primero que debemos hacer es calcular el porcentaje de
        #tiempo transcurrido de la maniobra para el siguiente punto
        Tm=(time.time()-t_control_ini+delta_time)/Tiempo_maniobra;
        #calculamos el siguiente TM (+delta_time)

        if(Tm>1):
            #Si supera a 1 es que el tiempo ha terminado y,
            #por lo tanto, el control ha terminado también
            pwm_motor.start(0)
            break;

        ##################################
        #Ejecución del algoritmo de control
        #Cálculos de la teoría de Tau ya explicados en el punto 2
        #y en el 3 con las simulaciones
        #Valores nominales deseados
        pos_nominal=-(1-Tm)**(2/K);
        vel_nominal=(2/K)*(1-Tm)**((2/K)-1);
        #Valores deseados
        pos_deseada=-pos_nominal*D_objeto;
        vel_deseada=vel_nominal*(D_objeto/Tiempo_maniobra)
        ##################################
        #Comenzamos con el PID
        ##################################
        #Calculamos los errores
        error_vel=vel_deseada-vel_act;
        error_pos=pos_deseada-(distance-50)
        ##################################
        #Calculamos los valores K I y D del control
        sal_kp=-(P*error_pos);
        sal_ki=-(I*error_pos*delta_time);
        sal_kd=D*(error_pos-error_pos_ant)/delta_time
        Salida_control=sal_kp+sal_ki+sal_kd;
        ##################################
        #Guardamos los valores de los errores
        error_vel_ant=error_vel;
        error_pos_ant=error_pos;
        ##################################
        ##################################
        #Guardamos los datos resultantes para su posterior análisis
        PID_sal.append(Salida_control)
        PID_K.append(sal_kp)
        PID_I.append(sal_ki)
        PID_D.append(sal_kd)
        pos_d.append(pos_deseada)
        vel_d.append(vel_deseada)
        error_pos_g.append(error_pos)
        error_vel_g.append(error_vel)

        ##################################
```

```
        ################################
        #Actuación del control en el motor
        if(Salida_control>0):
            #Sentido positivo
            Giro_Favor_Reloj_Motor()
            if(Salida_control>100):
                Salida_control=100;
        else:
            Giro_Contra_Reloj_Motor()
            #Sentido negativo
            #Aunque no deba activarse nunca
            #En la práctica es más aconsejable limitar el control
            #para que solo avance
            Salida_control=abs(Salida_control)
            if(Salida_control>100):
                Salida_control=100;

        ################################
        #cambiamos el valor del PWM
        pwm_motor.start(int(Salida_control))
        ################################
        #Fin control

    ###############################
    #Condición de activación control
    #Comenzaría en 550-50=500 aproximadamente
    if(distance<550 and control_activo==0):
        #Cuando se cumpla se activará el control para que,
        #en la siguiente iteración actúe
        control_activo=1;

        ###############################
        #Por lo tanto, se definirán las variables que necesita
        #el control al inicio
        #Distancia a la que se desea detenerse, se coloca
        #un poco antes para evitar posibles choques por fallo
        D_objeto=distance-50;
        #Se calcula la velocidad inicial haciendo la media
        vel_ini=vel_media_inicio/contador_vel;
        #Se calcula tau sub zero
        tau_zero=-D_objeto/vel_ini;
        #Se calcula el tiempo total que durará la maniobra
        Tiempo_maniobra=-(2*tau_zero)/K; #tiempo de maniobra
        #tiempo en el que se inicia el control para
        #saber cuándo acaba la maniobra
        t_control_ini=time.time();

        ###############################
        ###############################
        #Se activan las luces freno
        GPIO.output(luz_freno1,True)
        GPIO.output(luz_freno2,True)
```

Para terminar con la ejecución del bucle, se guardan los valores necesarios para la siguiente iteración, y se pasa a la espera activa para el siguiente ciclo de ejecución, en la cual se comprobará cuando se ha alcanzado el tiempo ene l que el bucle deberá iniciar de nuevo y se comprobara si el tiempo ha excedido al que debía y devolverá un error, deteniendo el movimiento.

```
    pre_vel=vel_act;
    pre_pos=distance-50
    distance_ans=distance
    ################
    #Se guardan también datos para su representación
    pos_g.append(pre_pos)
    vel_g.append(vel_act)
    ##############################

    while(time.time()-start_cicle_time<tiempo_ciclo/1000000.00):
        pass
    delta_time=time.time()-start_cicle_time;
    #Se comprueba que el tiempo de ejecución se haya cumplido
    if(delta_time>0.07):
        #En caso contrario se tratará como si hubiera
        #ocurrido un fallo, deteniéndolo todo
        pwm_motor.start(0)
        break;

##############################
#Fin del bucle
##############################
```

5.1.4 Recogida de Datos

Una vez haya terminado de ejecutarse el control y el robot se haya detenido, es hora de apreciar los resultados de forma numérica, para esto durante la ejecución del mismo se han ido guardando las variables que se han creído necesarias para apreciar con más detalle si el robot ha seguido la trayectoria correcta. Estos datos recogidos, se guardan en un documento de texto dentro de la Raspberry y pueden ser observados o enviados a través de una conexión wifi, y en caso de poder acceder de forma remota al dispositivo los datos serán pintados al terminar el control. Estos datos pueden ser recogidos de distintas formas debido a la amplia variedad que nos ofrece la Raspberry, pero en este caso se ha optado por una comunicación simple y segura, SSH o Secure Shell.

En el siguiente código, se muestra un ejemplo de cómo se guardarían los datos en el fichero:

```
##############################
#A continuación, guardamos los datos deseados
#En nuestro caso se dejará un dato de ejemplo
#Este dato se guardará de manera que se pueda copiar y
#pegar en la ventana de Matlab para su análisis sin tener que
#cambiar nada, para ello se guardará en forma de vector
file.write("%Pos(medida) Num datos:"+str(len(pos_g))+"\n")
file.write("Valores_Pos=[")
primera=1;
for l in pos_g:
    if(primera!=1):
        file.write(";")
```

```
    else:
        primera=0;
    file.write(str(l))
file.write("];"+ '\n')

##############################
#Por último, cerramos el fichero para evitar errores
file.close();
```

Aun con la recogida de datos en un fichero, también se ha implementado mediante la ayuda de Python y a la librería matplotlib, una muestra de los resultados al finalizar la prueba. Este código de ejemplo, quedaría de la siguiente forma:

```
###################################
#GRÁFICAS
#Esta parte del código está destinada para dibujar las gráficas
#Depende de lo que quiera el usuario
#A continuación, se deja un ejemplo como ene l caso anterior
plt.figure(1);
#En nuestro caso, vamos a pintar la posición guardada desde el inicio
plt.plot(tiempo_g,pos_g);
title('Posición relativa del Robot')
plt.xlabel('Tiempo / ms')
plt.ylabel('Distancia / mm')
plt.grid();
plt.show();

