UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Diseño, implementación y control de un prototipo de BallBot

Trabajo Fin de Máster

Máster Universitario en Ingeniería Mecatrónica

AUTOR/A: García Luengo, Óscar

Tutor/a: Casanova Calvo, Vicente Fermín

CURSO ACADÉMICO: 2022/2023

Índice

1. INTRODUCCIÓN 1

1.1.- ESTADO DEL ARTE 1
1.2.- CARACTERÍSTICAS Y USOS 3

2. OBJETIVOS 5

3. MARCO TEÓRICO 6

3.1.- VELOCIDAD ANGULAR DE LAS RUEDAS EN FUNCIÓN DE LA VELOCIDAD DEL ROBOT 6
3.1.1.- Desplazamiento únicamente sobre el eje Y: 7
3.1.2.- Desplazamiento únicamente sobre el eje X 10
3.1.3.- Giro únicamente sobre el eje Z 13
3.1.4.- Resumen y simplificación 13
3.2.- ESTRATEGIA DE CONTROL 14
3.2.1.- Control de estabilidad 15
3.2.2.- Control de posición 15

4. ETAPA DE SIMULACIÓN 17

4.1.- DISEÑO 3D PARA SIMULACIÓN 17
4.2.- CONSTRUCCIÓN DEL MODELO SIMULINK 18
4.3.- PRUEBAS INICIALES 21
4.4.- CONTROL DE POSICIÓN MANUAL 24
4.5.- CONTROL DE POSICIÓN AUTOMÁTICO 28

5. IMPLEMENTACIÓN REAL 39

5.1.- CONCEPTO DE DISEÑO 39
5.2.- COMPONENTES ELECTRÓNICOS 40
5.2.1.- Motor 40
5.2.2.- Microcontrolador 41
5.2.3.- Controlador de motor 41
5.2.4.- IMU (Inertial Measurement Unit) 42
5.2.5.- Batería principal 42
5.2.6.- Batería secundaria 43
5.2.7.- Módulo Bluetooth 43
5.3.- COMPONENTES MECÁNICOS 43
5.3.1.- Bola 44
5.3.2.- Ruedas omnidireccionales 44
5.3.3.- Acoplamientos rueda-motor 45
5.3.4.- Ejes de rueda 45

5.3.5.- Anclajes ... *46*
5.3.6.- Placa inferior ... *46*
5.3.7.- Placa superior ... *47*
5.3.8.- Anclajes batería ... *48*
5.4.- OTROS COMPONENTES ... 49
5.5.- MONTAJE DEL ROBOT ... 49
5.5.1.- Montaje final ... *49*
5.5.2.- Ajuste de la posición de la batería ... *50*
5.6.- APLICACIÓN MÓVIL ... 50
5.7.- CÓDIGO DE CONTROL ... 53
5.7.1.- Librerías utilizadas ... *53*
5.7.2.- Estructura del código ... *54*
5.7.3.- Modos especiales de funcionamiento ... *56*
5.8.- PRUEBAS INICIALES ... 56
5.9.- CALIBRACIÓN DE CONTROLADOR ... 58
5.10.- INCIDENCIAS ... 59
5.10.1.- Cambio de bola ... *59*
5.10.2.- Cambio de controlador de motor ... *60*
5.11.- RESULTADOS ... 63

6. PLIEGO DE CONDICIONES ... 67

6.1.- OBJETO ... 67
6.2.- REQUISITOS SIMULACIÓN ... 67
6.3.- REQUISITOS IMPLEMENTACIÓN REAL ... 67

7. PRESUPUESTO ... 69

7.1.- PIEZAS IMPRESAS EN 3D ... 69
7.2.- PLACAS DE METACRILATO ... 69
7.3.- TABLA RESUMEN ... 70

8. CONCLUSIONES Y PROPUESTAS DE MEJORA ... 71

9. BIBLIOGRAFÍA ... 73

10. ANEXOS ... 75

10.1.- CÓDIGO ARDUINO DE LA IMPLEMENTACIÓN ... 75
10.2.- PLANOS ... 89

1. Introducción

En este Trabajo Fin de Máster se va a diseñar y construir un prototipo de Ballbot (también llamado Ball Robot, o en español, aunque menos utilizado, Robot bola). Se trata de un robot que está en constante equilibrio sobre una bola y por tanto, a diferencia de la mayoría de robots, es capaz de moverse inmediatamente en cualquier dirección, desde su posición de reposo, sin tener que girar sobre sí mismo.

El principio de funcionamiento del robot es muy parecido al de un péndulo invertido, aunque con la dificultad añadida de que puede caer hacia cualquier dirección. Para mantenerse en pie y desplazarse, el robot utilizará tres ruedas omni-wheel accionadas por sus correspondientes motores, que serán controlados mediante un microcontrolador. Un IMU (Inertial Measurement Unit) proporcionará los ángulos de inclinación del robot.

El trabajo se divide en dos partes principales; primero se realizará una etapa de simulación, en la que se utilizará Simulink para desarrollar un modelo funcional del robot, con control manual mediante gamepad y también con control automático de posición. La segunda etapa consiste en el diseño detallado de las piezas del robot, la construcción del mismo y la realización de pruebas de funcionamiento.

1.1.- ESTADO DEL ARTE

La idea del Ballbot es relativamente moderna. El primer prototipo, llamado CMU Ballbot, se desarrolló en el año 2005, en la Carnegie Mellon University (EEUU), como un diseño experimental de robot capaz de moverse de forma autónoma entre humanos ideado para tareas de vigilancia (1).

En el mismo año salió a la luz un proyecto de la Universidad de Tokyo (Japón) consistente en un Ballbot que permite transportar personas, aunque el comportamiento no era lo suficientemente consistente y se abandonó el desarrollo (2). A partir de entonces, diversos grupos de investigación han ido sacando sus propias versiones del Ballbot, como por ejemplo el Kugle Ballbot, un robot diseñado para interactuar y servir de guía a personas, ya que incorpora una tablet para poder comunicarse.

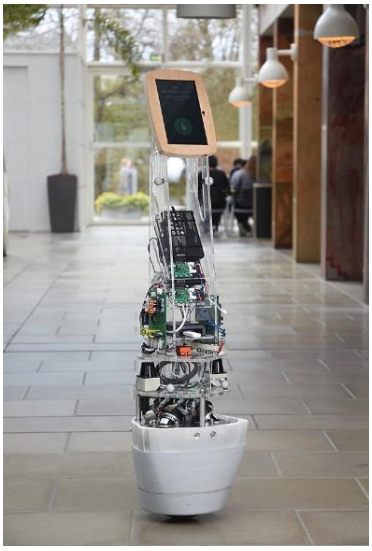


Figura 1.1.- Kugle Ballbot

También se desarrollaron Ballbots pensados para transportar cargas (levantar pesos y permitir que una persona empujando un poco pueda mover pesos grandes). Con el paso de los años los diseños han ido evolucionando, llegando incluso a incorporar brazos robóticos como es el caso del BionicMobileAssistant, el Ballbot diseñado por Festo, que incluye una mano biónica (3). Otro de los Ballbots más destacados es el Rezero, un robot diseñado en la ETH Zurich (Suiza) en el año 2010 que es capaz de moverse de manera extremadamente ágil y de medir distancias con su entorno, posibilitando así aplicaciones como por ejemplo seguir a una persona a una determinada distancia (4).

La investigación en el campo de los Ballbots continúa en la actualidad. Prueba de ello es el Ballbot es el desarrollado en el año 2022 por la Universidad de Illinois (EEUU) llamado PURE (Personal Unique Rolling Experience) (5). Se trata de un Ball robot que funciona como silla de ruedas, y consigue mejorar considerablemente la movilidad, permitiendo maniobrar mejor en espacios reducidos, así como desplazarse lateralmente.

Figura 1.2.- Ballbot PURE

Por último, destacar que no debe confundirse el Ball robot con el Spherical robot (o Robot esférico en castellano), pese a la similitud del nombre, ya que se trata de un robot muy distinto, tanto constructivamente como en sus posibles utilidades. A continuación, se muestra una imagen del Sphero robot, un modelo comercial de Robot esférico (6) orientado a que los jóvenes aprendan a programar:

Figura 1.3.- Sphero robot

1.2.- CARACTERÍSTICAS Y USOS

La principal característica que diferencia a los Ballbots de los robots de ruedas tradicionales es que estos últimos suelen tener bases anchas y el centro de gravedad bajo para tener estabilidad. Por el contrario, los Ballbots son más esbeltos, ya que tener el centro de gravedad alto no es un problema sino una ventaja, por lo que pueden maniobrar mejor

entre personas. Además, se dejan mover al hacer una ligera fuerza contra ellos, algo que no sería posible en un robot de ruedas.

En cuanto a los posibles usos de un robot de estas características se encuentran los siguientes:

- Ayuda a levantarse y sentarse a personas mayores: Al poder inclinarse hacia atrás, el Ball Robot es capaz de cargar con más peso sin necesidad de tener una masa muy elevada, como sería el caso de un robot convencional.
- Guía para personas con discapacidad visual: La persona puede sujetar un brazo del Ball Robot y de esta forma ser guiada por este.
- Tareas de vigilancia: Dada la facilidad que tienen para moverse en cualquier dirección y su esbeltez, la cual permite observar desde una altura considerable, podrían ser adecuados para realizar tareas de vigilancia, por ejemplo, en edificios.
- Transporte de cargas: El Ballbot puede transportar él mismo una cierta carga, o bien facilitar el transporte actuando a modo de carretilla omnidireccional, es decir, la mayoría del peso de la carga recae sobre el robot, pero es el usuario el que lo mueve en una dirección u otra.
- Transporte de personas. Aunque no es el método óptimo para el transporte punto a punto en general (otras soluciones como el segway resultan más cómodas de conducir) sí que es posible que una persona se suba a un Ballbot. Como se ha visto anteriormente, existen Ballbots que funcionan como silla de ruedas, permitiendo al usuario liberar las manos que tradicionalmente se han usado para mover las ruedas así como darle más maniobrabilidad que la que una silla de ruedas proporciona, por ejemplo, permitiendo el movimiento lateral (7).
- Interacción con personas: Debido a su esbeltez y agilidad para esquivar obstáculos, así como su buena respuesta a los empujones, son una muy buena opción como robots que interactúan con personas. Por ejemplo, pueden estar situados a la entrada de un edificio y actuar como recepcionista.

2. Objetivos

El proyecto estará dividido en dos etapas principales, simulación e implementación real. En la etapa de simulación se desarrollará un control del Ballbot, primero una versión manual mediante un gamepad, y posteriormente una versión con control de posición automático, y se demostrará su correcto funcionamiento mediante el seguimiento de varias trayectorias.

En cuanto a la etapa de implementación real, se realizará únicamente el control manual debido a la dificultad de obtener una medición de posición precisa. Para interactuar con el Ballbot, en vez de un gamepad se utilizará una aplicación bluetooth desde un dispositivo móvil. Dicha aplicación permitirá necesariamente la desconexión inmediata de la alimentación de los motores por razones de seguridad, y se podrá utilizar también para otros usos que se consideren adecuados, como por ejemplo modificar los parámetros del controlador con el robot en marcha. El robot deberá disponer de una batería propia de forma que no dependa de estar conectado mediante cables durante su operación. Por supuesto la parte de implementación real engloba también el diseño y selección de componentes, el montaje del robot y el desarrollo del código de control.

Además, se debe tener en cuenta que el robot debe ser fabricado en las instalaciones de la UPV, por lo que se buscarán procesos de fabricación como la impresión 3D y el corte láser. Por último, se elaborará un presupuesto que refleje el coste del prototipo.

3. Marco teórico

3.1.- VELOCIDAD ANGULAR DE LAS RUEDAS EN FUNCIÓN DE LA VELOCIDAD DEL ROBOT

Para establecer un control del Ball Robot, es necesario conocer la relación entre las velocidades de giro de los motores de las ruedas (w_1, w_2 y w_3), que es sobre las que se actúa, y las velocidades deseadas del robot (V_x, V_y) que es lo que se desea controlar. Para ello se ha realizado un análisis matemático, dividido en tres partes, que se corresponden con los tres posibles giros de la bola.

Es necesario antes de nada definir un sistema de referencia en el cual basar los cálculos. Se ha elegido de la siguiente figura, que muestra la bola vista desde arriba sin el robot:

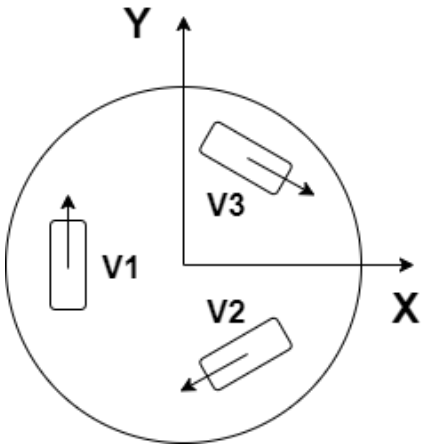


Figura 3.1.- Sistema de referencia

Las distintas velocidades que se muestran, V1, V2 y V3, corresponden con las velocidades lineales de cada rueda en el punto de contacto con la bola. Para trabajar con ellas en los distintos cálculos se deben utilizar sus componentes en ejes X e Y. Se pueden obtener en función de V1, V2 y V3 según las siguientes fórmulas, que serán utilizadas en los diversos apartados:

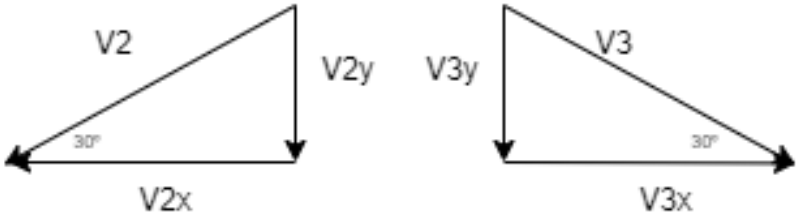


Figura 3.2.- Descomposición de V_2 y V_3

$$v_{1x} = 0 \tag{1}$$

$$v_{1y} = v_1 \tag{2}$$

$$v_{2x} = -v_2 * \cos 30º = \frac{-\sqrt{3}}{2} * v_2 \tag{3}$$

$$v_{2y} = -v_2 * \text{sen}\, 30º = \frac{-v_2}{2} \tag{4}$$

$$v_{3x} = v_3 * \cos 30º = \frac{\sqrt{3}}{2} * v_3 \tag{5}$$

$$v_{3y} = -v_3 * \text{sen}\, 30º = \frac{-v_3}{2} \tag{6}$$

A continuación, se procede a obtener las relaciones entre velocidades de cada rueda y velocidad angular de la bola. Todas las magnitudes están en unidades del sistema internacional.

3.1.1.- Desplazamiento únicamente sobre el eje Y:

Para que la bola se desplace sobre el eje Y, la rueda 1 deberá recorrer la trayectoria marcada en color azul, mientras que las ruedas 2 y 3 deberán desplazarse por la trayectoria marcada en color rojo, que es la misma para estas dos ruedas.

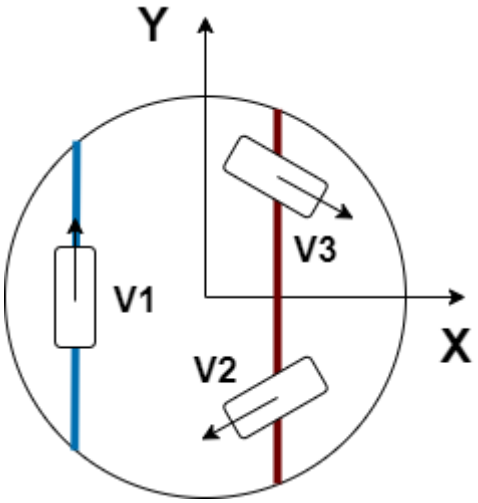


Figura 3.3.- Trayectorias desplazamiento eje Y

Estas trayectorias resultan ser circulares. Ambos movimientos deberán durar el mismo tiempo para que la bola avance únicamente en la dirección Y, por lo que se puede afirmar que:

$$t_1 = t_2 => \frac{d_{1x}}{v_{1y}} = \frac{d_{2x}}{v_{2y}} \quad (7)$$

Donde d_{1x} y d_{2x} corresponden a las distancias a recorrer por las ruedas 1 y 2 respectivamente en un giro alrededor del eje X. Se pueden obtener las expresiones de d_{1x} y d_{2x} ya que las trayectorias son circulares con radio r'_{1x} y r'_{2x}, los cuales dependen del radio de la bola (r) y del radio de contacto de las ruedas con la bola (a) y se pueden obtener trabajando con triángulos (se han resaltado en verde las variables a obtener en cada triángulo):

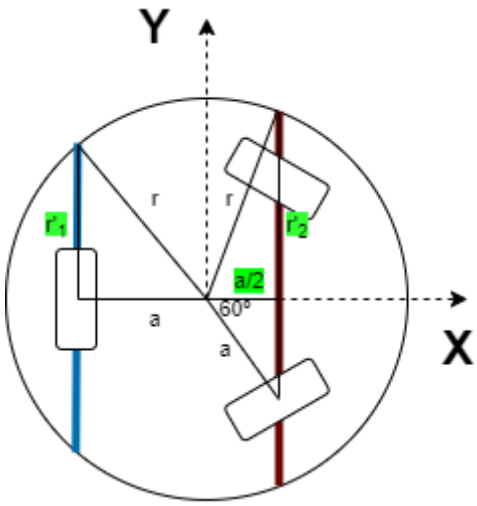


Figura 3.4.- Variables a obtener en cada triángulo

El caso de la rueda 1 es directo, pero en la rueda 2 es necesario operar primero en el triángulo de abajo a la derecha, en el que conociendo el ángulo de 60º y la distancia a es posible determinar que la base del triángulo de arriba a la derecha es igual a $a * \cos(60) = a/2$. Por tanto, las expresiones de los radios de las trayectorias quedan así:

$$r'_{1x} = \sqrt{r^2 - a^2} \tag{8}$$

$$r'_{2x} = \sqrt{r^2 - \frac{a^2}{4}} \tag{9}$$

Y resumiendo, se pueden expresar d_1 y d_2 como:

$$d_{1x} = 2 * \pi * \sqrt{r^2 - a^2} \tag{10}$$

$$d_{2x} = 2 * \pi * \sqrt{r^2 - \frac{a^2}{4}} \tag{11}$$

Por tanto, la expresión inicial igualando los tiempos quedaría:

$$\frac{2 * \pi * \sqrt{r^2 - a^2}}{v_1} = \frac{2 * \pi * \sqrt{r^2 - \frac{a^2}{4}}}{\frac{-v_2}{2}} => v_2 = \frac{-2 * \sqrt{r^2 - \frac{a^2}{4}}}{\sqrt{r^2 - a^2}} * v_1 \tag{12}$$

La expresión anterior relaciona las velocidades de las ruedas 1 y 2, pero no dice nada respecto a la velocidad del robot. Para solucionar esto, se procede de la siguiente manera. Para cualquier rueda i (i = {1,2,3}), se puede afirmar que:

$$w_x = \frac{2 * \pi}{t_i} \tag{13}$$

En el caso de la rueda 1, mediante las ecuaciones *(2)*, *(7)*, *(10)* y *(13)* se puede reescribir como:

$$w_x = \frac{2 * \pi * v_1}{2 * \pi * \sqrt{r^2 - a^2}} => v_1 = \sqrt{r^2 - a^2} * w_x \tag{14}$$

Ahora utilizando la relación entre v_1 y v_2 de la ecuación *(12)* se puede obtener la relación entre v_2 y w_x como:

$$v_2 = \frac{-2 * \sqrt{r^2 - \frac{a^2}{4}}}{\sqrt{r^2 - a^2}} * \sqrt{r^2 - a^2} * w_x =>$$

$$v_2 = -2 * \sqrt{r^2 - \frac{a^2}{4}} * w_x \quad (15)$$

Queda únicamente v_3. Se puede ver intuitivamente que $v_3 = v_2$, aunque a continuación se expone una prueba de ello, partiendo de la base de que ambas ruedas tardan el mismo tiempo en dar una vuelta y que la distancia que recorren es la misma:

$$t_3 = t_2 => \frac{d_{3x}}{v_{3y}} = \frac{d_{2x}}{v_{2y}} \xRightarrow{d_{3x}=d_{2x}} v_{3y} = v_{2y} =>$$

$$v_3 = v_2 = -2 * \sqrt{r^2 - \frac{a^2}{4}} * w_x \quad (16)$$

Finalmente, mediante el radio de la rueda r_r y el radio de la bola r se pueden obtener las relaciones en términos de velocidades angulares de las ruedas en función de velocidades lineales del robot (teniendo en cuenta que $V_y = r * w_x$):

$$w_1 = \frac{\sqrt{r^2 - a^2}}{r * r_r} * V_y \quad (17)$$

$$w_2 = \frac{-2}{r * r_r} * \sqrt{r^2 - \frac{a^2}{4}} * V_y \quad (18)$$

$$w_3 = \frac{-2}{r * r_r} * \sqrt{r^2 - \frac{a^2}{4}} * V_y \quad (19)$$

3.1.2.- Desplazamiento únicamente sobre el eje X

De manera similar al caso de desplazamiento en eje Y, se analizarán las trayectorias de las ruedas, que se muestran en la siguiente figura:

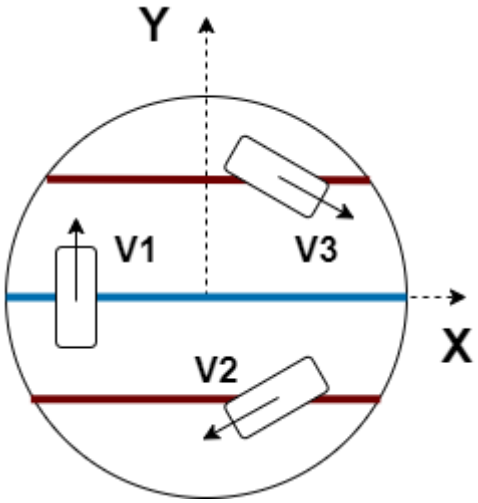


Figura 3.5.- Trayectorias desplazamiento eje X

En este caso, aunque las ruedas 2 y 3 no pasan por la misma trayectoria es fácil ver por simetría que sus recorridos tienen la misma longitud. En cuanto a la rueda 1, su trayectoria es perpendicular a V1, por lo que simplemente deslizarán los rodillos sin que la rueda gire, es decir:

$$\forall w_y, v_1 = 0 \tag{20}$$

Para la rueda 2 se tiene que el tiempo en recorrer su trayectoria alrededor del eje Y es de:

$$w_y = \frac{2 * \pi}{t_2} \tag{21}$$

Donde:

$$t_2 = \frac{d_{2Y}}{v_{2x}} \tag{22}$$

Al igual que en el caso de movimiento en eje Y, la trayectoria es circular por lo que:

$$d_{2Y} = 2 * \pi * r'_{2y} \tag{23}$$

El cálculo de r'_{2y} es similar al caso del movimiento a lo largo del eje Y. En la figura siguiente se muestra un pequeño esquema con las principales variables a tener en cuenta, resaltando también en verde la variable a obtener de cada triángulo:

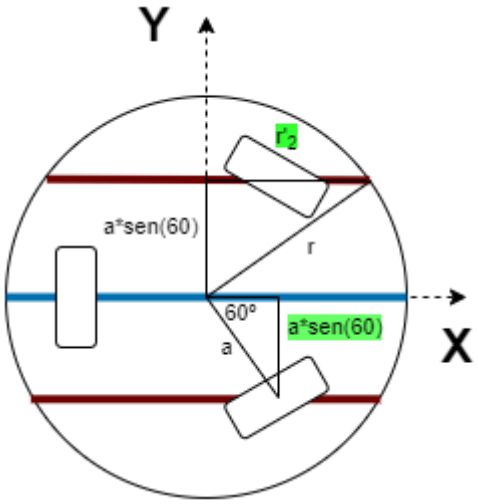


Figura 3.6.- Variables a obtener en cada triángulo

Se puede trabajar en el triángulo inferior primero para obtener la distancia entre las trayectorias, que resulta ser $a * sen(60)$, y de esta forma en el triángulo superior despejar r'_{2y}:

$$r'_{2y} = \sqrt{r^2 - (a * \text{sen}\,60)^2} = \sqrt{r^2 - \frac{3}{4} * a^2} \tag{24}$$

Se puede reescribir utilizando las ecuaciones *(3)*, *(21)*, *(22)* y *(23)* como:

$$w_y = \frac{2 * \pi * v_{2x}}{2 * \pi * \sqrt{r^2 - \frac{3}{4} * a^2}} => v_{2x} = \frac{-\sqrt{3}}{2} * v_2 = \sqrt{r^2 - \frac{3}{4} * a^2} * w_y =>$$

$$v_2 = \frac{-2 * \sqrt{r^2 - \frac{3}{4} * a^2}}{\sqrt{3}} * w_y \tag{25}$$

En cuanto a la rueda 3, la única diferencia respecto al desarrollo de la rueda 2 es en la expresión que relaciona v_{3x} con v_3, es decir, se utilizará la ecuación *(5)* en vez de la *(3)*, por lo que quedaría de la siguiente manera:

$$w_y = \frac{2 * \pi * v_{3x}}{2 * \pi * \sqrt{r^2 - \frac{3}{4} * a^2}} => v_{3x} = \frac{\sqrt{3}}{2} * v_3 = \sqrt{r^2 - \frac{3}{4} * a^2} * w_y =>$$

$$v_3 = \frac{2 * \sqrt{r^2 - \frac{3}{4} * a^2}}{\sqrt{3}} * w_y \tag{26}$$

Finalmente, al igual que en el caso anterior mediante el radio de la rueda r_r y el radio de la bola r se pueden obtener las relaciones en términos de velocidades angulares de las ruedas en función de velocidades lineales del robot teniendo en cuenta que $V_x = r * w_y$:

$$w_1 = 0 \tag{27}$$

$$w_2 = \frac{-2 * \sqrt{r^2 - \frac{3}{4} * a^2}}{\sqrt{3} * r * r_r} * V_x \tag{28}$$

$$w_3 = \frac{2 * \sqrt{r^2 - \frac{3}{4} * a^2}}{\sqrt{3} * r * r_r} * V_x \tag{29}$$

Este caso es particular porque para conseguir desplazamiento en el eje X simplemente se ha de girar las ruedas 2 y 3 a la misma velocidad, pero en sentidos opuestos, manteniendo la rueda 1 parada.

3.1.3.- Giro únicamente sobre el eje Z

Este caso es más sencillo e intuitivo que los dos anteriores, ya que fácilmente se puede ver que si se giran las tres ruedas a la misma velocidad la bola girará sobre el eje Z. No es necesario obtener la relación exacta entre la velocidad de giro sobre el eje Z de la bola y la velocidad de giro de las ruedas, basta con saber que el giro en Z se controla mandando una señal igual a las tres ruedas.

3.1.4.- Resumen y simplificación

Las deducciones anteriores se han centrado en obtener las velocidades de cada rueda en función de la velocidad deseada en cada eje de manera independiente. Se puede por tanto resumir las ecuaciones obtenidas anteriormente de la siguiente manera:

$$w_1 = \frac{\sqrt{r^2 - a^2}}{r * r_r} * V_y \tag{30}$$

$$w_2 = \frac{-2}{r * r_r} * \sqrt{r^2 - \frac{a^2}{4}} * V_y - \frac{2 * \sqrt{r^2 - \frac{3}{4} * a^2}}{\sqrt{3} * r * r_r} * V_x \quad (31)$$

$$w_3 = \frac{-2}{r * r_r} * \sqrt{r^2 - \frac{a^2}{4}} * V_y + \frac{2 * \sqrt{r^2 - \frac{3}{4} * a^2}}{\sqrt{3} * r * r_r} * V_x \quad (32)$$

A la hora de realizar la simulación mediante Simulink y la implementación real estas expresiones se sustituirán por constantes que multiplican a V_x y V_y. Así se podrán corregir fácilmente si hubiese errores en la medida de r, r_r o a, además de que en la implementación real no se gastará capacidad de procesado en hacer una y otra vez el mismo cálculo. De esta forma las expresiones quedan:

$$w_1 = k_1 * V_y \quad (33)$$

$$w_2 = k_2 * V_y - k_3 * V_x \quad (34)$$

$$w_3 = k_2 * V_y + k_3 * V_x \quad (35)$$

Con los valores estimados de $r = 0{,}1m, r_r = 0{,}06m, a = 0{,}034m$ las constantes serían:

$$k_1 = \frac{\sqrt{r^2 - a^2}}{r * r_r} = 15{,}674 \quad (36)$$

$$k_2 = \frac{-2}{r * r_r} * \sqrt{r^2 - \frac{a^2}{4}} = -32{,}848 \quad (37)$$

$$k_3 = \frac{2 * \sqrt{r^2 - \frac{3}{4} * a^2}}{\sqrt{3} * r * r_r} = 18{,}392 \quad (38)$$

3.2.- ESTRATEGIA DE CONTROL

Para lograr que el Ballbot se mantenga en equilibrio es necesario establecer un control dividido en dos partes principales, que se han llamado control de estabilidad y control de posición. El control de estabilidad es el encargado de que el ángulo que forma el robot con el plano horizontal sea lo más cercano a cero posible, mientras que el control de posición busca que el robot no se desplace del lugar donde debe estar.

3.2.1.- Control de estabilidad

El control de estabilidad, como su nombre indica, buscará que el robot se mantenga encima de la bola de la forma más estable posible, es decir, evitando movimientos bruscos. Para ello contará con la lectura del IMU (Inertial Measurement Unit), que estará dispuesto de forma que coincida con los ejes X e Y utilizados.

Se puede analizar el comportamiento del robot por separado en cada eje, ya que se comporta como un péndulo invertido; rota alrededor del punto central de la bola y su centro de masas se encuentra en una posición superior, por lo que si está completamente vertical estará en un equilibrio inestable. Para mantener el robot en esa posición es necesario por tanto incluir un controlador, que tendrá siempre como referencia un ángulo de cero grados.

Este controlador será de tipo PD, con una componente proporcional a la señal del sensor y otra componente proporcional a la derivada de la señal, es decir, a la velocidad angular. En principio no se ha incluido componente integral ya que aumentaría la inestabilidad del sistema y su efecto se vería en cierto modo sustituido por el control de posición.

Es importante destacar que un control de estabilidad por sí solo no es suficiente, ya que inevitablemente el robot comienza a desplazarse en una dirección cada vez más rápido (aunque el ángulo que forme con el plano horizontal sea casi nulo), y acabará cayendo al no poder girar las ruedas lo suficientemente deprisa como para mantener el robot en la parte superior de la bola. Por ello, se hace necesario incluir algún tipo de control de posición que impida al robot desplazarse libremente.