#Esta última función dejara el programa suspendido hasta que se cierren
las gráficas
```

5.2 Resultados

Tras la depuración de código y ajustar el control PID un poco, se realizaron varias pruebas y se obtuvieron resultados diferentes. Aunque se podían intuir los diferentes resultados con las simulaciones en Matlab los resultados reales varían un poco de los predichos.

Comenzando por una K=0.4 el resultado es que el movimiento es demasiado suave en la fase final, lo que provoca que el robot o se quede quieto o en el caso de continuar el movimiento, este no sea lo suficientemente lento como para adaptarse al punto final. Estos datos se pueden apreciar en la Figura 5:2. En esta grafica se puede ver como el robot supera el punto límite, aunque muy poco a poco, además no cumple con el tiempo de maniobra calculado, siendo este de casi 6 segundos. En el caso de ser otro tipo de robot esta trayectoria tan lenta en la última fase si puede ser útil, pero al ser un robot móvil no necesita tal precisión además de que no puede lograrla por motivos mecánicos. Sin embargo, como primera prueba sirve para comprobar que el robot se adapta con precisión a la trayectoria pedida y ver que se va en buen camino.

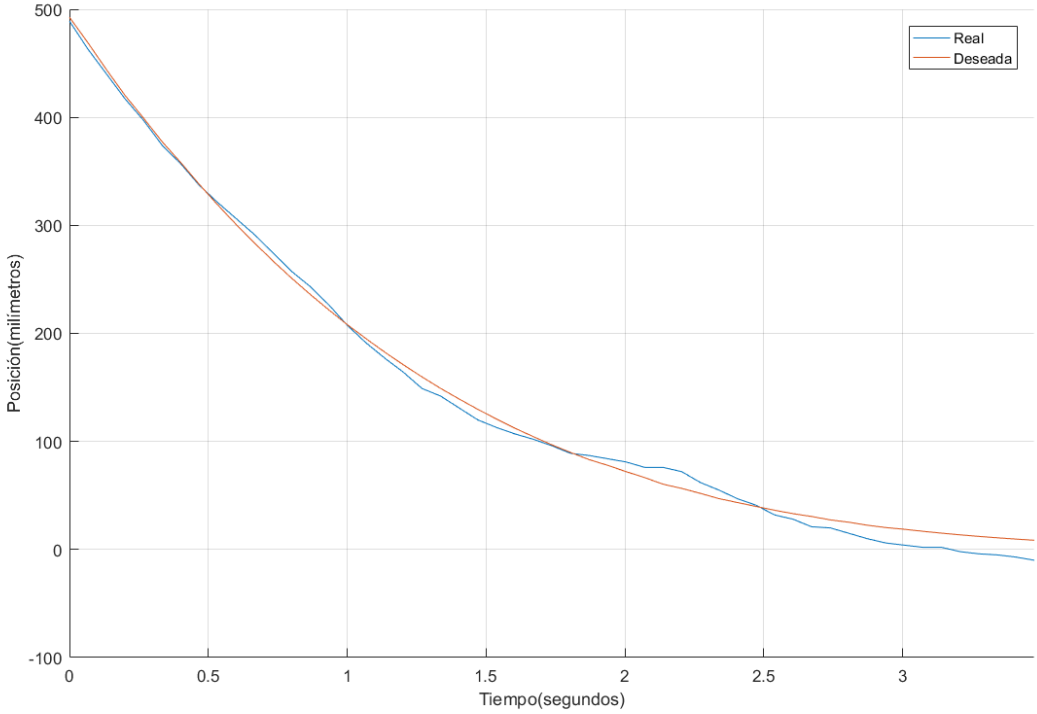


Figura 5:2 Resultados de la prueba para K=0.4, trayectoria del robot real recortada.

Una vez descartadas la K anterior como las inferiores pasaremos a comprobar una K >0.6 que es el punto limite que fijamos cuando hicimos la simulación. Pasaremos a realizar una simulación con K=0.8 siendo esta K bastante elevada para ver con simpleza los errores de una K más alta.

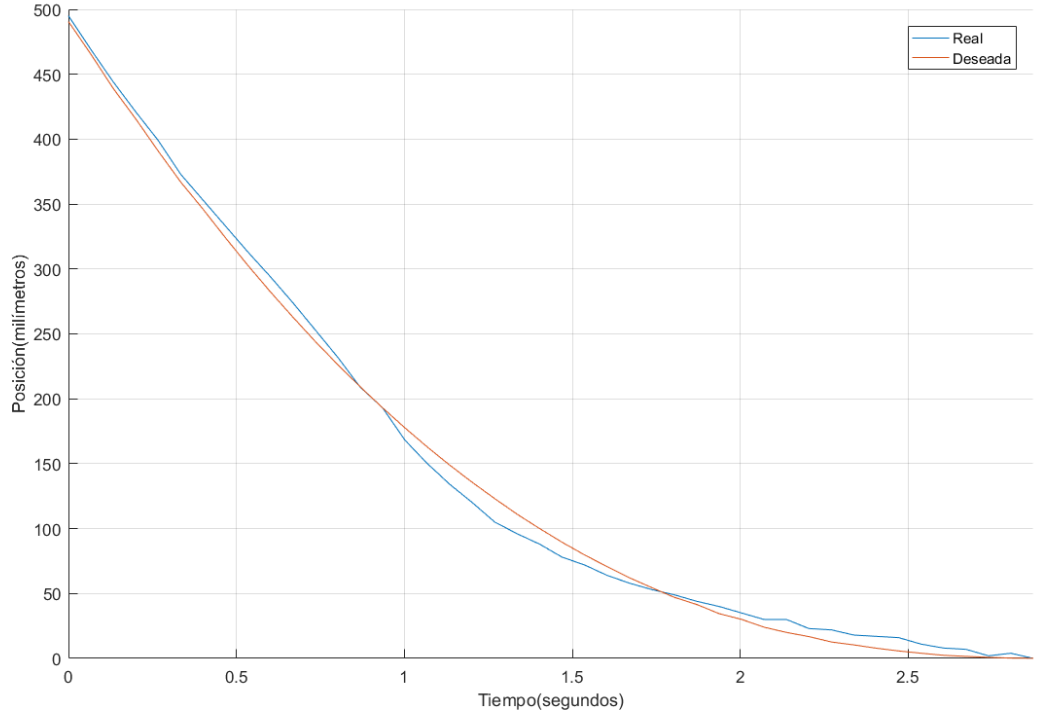


Figura 5:3 Resultados de la prueba para K=0.8, trayectoria del robot real recortada.

En este caso el tiempo de maniobra se cumple, siendo este unas décimas inferiores a 3 segundos y el movimiento acaba por límite de tiempo y no por límite de posición como en el caso anterior. Esto se puede apreciar en la Figura 5:3. Por otra parte, este movimiento no es el que deseamos, ya que también contiene algunos errores, que en el caso de querer un movimiento suave y que se acerque con suavidad al final, son significativos. La curva demandada por el controlador no se asemeja a un movimiento más visto en la naturaleza por aves rapaces que intentan llegar a su presa con una velocidad elevada y así poder cazarla, sin embargo, en nuestro caso esa última desaceleración agresiva no nos conviene y preferimos que sea un poco más suave para que el robot llegue al punto deseado con el mínimo error posible.

Al final tras varias pruebas más se ha visto que la K más óptima para este tipo de movimiento oscila entre 0.5 y 0.7, ya que en las simulaciones no se tuvo en cuenta que el robot no pudiera moverse a velocidades muy lentas o en su defecto velocidades concretas, que en nuestro caso debemos controlarlas a través de PWM que no contiene el rango completo. Que no pueda moverse a velocidades concretas impide un acercamiento muy lento (K<0.5), pero a su favor el rozamiento del suelo como del propio motor le permite tener una frenada mayor y poder manejar K superiores a las simuladas.

Una vez analizados los resultados anteriores, se mostrarán las gráficas generadas para K=0.6 y se comentarán los resultados obtenidos que se podrán ver tanto en la Figura 5:4, como en las siguientes.

(La línea discontinua magenta indica el inicio del control.)

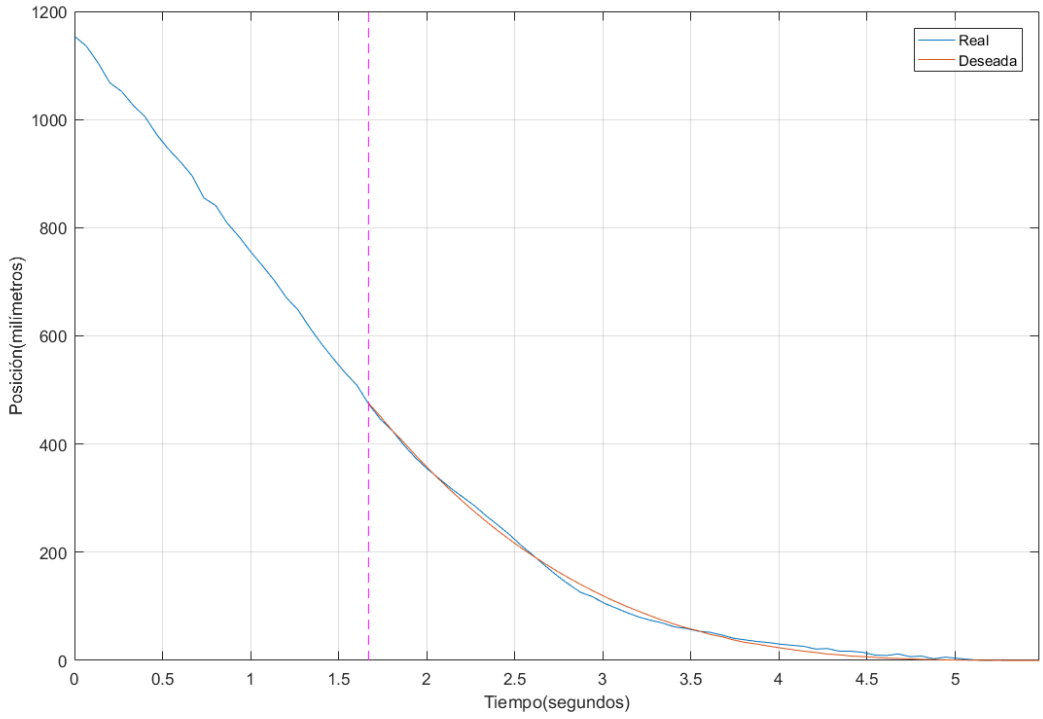


Figura 5:4 Resultados de la prueba para K=0.6, trayectoria del robot real.

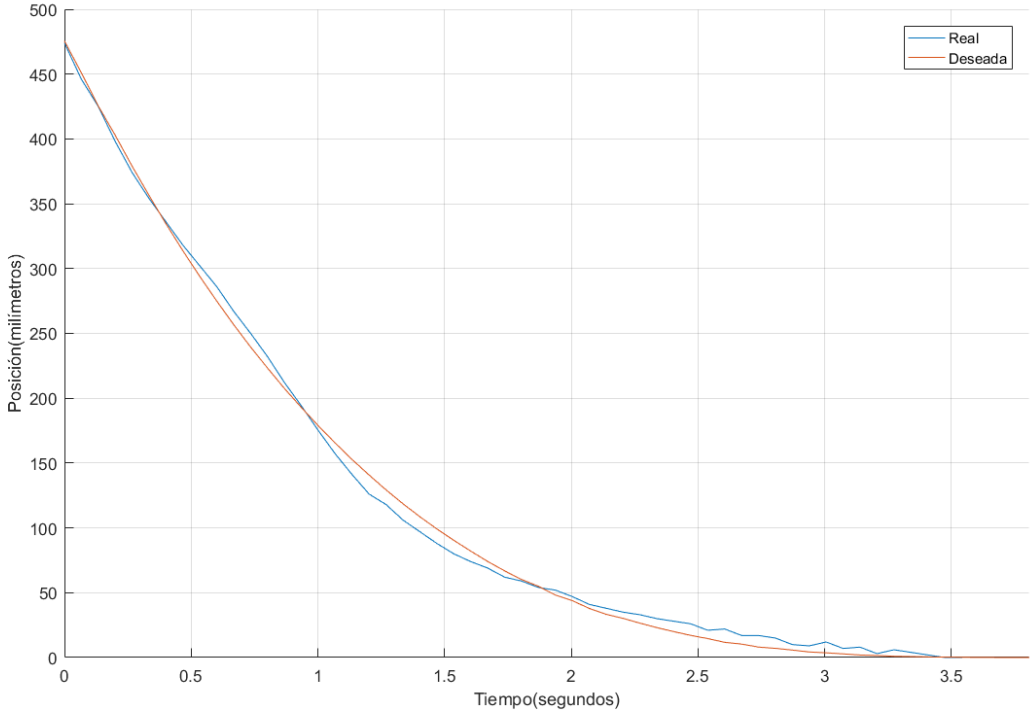


Figura 5:5 Resultados de la prueba para K=0.6, trayectoria del robot real recortada.

Tras analizar los resultados para la K=0.6, en primer lugar, podemos ver en la Figura 5:5, cómo evoluciona la posición y llega al punto objetivo con mucha precisión, además de seguir la trayectoria de manera bastante fiel. Sin embargo, si pasados de la posición a la gráfica de la velocidad, la cual no está siendo controlada de manera tan directa, podemos ver que la gráfica se asemeja como se ha visto de las simulaciones en Matlab, pero no es tan perfecta.

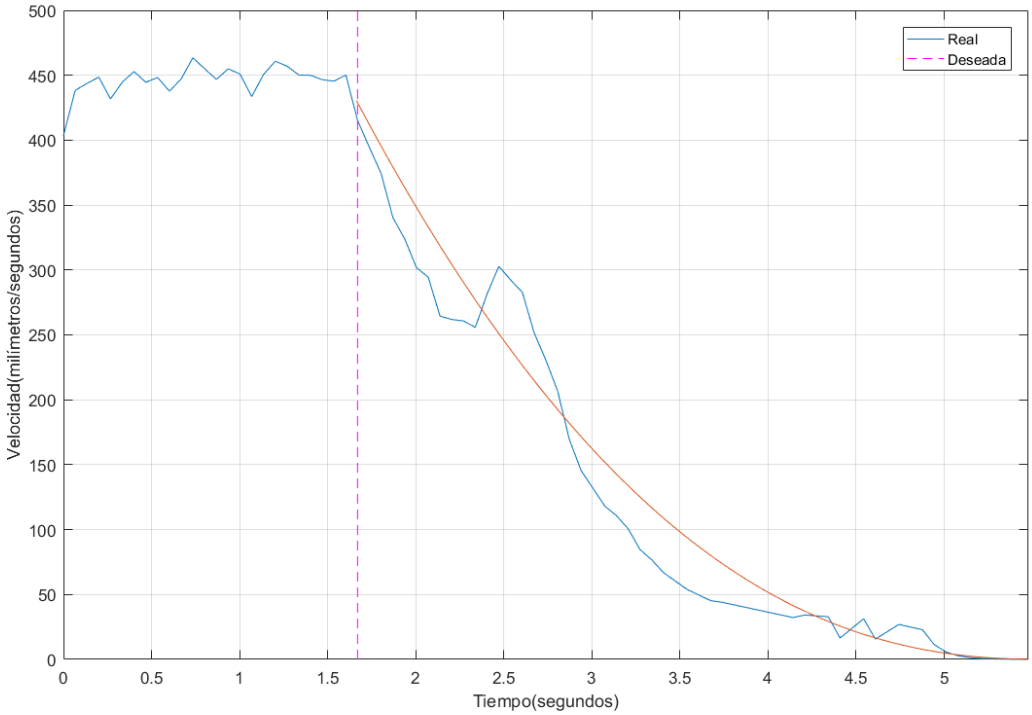


Figura 5:6 Resultados de la prueba para K=0.6, velocidad del robot real.

Este error en la gráfica de la velocidad, que se muestra en la Figura 5:6, puede que sea debido a fallos en las medidas del encoder, así como en velocidades muy bajas que, no se pueden medir con el encoder, debido a que el pulso tardaría en llegar más de lo que dura el ciclo del control. Así mismo, a esta variable no se le está aplicando ninguna corrección, que comparada con la posición si lleva un tratamiento, de modo que con saber que la velocidad se asemeja bastante a la pedida debería bastar, ya que en la posición se puede ver como el movimiento es muy similar.

6 CONCLUSIONES Y TRABAJO FUTURO

«Cualquiera que deje de aprender es viejo, ya sea a los veinte u ochenta. Cualquier persona que sigue aprendiendo se mantiene joven.»

- Henry Ford -

Durante la evolución de este Proyecto, se han podido ver diferentes cambios y formas de sortear los diferentes impedimentos que ha planteado. Ahora, una vez finalizado, es hora de plantear que ha proporcionado la realización del mismo y como se podría seguir indagando sobre el tema.

6.1 Conclusiones

A través de la realización de este estudio, se ha podido ver las ventajas de implementar un control bio-inspirado respecto de un control tradicional. De esta manera de han sacado diversas conclusiones tras finalizar el proyecto:

- Se ha podido ver como el control ha propuesto una trayectoria en la cual ha acompañado al sistema y no le ha impuesto ejecutar un gran cambio en su movimiento para obligarle a cumplir sus expectativas.
- Tras conseguir ver los resultados de este tipo de control, podemos sacar ciertas conclusiones. Una de estas conclusiones es: que un control basado en una habilidad cognitiva desarrollada tras años de evolución, proporciona una salida de actuación bastante más lineal y sin cambios bruscos.
- Se ha podido ver, que el estudio de la Teoría τ ha servido para desarrollar la aproximación a un obstáculo de manera que el sistema se asegure de que nunca va a impactar con el objeto, si se sigue una buena ejecución de la teoría.

- Esta teoría, revela las claves para saber en todo momento si el sistema impactará o no con el objeto y en tal caso de que forma lo hará, y de qué manera hay que actuar para evitar tales condiciones no deseadas. Todo esto, mientras el control pueda sobrellevar el sistema. Luego otra conclusión a sacar, es que debemos conocer ante todo el sistema para poder aplicar esta teoría.

En resumen y como conclusión final, podemos decir que la implementación de un control basado en una habilidad innata de las aves, ayudará en gran medida a la detención de los Drones, de una manera segura y simple como se ha podido comprobar con el sistema de robot móvil en este proyecto.

6.2 Trabajo futuro