3.2.2.- Control de posición

La idea básica detrás del control de posición es que si se suma un nivel de continua a la señal correspondiente a uno de los ejes el robot se desplazará en esa dirección. Para ejecutar esta idea hay dos posibles enfoques. En primer lugar, el nivel de continua puede venir dado por un usuario, que viendo la posición y velocidad del robot decida hacia dónde debería moverse. La otra opción es diseñar un sistema completamente automatizado, sin necesidad de intervención humana, que sea capaz de llevar al robot a cualquier posición e incluso seguir trayectorias.

En la implementación real este trabajo se va a limitar a la primera opción, y será el usuario el que por medio de una aplicación móvil y conexión bluetooth con el robot, dirija su movimiento. Esta decisión viene dada por la complejidad técnica de implementar un control de posición automático preciso. Sin embargo, en la etapa de simulación se desarrollarán ambos tipos de control, ya que siempre es posible obtener una lectura precisa de posición. En este caso, para el control manual se utilizará como entrada un gamepad.

El control de posición, al igual que el de estabilidad, estará dividido en los dos ejes, X e Y, y será también un PD. Además, contará con una medida de seguridad; estará limitado, tanto en las acciones de control positivas como en las negativas, para garantizar que el control de estabilidad es capaz de mantener el equilibrio del robot. Lo que se consigue con esto es evitar que el robot se incline más de lo debido, lo cual podría causar su caída.

4. Etapa de simulación

Para realizar la simulación se va a utilizar la herramienta Simscape Multibody de Simulink, que permite modelar de una manera relativamente sencilla un sistema físico, así como incluir la parte de control. También ofrece la posibilidad de ejecutar las simulaciones en tiempo real (siempre y cuando el ordenador sea lo suficientemente potente) y recibir información de un gamepad, por lo que proporciona todo lo necesario para el desarrollo de este trabajo

4.1.- DISEÑO 3D PARA SIMULACIÓN

Lo primero que se debe hacer es elaborar un diseño 3D del Ball Robot para poder ejecutar las distintas simulaciones. En esta etapa no se busca tener el diseño definitivo, ni siquiera hace falta un diseño detallado, ya que el objetivo principal es facilitar la simulación. Por ello, se han incluido únicamente los componentes esenciales: bola, rodillos, ruedas y cuerpo del robot, dejando fuera del diseño en esta etapa otros componentes como motores, microcontroladores o baterías. A continuación, se muestra una imagen del modelo desarrollado:

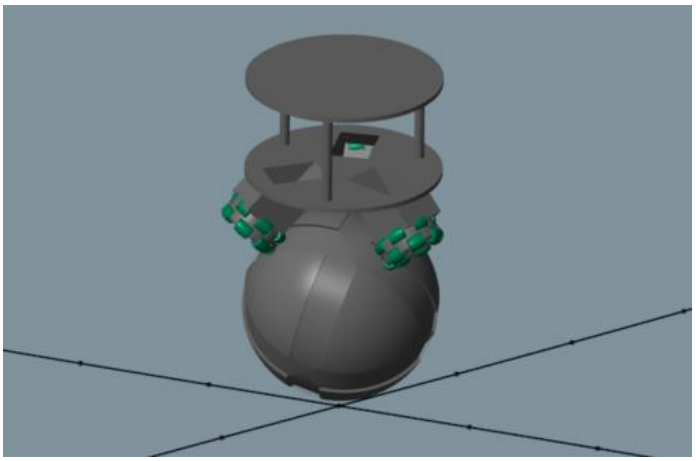


Figura 4.1.- Diseño 3D para simulación

Este diseño no busca tener las medidas definitivas, simplemente que las proporciones sean parecidas, ya que los parámetros de los controladores deberán, con toda seguridad, ser obtenidos de nuevo en el prototipo real. Otro aspecto destacable es que se ha modelado la bola con surcos, de forma que cuando gire se pueda ver fácilmente. Estos surcos no afectarán

en absoluto al funcionamiento del robot ya que en Simulink el robot interactuará solo con una bola transparente completamente esférica que cubra la que tiene surcos.

4.2.- CONSTRUCCIÓN DEL MODELO SIMULINK

El modelo Simulink desarrollado ha ido evolucionando, aumentando poco a poco de complejidad, y en los apartados siguientes se explicarán con más detalle las peculiaridades de cada versión del modelo (pruebas iniciales, control de posición manual y control de posición automático). Aun así, hay características que son comunes a todos y facilitarán la comprensión de los mismos más adelante, por lo que se ha juzgado conveniente introducirlas ahora.

Todas las versiones constan de dos partes principales que confluyen en el subsistema BallRobot. Se muestra la Figura 4.2 a modo de esquema general, con la información que entra y sale de cada parte:

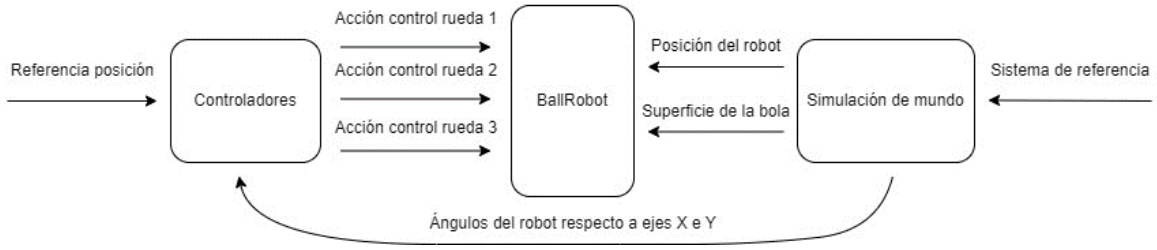


Figura 4.2.- Esquema del modelo simulink

Comenzando por la derecha, se encuentra la parte de simulación de mundo. Aquí se modelan los distintos objetos que interaccionarán con el robot; la bola y el suelo. El flujo de datos va de derecha a izquierda hasta llegar al subsistema BallRobot, y está subdividido en tres partes, que se ven en la Figura 4.3 ya en formato Simulink:

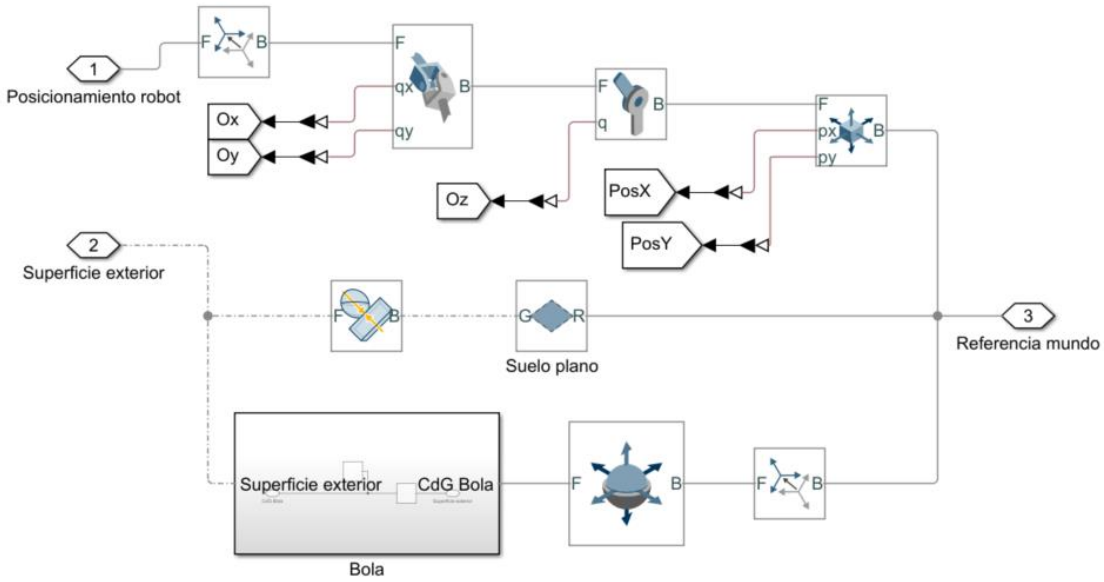


Figura 4.3.- Simulación de mundo

La línea central es la parte del suelo y es la más sencilla, simplemente establece una fuerza de contacto entre este y la superficie exterior de la bola. La parte de la bola (la línea inferior) le permite todos los grados de libertad, y para evitar problemas en el instante inicial la levanta muy ligeramente del suelo. Además, incluye el ensamblaje de la bola con surcos dentro de la bola invisible (subsistema "Bola"), de forma que el robot solo interactúa con esta última. Por último, la parte de posicionamiento del robot (línea superior) le permite todos los grados de libertad, aunque en vez de utilizar un único bloque "6 DOF Joint" se han utilizado bloques que permiten menos grados de libertad en serie, para así poder obtener más fácilmente las posiciones y ángulos necesarios para el control.

A la izquierda del subsistema BallRobot en la Figura 4.2 se encuentran los controladores. Estos variarán considerablemente de versión a versión, por lo que se comentarán con más detalle posteriormente. Lo que sí es común a todas las versiones es el subsistema V2Uruedas, que es el encargado de convertir velocidades en los ejes X e Y, que son las variables que se quieren controlar, en acciones de control a cada una de las ruedas. Esta conversión se ha descrito anteriormente en la sección 3.1. Este subsistema también tiene en cuenta la velocidad de giro respecto al eje Z deseada, como se puede observar en la Figura 4.4:

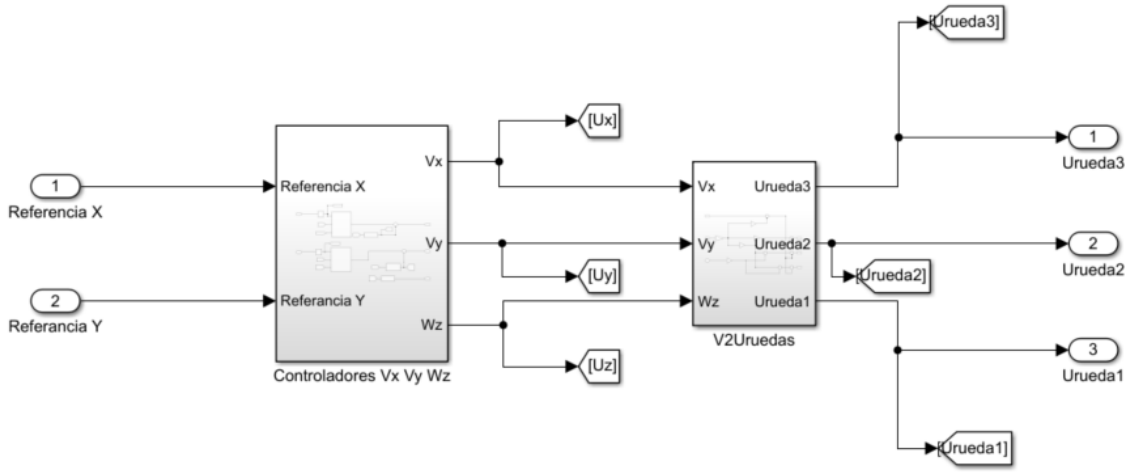


Figura 4.4.- Controladores

Por último, el subsistema BallRobot en sí tiene, como es de esperar, entradas para las acciones de control de las tres ruedas, además de un puerto para la información de posicionamiento en el mundo y otro para la superficie de la bola. Por dentro conecta cada rueda con el cuerpo del robot y con su acción de control correspondiente (puerto "entrada"), como se ve en la Figura 4.5:

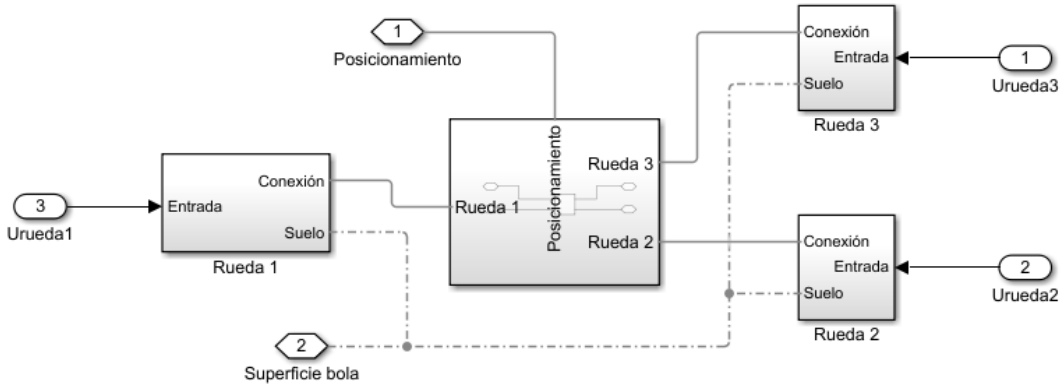


Figura 4.5.- Subsistema BallRobot

Dentro del subsistema de cada rueda se incluye la conexión entre cada rodillo y el cuerpo central de la rueda, la articulación de rotación correspondiente para permitir el giro de los rodillos, otra articulación de rotación para permitir el giro de la rueda respecto al cuerpo del robot y el contacto de todos los rodillos con la superficie de la bola.

A continuación, se describen las pruebas iniciales, el control manual y el control automático, así como las peculiaridades de sus modelos Simulink.

4.3.- PRUEBAS INICIALES

Antes de proceder con las distintas simulaciones del robot es importante comprobar que la conversión obtenida previamente entre las velocidades en ejes X e Y y las acciones de control de cada rueda sea correcta, ya que de lo contrario los distintos controladores funcionarán mal. Para ello se han considerado dos opciones; la primera consiste en colocar el robot boca arriba, es decir, apoyado sobre el suelo dado la vuelta y sujetando la bola. De esta forma, dando una acción de control nula a una de las direcciones (por ejemplo V_x) y no nula a la otra (V_y) se debería observar un giro de la bola únicamente alrededor de un eje (en este ejemplo el eje X), mientras que el giro sobre el otro eje (en este ejemplo el Y) debería ser nulo. La otra opción es un modelo Simulink ligeramente modificado en el que el cuerpo del robot tiene restringido el giro respecto a los ejes X e Y. Esto es equivalente a un control de estabilidad perfecto, en el que el robot nunca se cae porque es imposible que se incline, y permite ver hacia dónde se mueve el robot al aplicar una determinada acción de control a cada rueda.

En un principio se probó con la primera opción por ser más intuitiva, pero los resultados no eran claros ya que tenían algo de ruido y aunque parecían ser correctos no se podía asegurar con certeza. Así que se siguió adelante con la segunda opción, que permite ver con más precisión si el robot se desvía de la trayectoria esperada. Para evaluar el comportamiento del robot se han realizado varias pruebas, que se detallan a continuación.

La primera prueba consiste en mover el robot a velocidad constante de 0,1m/s sobre el eje X. Aplicando esta referencia se observa como en efecto el robot se desplaza en la dirección correcta y a la velocidad especificada, ya que en un tiempo de ejecución de 5 segundos recorre 0,5m.

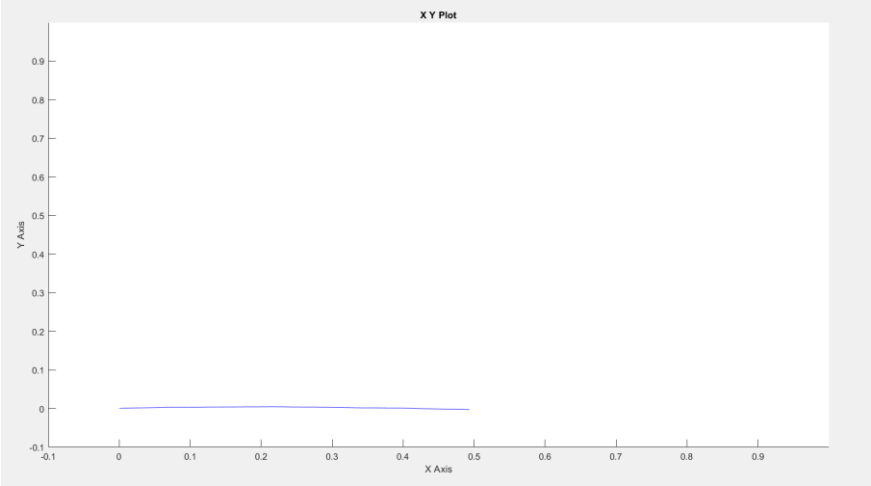


Figura 4.6.- Trayectoria del BallRobot: desplazamiento sobre eje X

La segunda prueba es igual que la primera, pero en el eje Y. En este caso se puede ver como la trayectoria del robot no es del todo recta, y tiende a desplazarse hacia X positivo en una trayectoria curva:

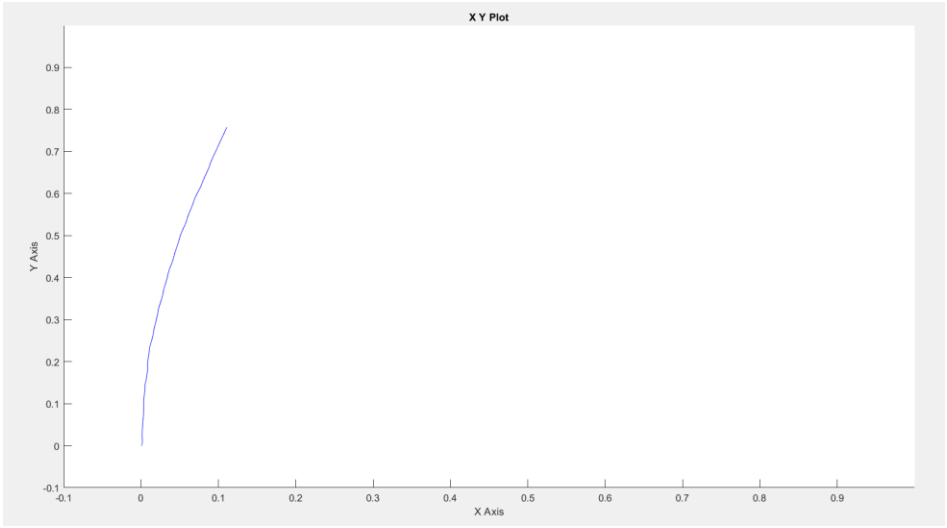


Figura 4.7.- Trayectoria del BallRobot: desplazamiento sobre eje Y

Analizando la simulación se observa que es debido a que se produce un ligero giro en el eje Z, el cual hace que poco a poco la dirección Y del robot se desvíe de la dirección Y real. El problema probablemente sea que los coeficientes calculados k1, k2 y k3 son muy sensibles a imprecisiones en la medida de las magnitudes r,r_r, y a, y se está trabajando con valores estimados de a al ser un valor difícil de medir.

Para arreglar esto se podría buscar una manera de obtener a de forma más precisa, pero se ha optado por una solución alternativa más fácil de replicar en la implementación

real del prototipo. Esta solución consiste en introducir una señal continua de igual magnitud en las tres ruedas y proporcional a la referencia de velocidad en Y. Esto equivale a producir un giro constante respecto al eje Z, y la idea es que contrarreste el giro en Z inducido por las imprecisiones en la medida de las variables. Por supuesto, al estar metiendo una nueva señal es necesario introducir un coeficiente corrector a la referencia de velocidad en Y para que la velocidad real sea la esperada. Tanto la ganancia introducida para la señal de giro en Z como el coeficiente corrector a la referencia de velocidad en Y se han obtenido experimentalmente de forma sencilla. Así, el subsistema quedaría de la siguiente manera:

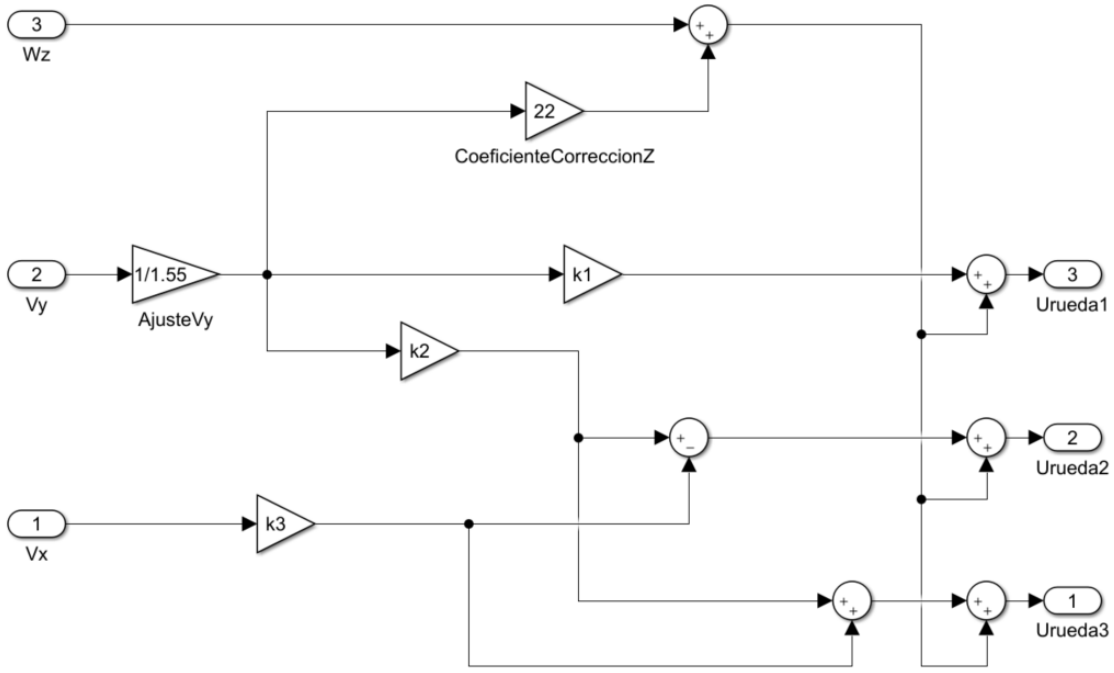


Figura 4.8.- Subsistema V2Uruedas con corrección de desplazamiento en Y

De esta forma se consigue un movimiento en eje Y vertical, sin tender a curvarse hacia la derecha, y a la velocidad especificada de 0,1 m/s:

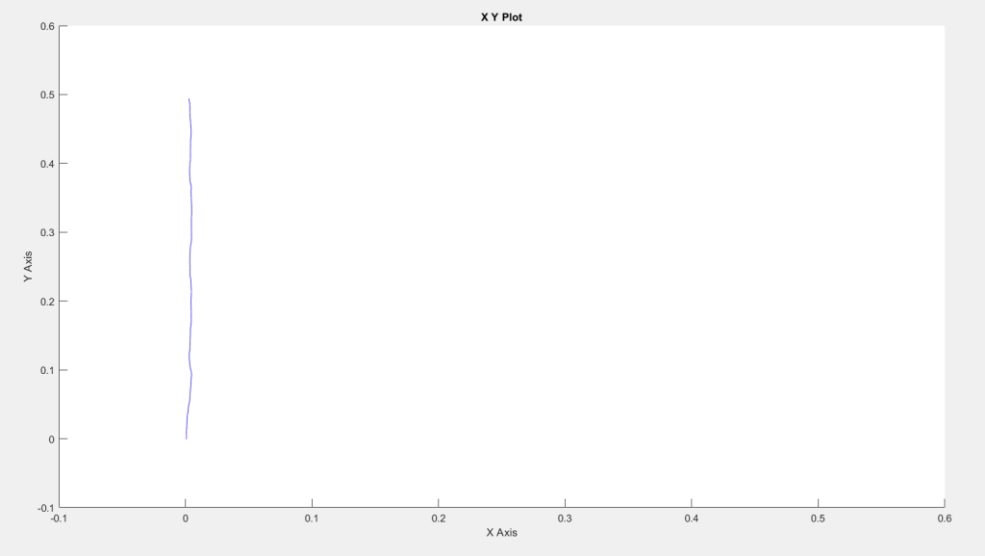


Figura 4.9.- Trayectoria del BallRobot: desplazamiento sobre eje Y corregido

También se ha probado a mover el robot en los dos ejes a la vez a la misma velocidad que en las pruebas anteriores, 0,1 m/s, y el resultado es una vez más el esperado; movimiento en línea recta hasta llegar al punto X=Y=0,5

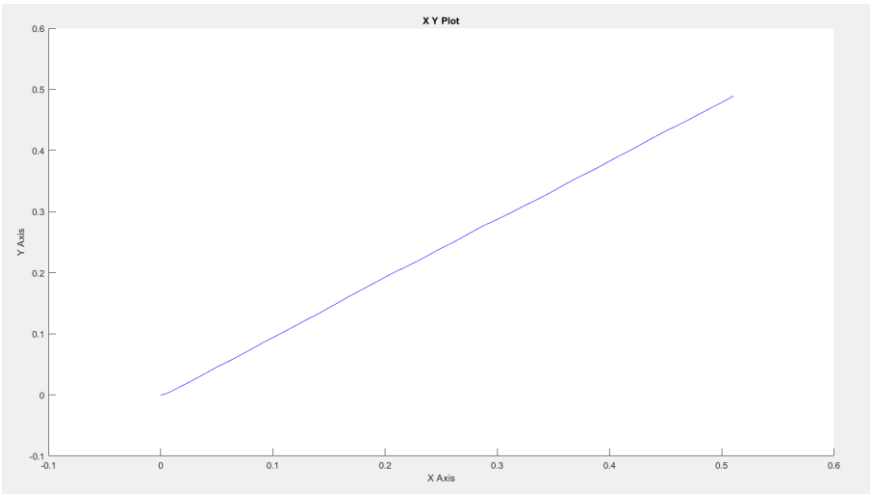


Figura 4.10.- Trayectoria del BallRobot: desplazamiento simultáneo sobre ejes X e Y

Una vez comprobado el funcionamiento del subsistema se puede comenzar con el control de estabilidad del robot.

4.4.- CONTROL DE POSICIÓN MANUAL

Se ha probado a colocar el robot en su posición natural, es decir, encima de la bola, implementando un control de estabilidad que impida que el robot se caiga. Dicho control está dividido en dos partes, una para el eje X y otra para el eje Y. Cada parte está compuesta

por un PD que toma como entrada el ángulo del robot, ya que esta será la medida del sensor en la implementación real. En este caso al tratarse de un control de estabilidad la referencia será siempre 0, es decir, el controlador intentará que el robot esté en posición horizontal.

Para introducir el control manual de posición se ha incorporado una entrada de gamepad, de forma que el usuario pueda desplazar el robot fácilmente al ejecutar la simulación en tiempo real. Este control funciona, tal y como se ha explicado en el Marco Teórico, sumando un nivel de continua proporcional a la señal que llega del joystick correspondiente del gamepad. El esquema del controlador en bloques Simulink puede verse en la siguiente imagen:

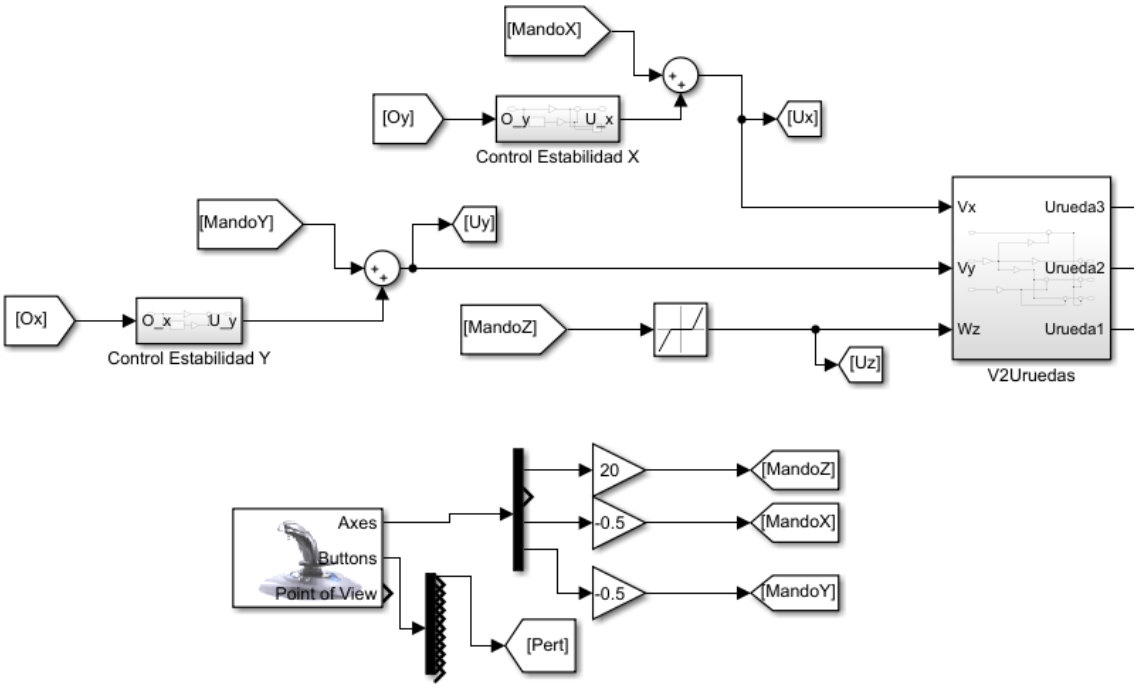


Figura 4.11.- Esquema del controlador manual

Se ha implementado también la posibilidad de introducir una perturbación mediante un botón del gamepad. Esta perturbación consistirá en una fuerza de 2N en el eje X, y estará activa durante todo el tiempo que esté pulsado el botón correspondiente en el gamepad. La forma de implementarlo en Simulink consiste en añadir un bloque “External Force and Torque” cuya entrada sea el botón del gamepad, y se puede ver rodeado en azul en la Figura 4.12:

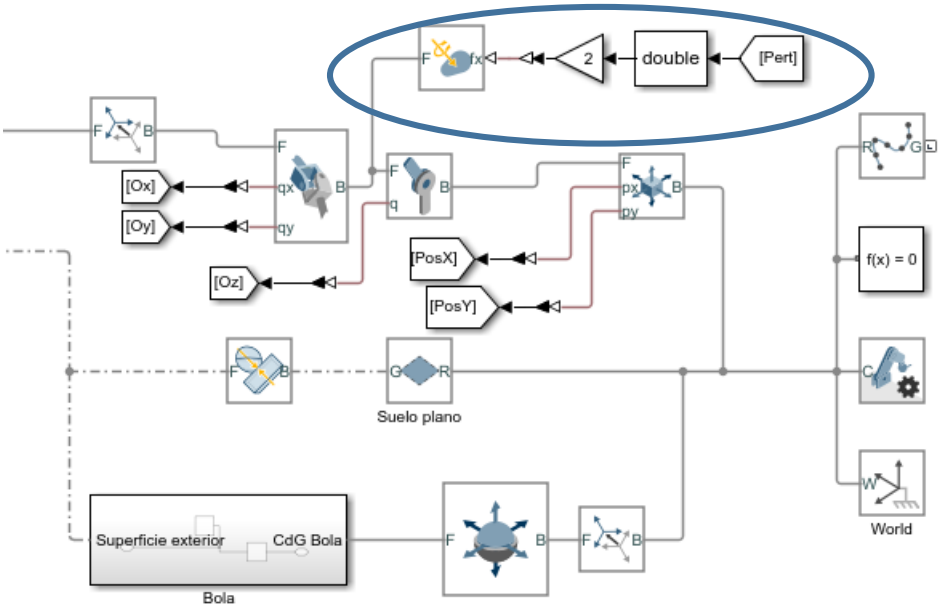
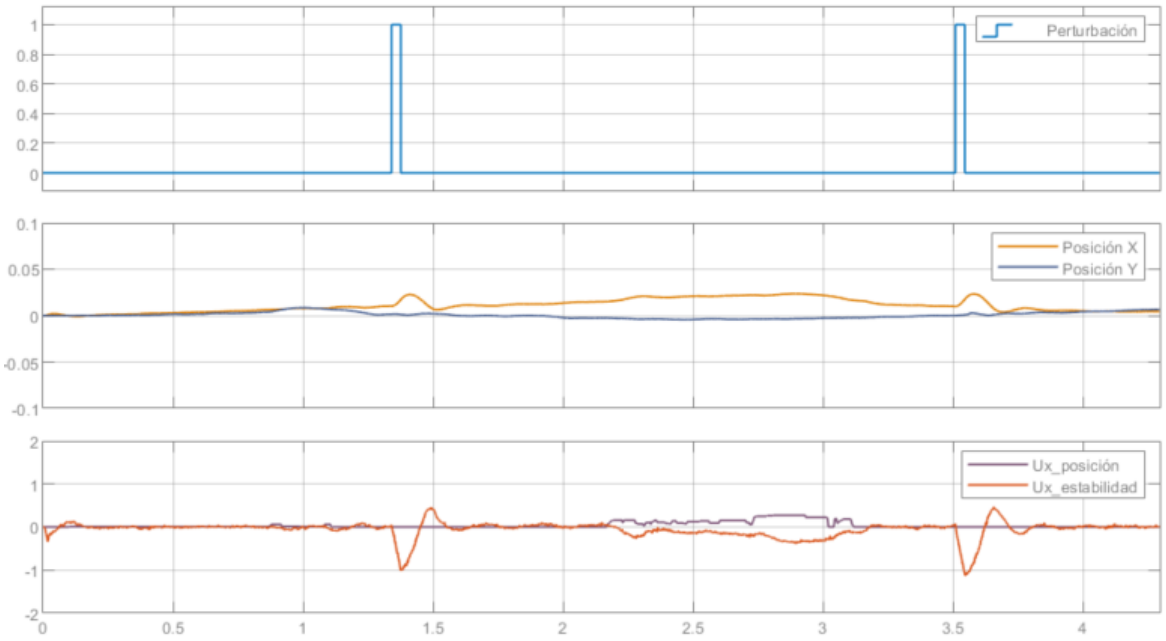


Figura 4.12.- Implementación de perturbaciones en la simulación de mundo

Notar que, aunque hay bloques “Goto” que obtienen la posición X e Y, esta información se utiliza únicamente para obtener gráficas, no para el control. Lo primero es comprobar que el control de estabilidad funciona, y el robot es capaz de mantenerse en la posición inicial si se corrige la posición con el gamepad. Además, se ha probado a aplicar una perturbación para ver cómo reacciona:



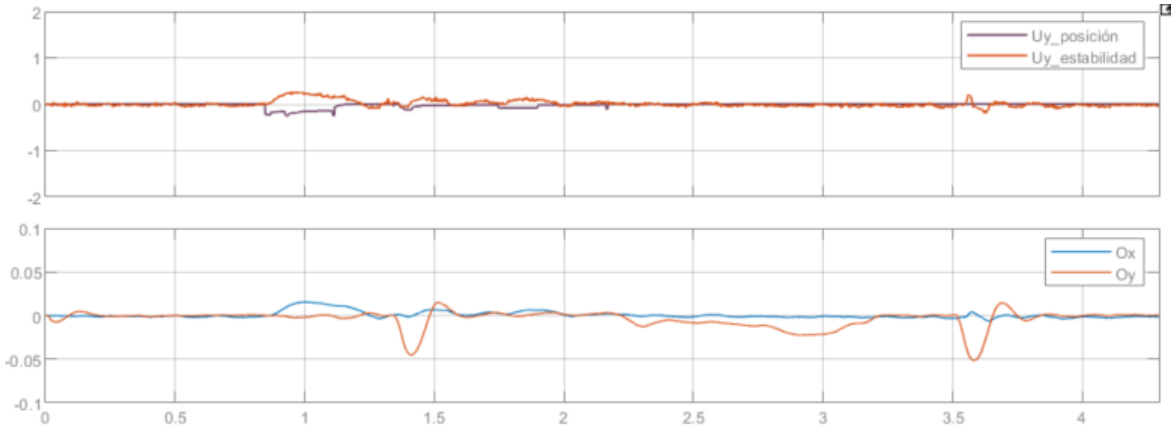


Figura 4.13.- Comprobación de control de estabilidad y respuesta a perturbaciones en control manual

Se puede ver cómo el robot se puede mantener en la posición inicial con facilidad, ya que apenas es necesaria acción de control del mando para mantenerlo en el sitio. Además, es capaz de recuperarse de una perturbación sin ayuda, por lo que la prueba es satisfactoria. A continuación, se ha realizado una prueba de seguimiento de trayectoria. La trayectoria se ha definido mediante un spline cerrado a partir de los puntos (0,0), (0,1), (1,0), (0,-1) y el resultado se muestra a continuación:

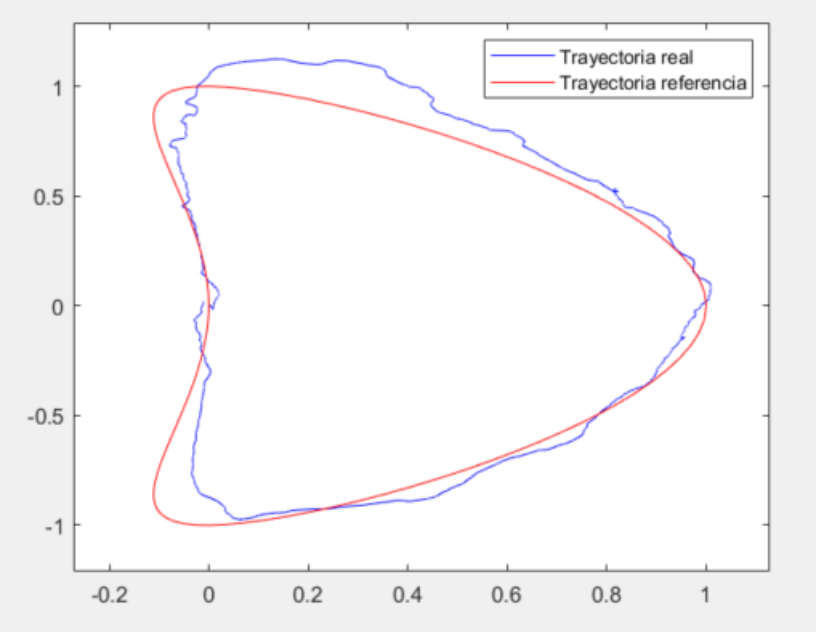


Figura 4.14.- Trayectoria real vs referencia en control manual

Si bien el seguimiento de trayectoria dista de ser perfecto, se debe tener en cuenta que el control es manual mediante un joystick, y con la dificultad añadida de que durante la simulación no se llega a procesar a tiempo real ya que el ordenador no es capaz de realizar

los cálculos necesarios a tiempo, por lo que el sistema aparenta ser más lento que de normal. Aun así, el control resulta relativamente sencillo e intuitivo siempre y cuando la velocidad de ejecución de la simulación sea más o menos parecida a la velocidad real. Las acciones de control y los ángulos de inclinación del robot en cada eje son las siguientes:

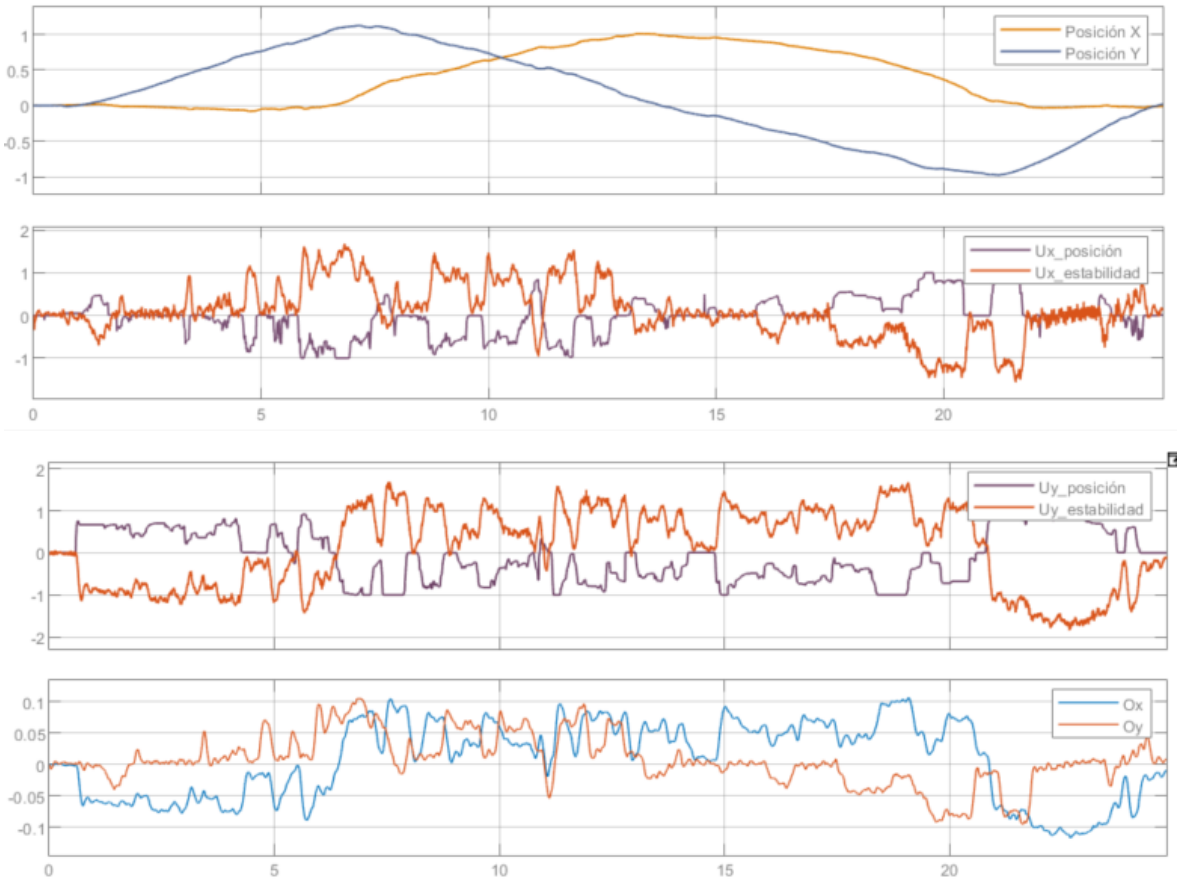


Figura 4.15.- Acciones de control y ángulos de inclinación en seguimiento de trayectoria con control manual