Como trabajo futuro, como ya se ha comentado durante la ejecución de este proyecto, habría un punto principal que sería llevar la Teoría τ y la implementación de este control, a un mundo de la aeronáutica, con la finalidad de poder implementarlo en un robot tipo dron.

En primer lugar, habría que rediseñar el control usado, ya que, aunque en este proyecto solo se use la dimensión útil de esta teoría, y la otra se considere como un movimiento rectilíneo uniformemente desacelerado. Habría que implementar las 3 coordenadas a la hora de poder controlar un dron. Por supuesto, se debe que realizar un estudio para ver qué movimiento de los dados por la Teoría τ es más adecuado a la hora de utilizarlo en un dron, es decir, que K sería la más idónea en cada momento, ya que, dependiendo de la distancia, velocidad, viento u inercia del modelo, podrían variar de forma que el control llegara a ser inestable para un Dron.

Una vez establecidos los parámetros correctos del modelo del dron y del control a usar, habría que realizar las supuestas simulaciones, que bastaría con tomar como base las realizadas en este proyecto, con la diferencia de que el modelo cambiaría a uno que simularía un Dron. Por último, y una vez generado un control útil y que genere una correcta salida tras su simulación se implementaría en un dron real con sus correspondientes medidas de seguridad, tanto para el Dron como para el piloto.

Aparte del trabajo de la implementación del control en un Dron, se podrían realizar algunos trabajos para la mejora del propio control en el robot móvil.

- Se podría realizar una mejora que implemente un algoritmo, que detecte cuando deba ejecutar el inicio de la maniobra según le indiquemos la desaceleración máxima u otros parámetros a elegir, y este algoritmo decida cuando empezarla para cumplir con nuestras exigencias.
- Otra mejora similar y quizás más útil seria fijar una distancia a la que empezar la maniobra y que el robot a partir de su experiencia sepa que K elegir para que su parada sea la más eficiente en cuanto a términos de combustible o en nuestro caso batería.

APÉNDICE 1: ENCODER

Para este proyecto ha sido de vital importancia el conocimiento del funcionamiento de un encoder, así que en este apartado se tratará de explicar el funcionamiento de estos y más en concreto del que se usará en este proyecto.

En primer lugar, y a partir de la información proporcionada por [19], un encoder es un dispositivo de detección que proporciona una respuesta.

Los Encoders convierten el movimiento en una señal eléctrica que puede ser leída por algún tipo de dispositivo de control en un sistema de control de movimiento. El encoder envía una señal de respuesta que puede ser utilizado para determinar la posición, contar, velocidad o dirección. Un dispositivo de control puede usar esta información para enviar un comando para una función particular. En nuestro caso será un pulso eléctrico enviado hacia el dispositivo de control que será la Raspberry, pero sus aplicaciones son muy diversas en nuestro mundo.

Para poder describir cómo funciona más concretamente un encoder haría falta referirnos al tipo de encoder en concreto, ya que hay muchos tipos con diferentes mecanismos y finalidades. En [20], podemos ver los tipos de encoder.

Uno de los más comunes por su simpleza es el **Encoder óptico,** que está compuesto por una fuente emisora de luz, un disco giratorio y un detector de luz "foto detector".

El disco está montado sobre un eje giratorio y dispone de secciones opacas y transparentes sobre la cara del disco. La luz que emite la fuente es recibida por el fotodetector o interrumpida por el patrón de secciones opacas produciendo como resultado señales de pulso. Hasta aquí terminaría lo conocido como encoder, pero para saber el punto exacto suele ser acompañado de algún microprocesador que indique la posición exacta del encoder en caso de necesitarse.

Por otro lado, tenemos el **Encoder lineal**. Este sensor cuenta con una escala graduada para determinar su posición. Los sensores en el encoder leen la escala para después convertir su posición codificada en una señal digital que puede ser interpretada por un controlador de movimiento electrónico. La diferencia más reseñable respecto al anterior es que daría directamente la posición actual y no solamente un pulso.

Los encoders lineales pueden ser absolutos o incrementales y existen diferentes tipos de encoders lineales según la tecnología que se usa en su mecanismo.

Pero, sin embargo, aunque un encoder lineal nos indique su posición, si queremos mayor precisión, deberemos usar un **Encoder absoluto**, ya que ofrecen un código único para cada posición y se dividen en dos grupos: los encoders de un solo giro y los encoders absolutos de giro múltiple. Su tamaño es pequeño, lo que permite una integración más simple.

Los encoders absolutos se aplican en motores eléctricos de corriente directa y sin escobillas, en sectores específicos como la maquinaria sanitaria.

Por último, nos quedarían el **Encoder incremental**, que se trata de un tipo de encoder óptico, y que determina el ángulo de posición a raíz de realizar cuentas incrementales, donde cada posición es completamente única, y el **Encoder de cuadratura**, que es un tipo de encoder rotativo incremental el cual tiene la capacidad para indicar la posición, dirección y velocidad del movimiento.

En relación a sus aplicaciones, podemos encontrarlos en muchos productos eléctricos de consumo y en una infinidad de aplicaciones comerciales. Entre sus principales ventajas destaca su flexibilidad, sencillez y durabilidad.

Ahora, centrándonos un poco más en el encoder usado en nuestro proyecto, el cual es un Encoder de cuadratura con sensores de efecto Hall. Para explicar el funcionamiento de este sensor me basaré en la información en [21].

Este tipo de encoder que funciona mediante el principio del efecto Hall, generan 2 señales de onda cuadrada con un desfasamiento de 90°. Es por esto que se conocen como encoders en 'cuadratura', ya que ocupan un cuadrante del círculo de 360°.

Antes de hablar del sensor por completo, hará falta conocer un poco más el tipo de sensor, los sensores de efecto Hall son transductores que generan un voltaje de salida en respuesta a la presencia de un campo magnético variable. Una foto que puede aclarar más su funcionamiento dentro del encoder:

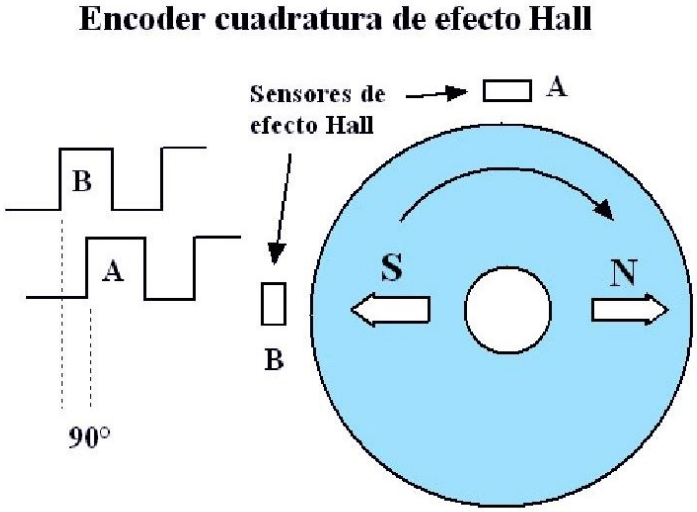


Figura Apéndice 1:1 Esquema Encoder son sensor de Efecto Hall.

Para aplicaciones de cálculo de velocidad y posición, los sensores son colocados con una separación de 90°, con respecto al círculo completo de giro del eje del motor. Un imán de ferrita induce en los sensores las señales A y B mostradas en la figura. Ambas señales están desfasadas 90°, debido a la posición de los sensores con respecto al círculo de giro.

Generalmente los encoder de efecto Hall usan imanes de ferrita, los cuales pueden contar con un gran número de polos, esto hace que el encoder genere una gran cantidad de pulsos por vueltas, determinando de forma más precisa su posición.

Figura Apéndice 1:2 Fotos del motor usado descargadas de [9]

En las fotos anteriores se puede apreciar la ferrita y el sensor de efecto Hall del encoder del motor usado.

Por último, en este apéndice se hablará de cómo conseguir los datos del encoder, aunque de esto también se ha hablado en el documento principal se tratará de forma breve aquí.

En primer lugar, el sentido de giro del motor es fácil de determinar al tener dos señales desfasadas 90 grados, bastaría con ver cuál de las dos señales llega antes, para visualizar este fenómeno es aconsejable usar un osciloscopio.

En segundo la velocidad del motor, para este caso haría falta primero saber si el motor tiene alguna caja de cambios y su relación de vueltas y además el número de polos de la ferrita. Multiplicando estos dos valores hallaremos el número de pulsos totales que enviará el encoder en 1 vuelta completa y a partir de ahí solo habrá que calcular cuánto tiempo ha necesitado para alcanzar el número de vueltas. Esta sería la base para saber tanto los rpm del motor como la velocidad en nuestro caso, para la que habrá que saber también el tamaño de la rueda.

APÉNDICE 2: COMPARACIÓN DE LOS SENSORES DE DISTANCIA

En este apartado se compararán los dos tipos de sensores de distancia más comunes que son ultrasonidos de sensores lidar.

Los sensores de ultrasonidos según [22] son detectores de proximidad que trabajan libres de roces mecánicos y que detectan objetos a distancias que van desde pocos centímetros hasta varios metros. Su funcionamiento es simple, el sensor emite un sonido y mide el tiempo que la señal tarda en regresar. El sonido, se refleja en un objeto, el sensor recibe el eco producido y lo convierte en señales eléctricas, las cuales son elaboradas en el aparato de valoración. Estos sensores trabajan solamente donde tenemos presencia de aire (no pueden trabajar en el vacío, necesitan medio de propagación), y pueden detectar objetos con diferentes formas, diferentes colores, superficies y de diferentes materiales. Los materiales pueden ser sólidos, líquidos o polvorientos, sin embargo, han de ser reflectores de sonido. Los sensores trabajan según el tiempo de transcurso del eco, es decir, se valora la distancia temporal entre el impulso de emisión y el impulso del eco.

Por otro lado [23], nos encontramos con los sensores lidar (Light Detection And Ranging) que ocupan una posición relevante en el mundo de la robótica móvil. Su campo abarca desde los simples sensores de rango que miden distancia en una única dirección, hasta elementos que capturan información 3D precisa en una gran área de trabajo, tanto vertical como horizontal. Sus principales cualidades son la buena resolución en ángulo y distancia y la elevada tasa de muestreo. La resolución en distancia tiene la característica de permanecer constante en todo el recorrido, a diferencia de lo que ocurre con otros métodos como la triangulación o la luz estructurada.

Su principio de funcionamiento es conocido como Time of Flight (TOF) y se basa en estimar la distancia a la que se encuentra un objeto mediante el tiempo transcurrido entre la emisión y la recepción del haz de luz, suponiendo constante y conocida la velocidad de la luz en el aire. Es el mismo método que emplean los sensores de ultrasonido, pero con otra tipología de onda. Debido a la compleja tarea de medir incrementos temporales con precisión se usa luz modulada.

Una vez definidos los dos tipos de sensores, nos centraremos en el estudio realizado en [24] sobre la comparación algunos sensores de estos tipos de bajo coste.

Los sensores a analizar serán principalmente el sensor HC-SR04 de ultrasonidos y el VL53L0X que son los dos en los que he pensado a la hora de iniciar este proyecto, aunque en la página nombrada en la referencia usa alguno más.

Figura Apéndice 2:1 Sensor de ultrasonidos HC-SR04.

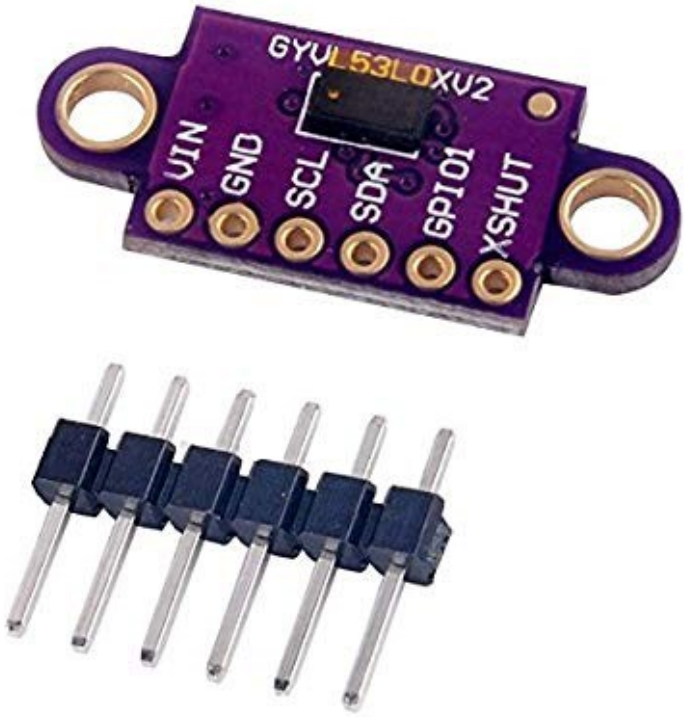


Figura Apéndice 2:2 Sensor laser VL53L0X.

En primer lugar, un breve resumen de las principales características de cada sensor:

Sensor HC-SR04

Captura de ultrasonido Medida de distancia Obstáculo El HC-SR04 es un sensor analógico que requiere dos cables. El primero (TRIG - Trigger) genera un haz de ultrasonido. El segundo se activa tan pronto como se detecta el haz devuelto. No es necesario volver a desarrollar la rueda, hay varias bibliotecas de Arduino y Python para integrar el HC-SR04 en sus proyectos. Se puede comprar por menos de un euro.

Como dato adicional, hay un montón de librerías de Arduino en las que se habla de este sensor.

Características de HC-SR04

- Rango de medición de distancia: 2 cm a 450 cm (4,5 m)
- Precisión de la medida: 0.3cm
- Voltaje de alimentación: 5V
- Salida digital: PWM
- Peso: 9g
- Dimensiones: 45 mm x 20 mm x 18 mm.

Sensor VL53L0X

El VL53L0X es un sensor que permite realizar mediciones de distancia al medir el tiempo de vuelo de un láser. La medida de la distancia es numérica. Se puede recuperar a través del bus I2C. Mucho más compacto y preciso que el HC-SR04, este sensor será mucho más fácil de integrar en proyectos robóticos, RC, drones ...

Este sensor se puede encontrar también a un precio bastante razonable de unos 2 ó 3 euros, sim embargo no es tan popular a la hora de encontrar librerías.

El VL53L1X puede alcanzar los 400cm. Pero sin embargo el VL6180X está dedicado a la medición de precisión por debajo de 10 cm. Su precio es similar, unos 3 €.

Características del VL53L0X

- Rango de medición de distancia: hasta 200cm (2m)
- Bus I2C: dirección 0x29
- Longitud de onda del rayo láser: 940nm
- Tamaño de la tarjeta (excluyendo el conector): 25 mm x 13 mm (depende del fabricante)
- Rango de potencia: 2.8V a 5.5V

Ahora pasemos a los resultados de los experimentos, para el sensor de ultrasonidos **HC-SR04** encontramos que:

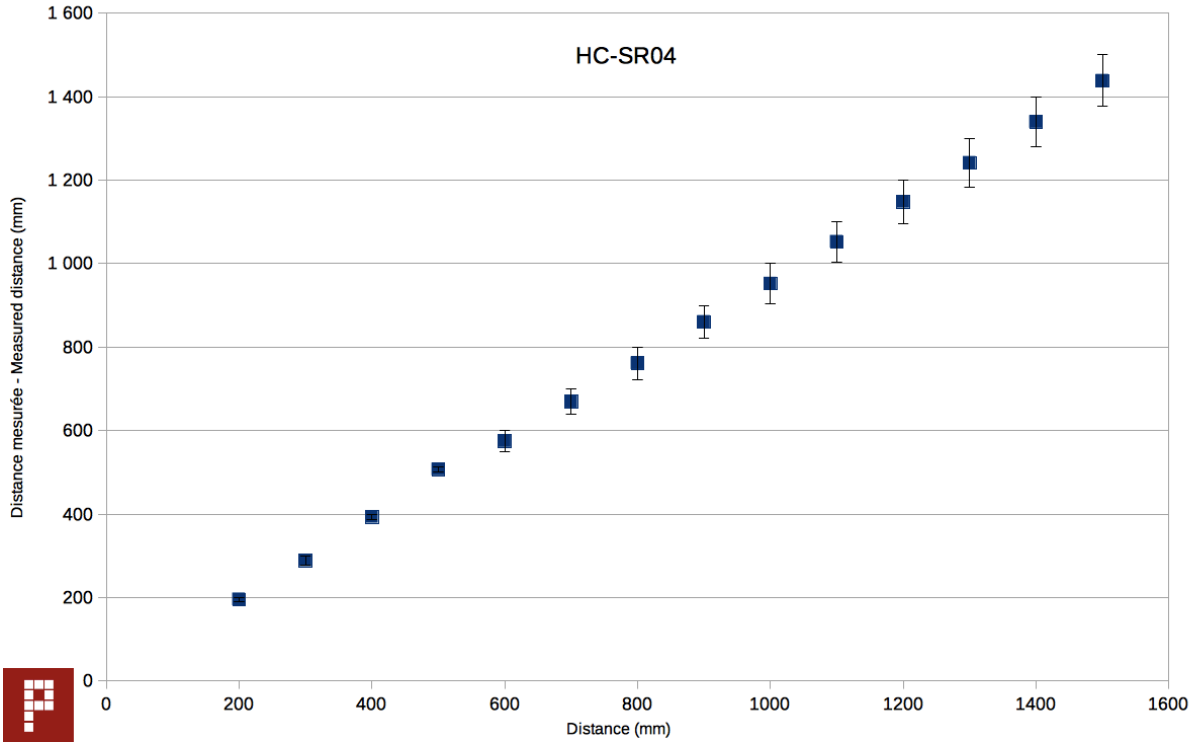


Figura Apéndice 2:3 Test de distancia para el sensor HC-SR04.

Distancia (mm)	Distancia medida con HC-SR04 (mm)	Delta (mm)	Desviación estándar (mm)
200	195,3	-4,7	1,25
300	288,8	-11,2	1,69
400	393,3	-6,7	3,53
500	506,9	6,9	6,61
600	574,8	-25,2	2,7
700	669	-31	3,16
800	761,2	-38,8	4,52
900	860,2	-39,8	1,69
1000	952,1	-47,9	6,61
1100	1 052,2	-47,8	7,19
1200	1 148,1	-51,9	7,09
1300	1 241,5	-58,5	4,48
1400	1 339,5	-60,5	6,22
1500	1 438,1	-61,9	6,49

Tabla Apéndice 2:1 Síntesis de los resultados de las mediciones del sensor HC-SR04.

Y los resultados para el sensor laser **VL53L0X:**

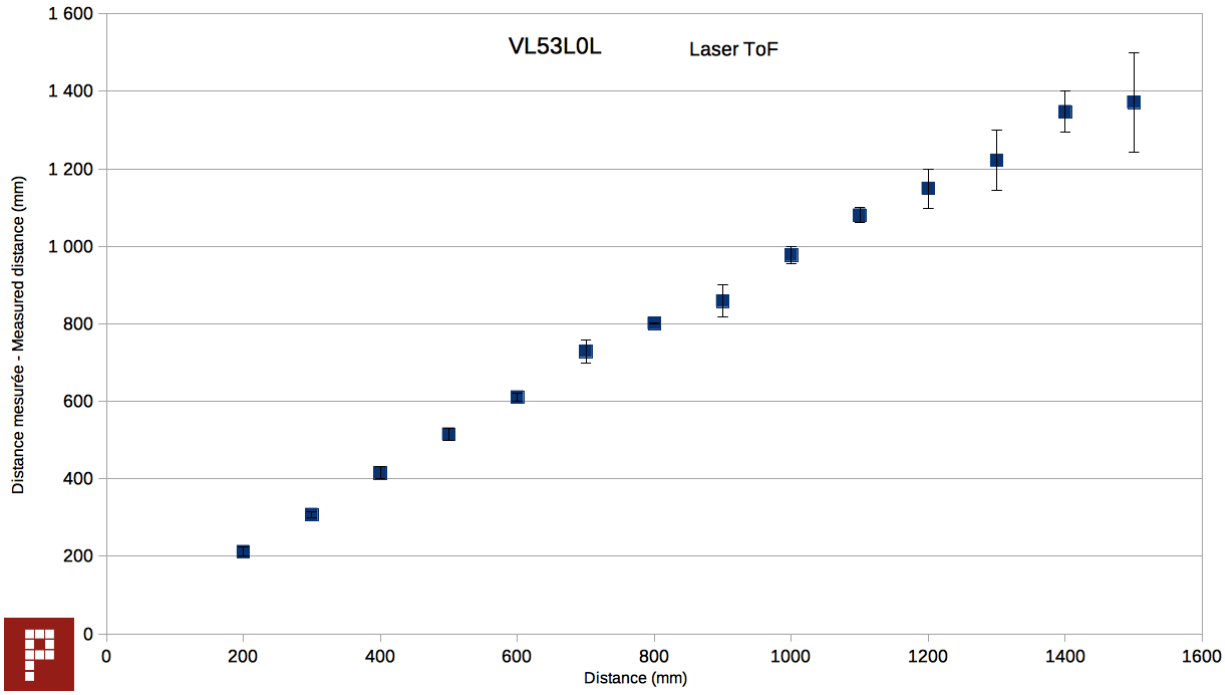


Figura Apéndice 2:4 Test de distancia para el sensor VL53L0X.

Distancia (mm)	Distancia medida con VL53L0X (mm)	Delta (mm)	Desviación estándar (mm)
200	211,8	11,8	1,81
300	307,7	7,7	3,16
400	415,5	15,5	3,03
500	515,1	15,1	5,93
600	610,7	10,7	7,23
700	729	29	5,96
800	801	1	11,24
900	858,6	-41,4	24,39
1000	977,3	-22,7	17,99
1100	1 080,3	-19,7	19,79
1200	1 149,3	-50,7	50,11
1300	1 221,8	-78,2	39,89
1400	1 347,3	-52,7	42,06
1500	1 371,9	-128,1	45,45

Tabla Apéndice 2:2 Síntesis de los resultados de las mediciones del sensor VL53L0X.

Conclusiones

Como se pueden ver en las imágenes el sensor de ultrasonidos va aumentando su error conforme se aleja del objeto, sim embargo el sensor laser solo aumenta cuando se sale del rango de funcionamiento normal, si se usa el modo del sensor para distancias largas mejoraría su resultado considerablemente.
Tras ver este estudio realizado por [24] se puede apreciar de una mejor forma porque se ha optado por el sensor laser además de su tiempo de reacción.

APÉNDICE 3: IMPRESIÓN 3D

La impresión 3D ha aumentado en fama durante las últimas décadas y ha llegado hasta el uso doméstico, sim embargo y aunque se pueden redactar otro proyecto entero hablando sobre ella como no ha sido imprescindible en este proyecto solo se pasaran a comentar los rasgos más significativos.

Introducción de la impresión 3D

A partir de la información sacada de [25] podemos hacer un resumen sobre lo que significa esta nueva técnica:

"Una impresora 3D es una máquina capaz de realizar réplicas de diseños en 3D, creando piezas o maquetas volumétricas a partir de un diseño hecho por ordenador, descargado de internet o recogido a partir de un escáner 3D. Surgen con la idea de convertir archivos de 2D en prototipos reales o 3D. Comúnmente se ha utilizado en el prefabricado de piezas o componentes, en sectores como la arquitectura y el diseño industrial. En la actualidad se está extendiendo su uso en la fabricación de todo tipo de objetos, modelos para vaciado, piezas complicadas, alimentos, prótesis médicas (ya que la impresión 3D permite adaptar cada pieza fabricada a las características exactas de cada paciente), etc.

La impresión 3D en el sentido original del término se refiere a los procesos en los que secuencialmente se acumula material en una cama o plataforma por diferentes métodos de fabricación, tales como polarización, inyección de aporte, inyección de aglutinante, extrusión de material, cama de polvo, laminación de metal, depósito metálico."

Impresora usada

En este caso se ha usado la impresora **Creality Ender 3.** Mas información sobre esta impresora, así como de otras, se puede encontrar en la página oficial [26].

Figura Apéndice 1 Impresora Creality Ender 3.

Programa de Edición

El programa usado para la creación de las piezas en 3D ha sido **OpenSCAD** [27].

"OpenSCAD es un software para crear modelos sólidos de CAD en 3D. Es un software gratuito y está disponible para Linux / UNIX, Windows y Mac OS X. A diferencia de la mayoría de los programas gratuitos para crear modelos 3D (como Blender), no se centra en los aspectos artísticos del modelado 3D, sino en los aspectos CAD. Por lo tanto, sería la aplicación que está buscando cuando planea crear modelos 3D de partes de máquinas, pero está bastante seguro de que no es lo que está buscando cuando está más interesado en crear películas animadas por computadora.

OpenSCAD no es un modelador interactivo. En cambio, es algo así como un compilador 3D que se lee en un archivo de secuencia de comandos que describe el objeto y representa el modelo 3D de este archivo de secuencia de comandos. Esto le da a usted (el diseñador) un control total sobre el proceso de modelado y le permite cambiar fácilmente cualquier paso en el proceso de modelado o hacer diseños que estén definidos por parámetros configurables.

OpenSCAD proporciona dos técnicas principales de modelado: primero, existe una geometría sólida constructiva (también conocida como CSG) y, segundo, existe la extrusión de contornos en 2D. Los archivos DXF de Autocad se pueden usar como formato de intercambio de datos para dichos esquemas 2D. Además de las rutas 2D para la extrusión, también es posible leer los parámetros de diseño de los archivos DXF. Además de los archivos DXF, OpenSCAD puede leer y crear modelos 3D en los formatos de archivo STL y OFF."

Pero, en resumen, se ha escogido OpenSCAD para el modelado de las piezas porque es un programa que trabaja de una manera más simple con piezas no-orgánicas al ser un programa en el que se trabaja con un lenguaje de programación propio muy parecido a C y que permite crear piezas solo con saber las dimensiones que queremos y no la posición relativa de cada parte de la pieza.

Piezas impresas

Una vez definidas tanto la herramienta de impresión como la herramienta de modelado pasaremos a describir las diferentes piezas creadas que se han incluido en el proyecto.

El tiempo total sumando todas las piezas es de 30 horas, sin contar las horas de impresión de la balsa y el precalentamiento y enfriamiento de la cama.

La impresión se realizó con una boquilla de 0.2 milímetros y a velocidad de 30mm/s tanto para relleno como para borde.

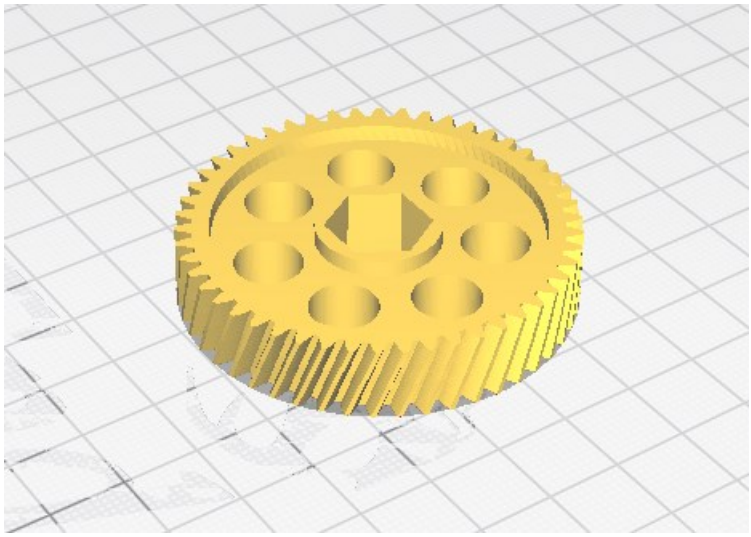


Figura Apéndice 3:1 Engranaje colocado en el eje de las ruedas.

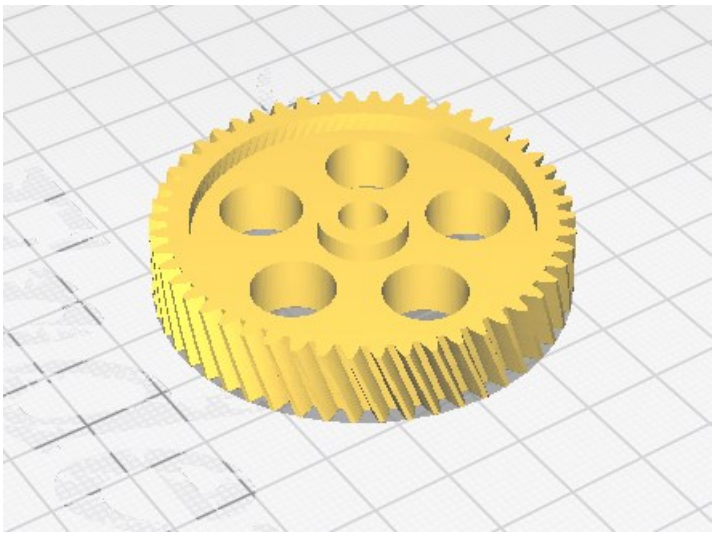


Figura Apéndice 3:2 Engranaje colocado en el eje del motor

Estos dos engranajes son del mismo tamaño y del mismo número de dientes como ya se ha comentado en el trabajo principal, por lo demás la única diferencia entre ellos es el ángulo de los dientes, son opuestos para que puedan encajar, y el tamaño del agujero central dependiendo de que valla en su interior.

También añadir que cada uno contiene pequeñas intercepciones para reducir su materia, así como las horas de impresión y por ambas el coste de su producción.

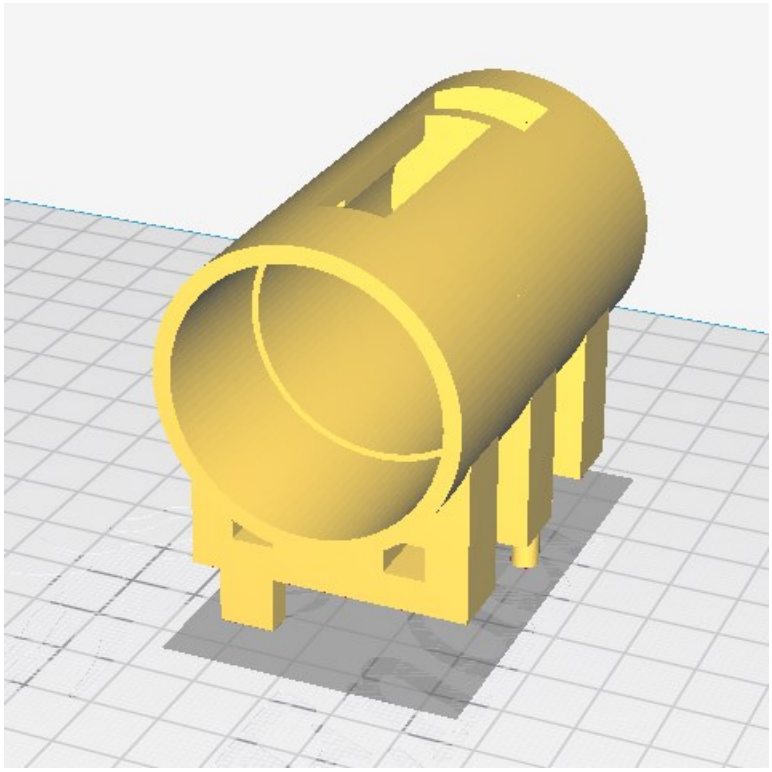


Figura Apéndice 3:3 Contenedor del motor.

La pieza anterior es la que sujetará el motor al robot y lo dejará fijo, para que los engranajes actúen de la forma correcta.

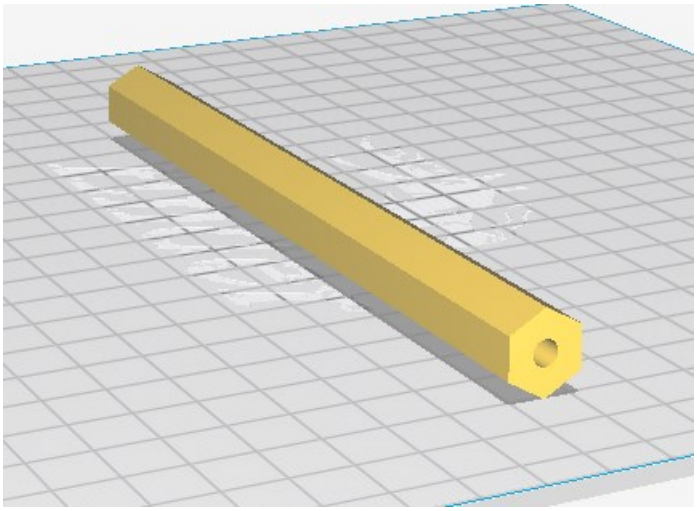


Figura Apéndice 3:4 Eje de las ruedas.

Esta pieza es el eje usado para sujetar las guerras y en su interior se incluye una varilla metálica para aumentar aún más su rigidez.

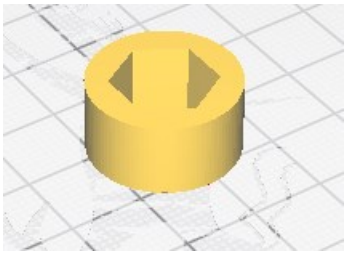


Figura Apéndice 3:5 Añadido al eje para su sujeción.

Esta pieza irá sujeta al eje y posteriormente será incluida a la siguiente pieza que sujetará el eje al robot. Esta pieza se podría haber impreso pegada al eje, pero para simplificar la impresión se ha decidido hacerla por separado. Para mejorar la rotación del eje se podría incluir algún engrasante a esta pieza que será la que este en rozamiento peor no se ha considerado necesario.

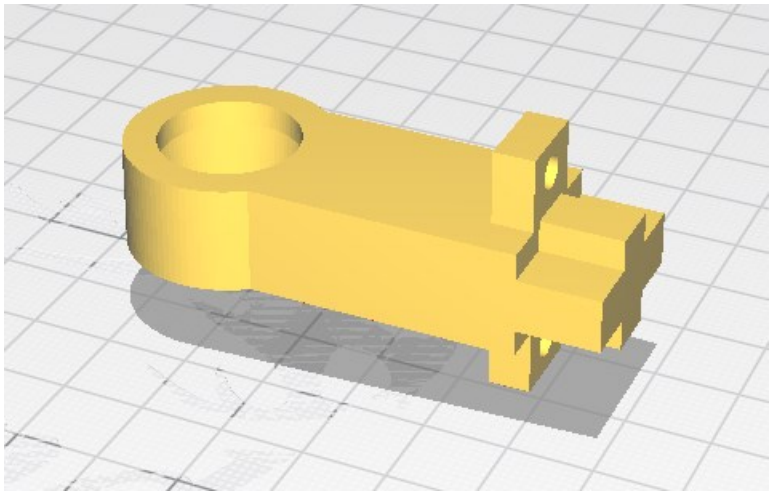


Figura Apéndice 3:6 Pieza que sujeta el eje al robot y permite que gire.

Por último, las dos siguientes piezas son un añadido que soporta las dos piezas de madera y crea un compartimento intermedio donde van colocadas las baterías.

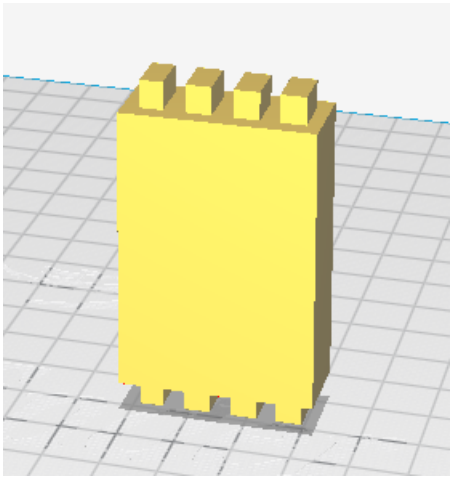


Figura Apéndice 3:7 Soporte delantero del interior del robot

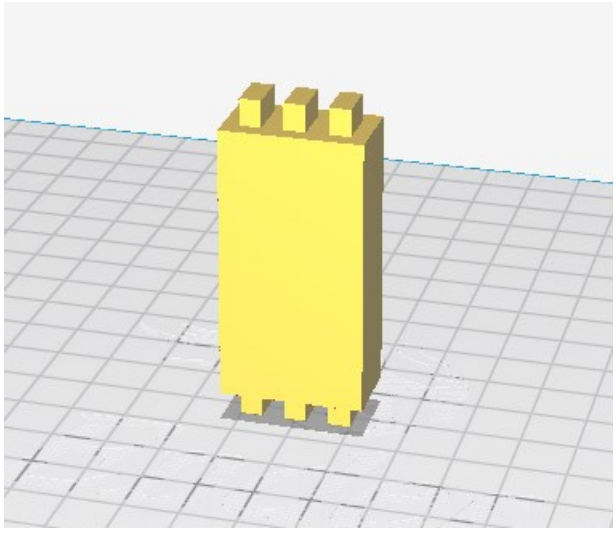


Figura Apéndice 3:8 Soporte trasero del interior del robot

APÉNDICE 4: CÓDIGO

A continuación, se deja en código contenido en la Raspberry, que ejecuta tanto el control como la recogida de datos, y se ha realizado durante el proyecto redactado en este documento.

```
#########################################################
#Código para el control del robot basado en la teoría Tau
#########################################################
#En primer lugar las librerías usadas
########################
#Librería que habilita la GPIO de la Raspberry
import RPi.GPIO as GPIO
########################
#Librerías que permiten usar funciones del sistema
import os
import sys
########################
#Librería que permite usar funciones referidas al tiempo
import time
########################
#Librería que habilita el uso de Señales POSIX
import signal
########################
#Librería del sensor laser SENSOR
import VL53L0X
########################
#Librería que permite pintar gráficas, similar a Matlab
import matplotlib.pyplot as plt
###############################################
###############################################
#Se crea la excepción en caso de querer usar señales POSIX
#Sin embargo, tienen que ser tiempos superiores a 1 segundo
class Time_Exception(Exception):
    pass
def handler(signum, frame):
    raise Time_Exception('Action took too much time')
signal.signal(signal.SIGALRM, handler);
signal.alarm(0);
```

```
#A continuación se definirán los diferentes pines usados y su numero
#Pines puente H
enable_motorA = 16
in1_puenteH = 20
in2_puenteH = 21

##Pines Encoder
Enable_encoder=26
Encoder_1=19
Encoder_2=13

#Configura los pines según el microprocesador Broadcom
GPIO.setmode(GPIO.BCM)
#Elimina los warnings
GPIO.setwarnings(False)

#Configuramos los pines del Puente H como salidas
GPIO.setup(enable_motorA,GPIO.OUT)
GPIO.setup(in1_puenteH,GPIO.OUT)
GPIO.setup(in2_puenteH, GPIO.OUT)

#Define las salidas PWM para el Puente H
pwm_motor = GPIO.PWM(enable_motorA,500)
#Inicia en 0
pwm_motor.start(0)

####################################
#Funciones de sentido de giro de los motores
def  Giro_Favor_Reloj_Motor():
       GPIO.output(in1_puenteH,False)
       GPIO.output(in2_puenteH,True)
def Giro_Contra_Reloj_Motor():
       GPIO.output(in1_puenteH,True)
       GPIO.output(in2_puenteH,False)

####################################
#Continuamos definiendo los pines
#Encoder
GPIO.setup(Enable_encoder, GPIO.OUT)
GPIO.setup(Encoder_1, GPIO.IN)
GPIO.setup(Encoder_2, GPIO.IN)

################################
#Pin conectado al botón de inicio
boton_inicio=17
GPIO.setup(boton_inicio, GPIO.IN)

#Pin conectado a las luces freno
luz_freno1=27
luz_freno2=22
GPIO.setup(luz_freno1, GPIO.OUT)
GPIO.setup(luz_freno2, GPIO.OUT)
GPIO.output(luz_freno1,False)
GPIO.output(luz_freno2,False)
################################
#Fin de la definición de entradas y salidas de la GPIO
```

```
####################################
#Ahora se definirán los parámetros del control
#K para el tipo de seguimiento
K=0.6;
#para el PID
P=1.5;
I=1.5;
D=0.05;
####################################
#Fin de la definición de los parámetros de control
####################################
#Una vez terminadas las definiciones empezamos
#en primer lugar definimos un sentido de giro
Giro_Favor_Reloj_Motor();
#y activamos el encoder
GPIO.output(Enable_encoder,True)

#####################################
#El siguiente código corresponde al inicio
#####################################
#En primer lugar hay que mantener pulsado un botón durante 1 segundo
#Que sea durante un segundo evita posibles errores
Inicio=0
while(Inicio==0):
    #Esperamos a que se pulse el botón
    GPIO.wait_for_edge(boton_inicio, GPIO.RISING)
    time.sleep(1)
    if(GPIO.input(boton_inicio) == GPIO.HIGH):
        #si continua activo, se pasa al siguiente paso
        Inicio=1
#####################################
#Para que el robot inicie la marcha habrá que empujarlo
#Primero se cambia la salida del motor al 10% para que sea fácil de
empujar
pwm_motor.start(10)
time.sleep(1)

#Miramos que el tiempo que tardan en llegar dos pulsos del encoder
sea
#lo suficiente pequeño para considerar que se le está empujando
t_vel=1;
while(t_vel>0.01):
    t_ini_vel=time.time();
    GPIO.wait_for_edge(Encoder_1, GPIO.RISING)
    GPIO.wait_for_edge(Encoder_1, GPIO.RISING)
    t_vel=time.time()-t_ini_vel;

#Cuando se cumpla la condición anterior,
#Ponemos el motor a la velocidad máxima
velocidad_motor=100;
pwm_motor.start(velocidad_motor)
#con esta velocidad se aproximará a la pared
####################################
#Tras iniciarlo, a continuación, se iniciará el sensor laser,
#y comenzara el bucle que incluye el control
```

```
#####################################
#A continuación se ejecutará el código que inicia y lee el sensor
# Primero se crea un objeto para el sensor VL53L0X
tof = VL53L0X.VL53L0X()

#Le manda la orden para que ejecute la lluvia de datos
tof.start_ranging(VL53L0X.VL53L0X_BETTER_ACCURACY_MODE)

#Se lee el tiempo que tardará en enviar cada dato
tiempo_ciclo = tof.get_timing()
#El tiempo devuelto está en microsegundos
#Se comprueba que es correcto
if (tiempo_ciclo < 20000):
    #Detenemos la ejecución del programa si es errónea
    sys.exit()

#Leemos la primera distancia
distance = tof.get_distance()
#Comprobamos que no es errónea
if(distance>0):
    #Si devuelve un número negativo es un error
    #Luego, detenemos la ejecución del programa
    sys.exit() ##detenemos la ejecución
#####################################
#####################################
#Variables del filtro de Kalman
#Variable que indica que el filtro de Kalman esta activo
FKalman_activo=2;
#Se inicia en 2 para que este primer valor indique que no hay
predicción
################
#Las diferentes variables usadas, en este caso simples al ser solo
posición
#desviación típica del ruido
dm_laser = 0.5;
dm_encoder = 0.2;
#Varianza del error
Q_laser = dm_laser ^2;
Q_encoder = dm_encoder^2;
#Matrices del sistema lineal
# A = 1;
# B = 0;
# H = 1;
#Matriz identidad
# I = 1;
#En este caso no harán falta al estar en tamaño 1X1
#####################################
#Definimos las variables que se usarán en el programa
control_activo=0;
fin_control=0;
vel_media_inicio=0;
contador_vel=0;
error_pos_ant=0;
error_vel_ant=0;
distance_ans=0;
###################
```

```
#Las siguientes variables se usarán para guardas los datos
tiempo_g=[]
PID_sal=[]
PID_K=[]
PID_I=[]
PID_D=[]
pos_g=[]
vel_g=[]
pos_d=[]
vel_d=[]
error_pos_g=[]
error_vel_g=[]
#####################################
#Empieza el bucle
start_time = time.time();
while(fin_control==0):
    #Miramos el tiempo de inicio de la iteración
    #para comprobar la duración total
    start_cicle_time = time.time()
    #Guardamos el tiempo para poder usarlo posteriormente
    #para pintar gráficas
    tiempo_g.append(start_cicle_time-start_time)
    #####################################
    #Leemos la distancia del sensor y a continuación,
    #ejecutamos el resto. La distancia podría leerse más tarde,
    #pero al ser el motivo de la duración de cada iteración sea:
    #tiempo_ciclo se ha considerado lo prioritario en el bucle
    distance = tof.get_distance()
    #Después, procedemos a lo demás, que es menos sensible al tiempo

    #####################################
    #####################################
    #La condición de final del control, es que el robot alcance
    #la posición objetivo, que por motivos de seguridad se ha
    #considerado como 50mm inferior a la del objeto
    #Además, el sensor no consigue medir correctamente en
    #distancias muy pequeñas, como se ha comentado en la memoria
    if(distance<50):
        #Indicamos la condición de final
        fin_control=1;
        #Detenemos el motor
        pwm_motor.start(0)

    #####################################
    #####################################
    #El siguiente Dato a medir es la velocidad dada por el encoder
    #Sin embargo, para evitar que el encoder no devuelva ningún
    #valor, debido a que el motor no esté girando, es decir,
    #velocidades muy bajas, se ha optado por eliminar este paso,
    #si la diferencia entre las distancias actual y anteriores es
    # menor a 4, número que se considera como el error máximo del
    #sensor estando parado.
    #Aunque durante el movimiento solo se debe alcanzar esta
    #condición al final, durante las pruebas puede llegar a
    #ocurrir que en algún punto crítico se quedase bloqueado
    #durante varios segundos o de forma infinita
```

```
    #Como añadido de seguridad se puede colocar la señal POSIX
    #sigalarm y se verá cómo a continuación:
    if(abs(distance-distance_ans)>4 ):
        #si es mayor, miramos la velocidad con encoder
        ############################
        #Activamos sigalarm a 1 seg que es el tiempo mínimo permitido
        #en caso de que ocurriera algún error
        signal.alarm(1);
        ############################
        #Actuamos de forma normal dentro de un try por si ocurre
        #sigalarm, y tenemos que salir de él.
        try:
            #se realizarán 2 medidas y a continuación una estimación
            #para ver cuál de las dos es más fiable
            #Se podrían realizar más ya que no hay falta de tiempo
            #pero con dos medidas se ha visto que el error es mínimo
            #primera medida
            t_ini_vel=time.time();
            GPIO.wait_for_edge(Encoder_1, GPIO.RISING)
            GPIO.wait_for_edge(Encoder_1, GPIO.RISING)
            t_vel1=time.time()-t_ini_vel;
            #segunda medida
            t_ini_vel=time.time();
            GPIO.wait_for_edge(Encoder_1, GPIO.RISING)
            GPIO.wait_for_edge(Encoder_1, GPIO.RISING)
            t_vel2=(time.time()-t_ini_vel);

            #Para calcular la velocidad actual, hay que multiplicar
            #el tiempo por las diferentes características del robot
            velocidad_actual1=0.19074/t_vel1;
            velocidad_actual2=0.19074/t_vel2;
            #Nos quedamos con la medida adecuada:
            dif=abs(velocidad_actual1-velocidad_actual2)
            if(dif>0.5 ):
                vel_act=min(velocidad_actual2,velocidad_actual1);
            else:
                #media
                vel_act=(velocidad_actual1+velocidad_actual2)/2
        except  Time_Exception:
            #Si se ha tardado más de 1 seg en medir se considera 0
            vel_act=0;
        ############################
        #En el caso de que ocurra todo satisfactoriamente
        #se devuelve a 0 sigalarm
        signal.alarm(0);
    else:
        #En el caso que la diferencia entre la medida de la
        #distancia, no sea suficiente se considera 0, pero no
        #afectará a los cálculos posteriores.
        vel_act=0;
```

```
    ##########################################
    #Para conseguir la velocidad al inicio del movimiento lo mas
    #precisa posible, se realizará una media de las velocidades
    #durante el recorrido previo al inicio
    vel_media_inicio+=vel_act;
    contador_vel+=1;

    ##########################################################
    ##########################################################
    #Filtro de Kalman sobre la posición
    #Hay que tener en cuenta que para velocidades muy pequeñas el
    #encoder no medirá nada, así que no ejecutaremos el filtro de
    #Kalman a partir de que encontremos una velocidad nula.
    #Esto no deberá afectar al control exceptuando la última parte,
    #que la velocidad llegará a 0

       if(vel_act==0):
              FKalman_activo=0;

       #Si continua activo actuaremos de manera normal,
       #guardando el resultado de la posición en distance
       #para que el código funcione de la misma forma si se
       #ejecuta el filtro como si no

       if(FKalman_activo==1):
              ########################
              #Ganancia de Kalman
              kalman_K= kalman_cov* 1/(kalman_cov + Q_laser);
              #Estimación a posteriori
              distance=pos_kalman_pre+kalman_K*(distance-
              pos_kalman_pre);
              #Corrección de la covarianza
              kalman_cov = (1 - kalman_K) * kalman_cov;
              #Estimamos la siguiente posición,
              #Actual más la velocidad por el tiempo
              pos_kalman_pre=distance+vel_act*delta_time
              +Q_laser*Q_encoder;

       if(FKalman_activo==2):
              #Primera iteración, no hay predicción
              #Se colocan los valores iniciales y la predicción
              #para la siguiente iteración
              #Estimamos la siguiente posición actual más la
              #velocidad por el tiempo
              #Estimación del valor
              pos_kalman_pre=distance+vel_act*delta_time;
              #Estimación de la covarianza
              kalman_cov=Q_laser*Q_encoder;
              #Por último se devalúa la variable FKalman_activo,
              #para indicar que la siguiente iteración, se
              #realizará de manera normal.
              FKalman_activo=1;

    ##########################################################
```

```
    ##########################################################
    #tras obtener las medidas, pasamos al control
    if (control_activo==1 and fin_control==0):
        #Si el control esta activo
        #Lo primero que debemos hacer es calcular el porcentaje de
        #tiempo transcurrido de la maniobra para el siguiente punto
        Tm=(time.time()-t_control_ini+delta_time)/Tiempo_maniobra;
        #calculamos el siguiente TM (+delta_time)

        if(Tm>1):
            #Si supera a 1 es que el tiempo ha terminado y,
            #por lo tanto, el control ha terminado también
            pwm_motor.start(0)
            break;

        ###################################
        #Ejecución del algoritmo de control
        #Cálculos de la teoría de Tau ya explicados en el punto 2
        #y en el 3 con las simulaciones
        #Valores nominales deseados
        pos_nominal=-(1-Tm)**(2/K);
        vel_nominal=(2/K)*(1-Tm)**((2/K)-1);
        #Valores deseados
        pos_deseada=-pos_nominal*D_objeto;
        vel_deseada=vel_nominal*(D_objeto/Tiempo_maniobra)
        ###################################
        #Comenzamos con el PID
        ###################################
        #Calculamos los errores
        error_vel=vel_deseada-vel_act;
        error_pos=pos_deseada-(distance-50)
        ###################################
        #Calculamos los valores K I y D del control
        sal_kp=-(P*error_pos);
        sal_ki=-(I*error_pos*delta_time);
        sal_kd=D*(error_pos-error_pos_ant)/delta_time
        Salida_control=sal_kp+sal_ki+sal_kd;
        ###################################
        #Guardamos los valores de los errores
        error_vel_ant=error_vel;
        error_pos_ant=error_pos;
        ###################################
        ###################################
        #Guardamos los datos resultantes para su posterior análisis
        PID_sal.append(Salida_control)
        PID_K.append(sal_kp)
        PID_I.append(sal_ki)
        PID_D.append(sal_kd)
        pos_d.append(pos_deseada)
        vel_d.append(vel_deseada)
        error_pos_g.append(error_pos)
        error_vel_g.append(error_vel)

        ###################################
```

```
        ################################
        #Actuación del control en el motor
        if(Salida_control>0):
            #Sentido positivo
            Giro_Favor_Reloj_Motor()
            if(Salida_control>100):
                Salida_control=100;
        else:
            Giro_Contra_Reloj_Motor()
            #Sentido negativo
            #Aunque no deba activarse nunca
            #En la práctica es más aconsejable limitar el control
            #para que solo avance
            Salida_control=abs(Salida_control)
            if(Salida_control>100):
                Salida_control=100;

        ################################
        #cambiamos el valor del PWM
        pwm_motor.start(int(Salida_control))
        ################################
        #Fin control

    ###############################
    #Condición de activación control
    #Comenzaría en 550-50=500 aproximadamente
    if(distance<550 and control_activo==0):
        #Cuando se cumpla se activará el control para que,
        #en la siguiente iteración actúe
        control_activo=1;

        ###############################
        #Por lo tanto, se definirán las variables que necesita
        #el control al inicio
        #Distancia a la que se desea detenerse, se coloca
        #un poco antes para evitar posibles choques por fallo
        D_objeto=distance-50;
        #Se calcula la velocidad inicial haciendo la media
        vel_ini=vel_media_inicio/contador_vel;
        #Se calcula tau sub zero
        tau_zero=-D_objeto/vel_ini;
        #Se calcula el tiempo total que durará la maniobra
        Tiempo_maniobra=-(2*tau_zero)/K; #tiempo de maniobra
        #tiempo en el que se inicia el control para
        #saber cuándo acaba la maniobra
        t_control_ini=time.time();

        ###############################
        ###############################
        #Se activan las luces freno
        GPIO.output(luz_freno1,True)
        GPIO.output(luz_freno2,True)
```

```
    ##############################
    #Por último, en el bucle se guardan los valores que se
    #usaran en la siguiente iteración
    pre_vel=vel_act;
    pre_pos=distance-50
    distance_ans=distance

    ################
    #Se guardan también datos para su representación
    pos_g.append(pre_pos)
    vel_g.append(vel_act)
    ##############################

    ##############################
    #Para terminar, se pasa a la espera activa para el
    #siguiente ciclo de ejecución
    while(time.time()-start_cicle_time<tiempo_ciclo/1000000.00):
        pass
    delta_time=time.time()-start_cicle_time;
    #Se comprueba que el tiempo de ejecución se haya cumplido
    if(delta_time>0.07):
        #En caso contrario se tratará como si hubiera
        #ocurrido un fallo, deteniéndolo todo
        pwm_motor.start(0)
        break;

##############################
#Fin del bucle
##############################

#Una vez acabado el control, se pasa a detener
#todas las ejecuciones paralelas
#Empezando por mandarle la orden al sensor laser
#que detenga la lluvia de datos
tof.stop_ranging()

#Deteniendo el motor por si fuera necesario
pwm_motor.stop()

#Apagando las luces de freno
GPIO.output(luz_freno1,False)
GPIO.output(luz_freno2,False)

#Limpiando la GPIO al no volver a usarse mas
#Esto evita posibles fallos en siguientes iteraciones
GPIO.cleanup()
##############################

##############################
#Una vez acabada la maniobra se escribirán los datos
#en un fichero para su posterior análisis,
#en este caso en Matlab

#Abrimos el fichero de guardado en escritura
file = open("Salida_datos.txt","w")
```

```
##############################
#A continuación, guardamos los datos deseados
#En nuestro caso se dejará un dato de ejemplo
#Este dato se guardará de manera que se pueda copiar y
#pegar en la ventana de Matlab para su análisis sin tener que
#cambiar nada, para ello se guardará en forma de vector
file.write("%Pos(medida) Num datos:"+str(len(pos_g))+"\n")
file.write("Valores_Pos=[")
primera=1;
for l in pos_g:
    if(primera!=1):
        file.write(";")
    else:
        primera=0;
    file.write(str(l))
file.write("];"+ '\n')

##############################
#Por último, cerramos el fichero para evitar errores
file.close();

####################################

####################################
#GRÁFICAS
#Esta parte del código está destinada para dibujar las gráficas
#Depende de lo que quiera el usuario
#A continuación, se deja un ejemplo como ene l caso anterior
plt.figure(1);
#En nuestro caso, vamos a pintar la posición guardada desde el inicio
plt.plot(tiempo_g,pos_g);
title('Posición relativa del Robot')
plt.xlabel('Tiempo / ms')
plt.ylabel('Distancia / mm')
plt.grid();
plt.show();
#Esta última función dejara el programa suspendido hasta que se
cierren las gráficas
```

BIBLIOGRAFÍA

[1] G. D. Padfield, «The Tau of Flight Control,» vol. 115, 2011.

[2] P. X. a. O. M. Zhen Zhang, «Bio-Inspired Trajectory Generation for UAV Perching Movement Based on Tau Theory,» 2013.

[3] B. C. a. J. Z. Haijie Zhang, «Extended tau theory for robot motion control,» 2017.

[4] F. Kendoul, «Four-dimensional guidance and control of movement using time-to-contact: Application to automated docking and landing of unmanned rotorcraft systems,» vol. 33, 2014.

[5] I. S. M. P. R. K. J. N. Amedeo Rodi Vetrella, «Improved Tau-Guidance and Vision-aided Navigation for Robust Autonomous Landing of UAVs,» 2017.

[6] D. N. Lee, «Visual-A theory of visual control of braking based on information about time-to-collision,» 1976.

[7] K. S. R. P. Hemjyoti Das, «kalman-Bio-inspired Landing of Quadrotor using Improved State Estimation,» 2018.

[8] dademuchconnection, [En línea]. Available: https://dademuchconnection.wordpress.com/2018/09/27/ejemplo-3-funcion-de-transferencia-de-un-motor-dc-con-carga/.

[9] CQrobot, «Metal DC Geared Motor w/Encoder CQGB37Y001,» 2019. [En línea]. Available: http://www.cqrobot.wiki/index.php/Metal_DC_Geared_Motor_w/Encoder_CQGB37Y001.

[10] Sparkfun, «L298 H Bridge. DUAL FULL-BRIDGE DRIVER,» 2018. [En línea]. Available: https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf.

[11] M. P. Jones, «HC-SR04 User Guide,» [En línea]. Available: https://www.mpja.com/download/hc-sr04_ultrasonic_module_user_guidejohn.pdf.

[12] STMicroelectronics, «VL53L0X Datasheet,» [En línea]. Available: https://www.st.com/resource/en/datasheet/vl53l0x.pdf.

[13] J. B. Moore, «VL53L0X_rasp_python,» [En línea]. Available:

https://github.com/johnbryanmoore/VL53L0X_rasp_python.

[14] Pololu, «VL53L0X distance sensor library,» [En línea]. Available: https://github.com/pololu/vl53l0x-arduino.

[15] Hardzone, 2019. [En línea]. Available: https://hardzone.es/reviews/perifericos/analisis-raspberry-pi-3-modelo-b/.

[16] mootio, 2019. [En línea]. Available: http://www.mootio-components.com/blog/es/tipos-de-engranajes-cual-necesito/.

[17] Lab. Gluón, 2019. [En línea]. Available: http://www.laboratoriogluon.com/filtro-de-kalman-deduccion-ejemplos/.

[18] Wikipedia, «wikipedia.org,» [En línea]. Available: https://es.wikipedia.org/wiki/Filtro_de_Kalman.

[19] Encoder Products Company, 2019. [En línea]. Available: http://encoder.com/blog/encoder-basics/que-es-un-encoder/.

[20] CLR, 2019. [En línea]. Available: https://clr.es/blog/es/tipos-de-encoders-aplicaciones-motores/.

[21] P. f. S.a., 2019. [En línea]. Available: https://www.puntoflotante.net/FUNCIONAMIENTO-ENCODER-CUADRATURA-EFECTO-HALL.htm.

[22] Wikipedia, 2019. [En línea]. Available: https://es.wikipedia.org/wiki/Sensor_ultras%C3%B3nico.

[23] V. P. Bravo, Julio 2017. [En línea]. Available: http://oa.upm.es/48115/3/TFM_VICTOR_PINARGOTE_BRAVO.pdf.

[24] D. Projects, 2018. [En línea]. Available: https://diyprojects.io/hc-sr04-ultrasound-vs-sharp-gp2y0a02yk0f-ir-vl53l0x-laser-solutions-choose-distance-measurement-arduino-raspberrypi/#.XPzKHVUzb0O.

[25] Wikipedia, 2018. [En línea]. Available: https://es.wikipedia.org/wiki/Impresora_3D.

[26] Creality, 2018. [En línea]. Available: https://www.creality3d.cn/.

[27] OpenSCAD, 2019. [En línea]. Available: https://www.openscad.org/.