4.5.- CONTROL DE POSICIÓN AUTOMÁTICO

Este control busca sustituir el gamepad, de forma que no sea necesaria la acción humana para controlar la posición del robot. El mecanismo es el mismo en que se basa el control mediante mando; sumar a la acción de control correspondiente una señal de continua. Para esto es necesario utilizar un segundo controlador PD en cada eje, que ha requerido una etapa de ajuste hasta encontrar los valores adecuados. La parte del modelo Simulink relativa al control queda entonces de la siguiente manera:

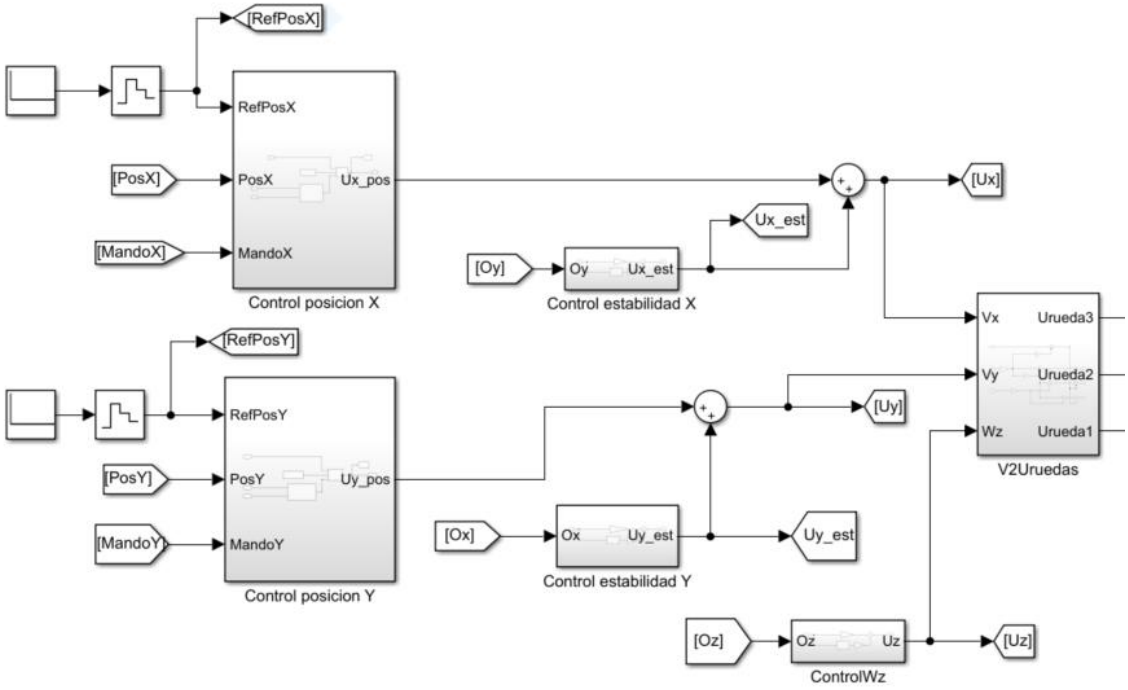


Figura 4.16.- Controladores del modelo Simulink

Los subsistemas Control estabilidad X y Control estabilidad Y son exactamente iguales que en el control manual de posición explicado anteriormente. El subsistema ControlWz mantendrá el ángulo respecto a Z lo suficientemente cerca de cero como para que no afecte al comportamiento del robot. Esto es porque si el robot se gira respecto al eje Z la lectura de ángulos X e Y no corresponderá con los ángulos X e Y globales, y por tanto el control de posición fallará. Por último, los subsistemas Control posición X y Control posición Y tienen una parte de controlador PD y además permiten implementar un control manual o automático en función del valor de la variable controlManual, por lo que, aunque también tienen como entrada la señal del mando, esta no se utiliza en el control Automático.

Una vez desarrollado el control lo primero es, al igual que se hizo en el caso del control manual de posición, comprobar que el robot es capaz de mantenerse estático en su posición de inicio. Además, se ha probado a causar una ligera perturbación para ver cómo reacciona:

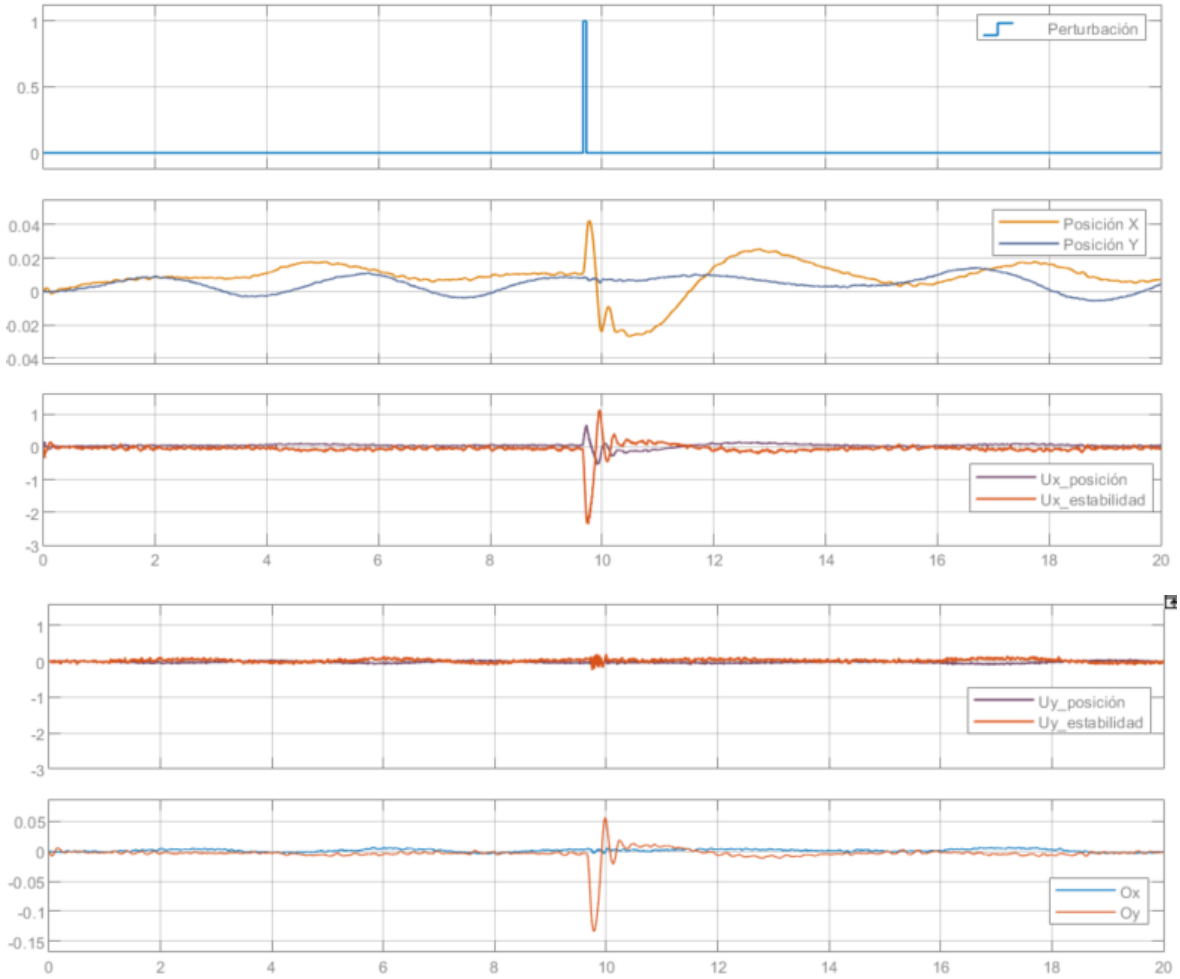


Figura 4.17.- Comprobación de controlador de estabilidad y respuesta a perturbaciones en control automático

El comportamiento es el esperado; el robot se mantiene perfectamente en su sitio, prácticamente inmóvil, y es capaz de volver a ese estado tras aplicar una perturbación. Se ve además como al ser una perturbación en el eje X los controladores del eje Y no reaccionan, tal y como cabe esperar. Una vez verificado el comportamiento estático se ha probado a mover el robot por la trayectoria más simple posible, una línea recta:

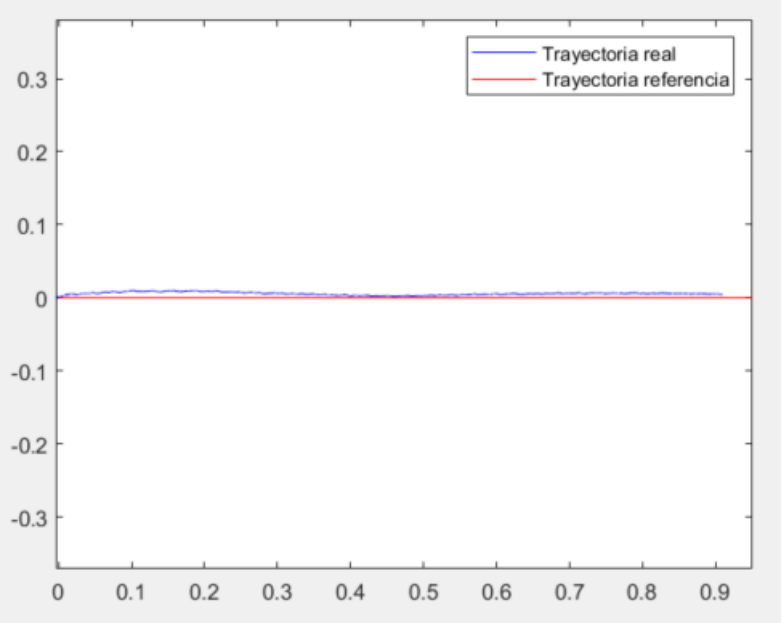
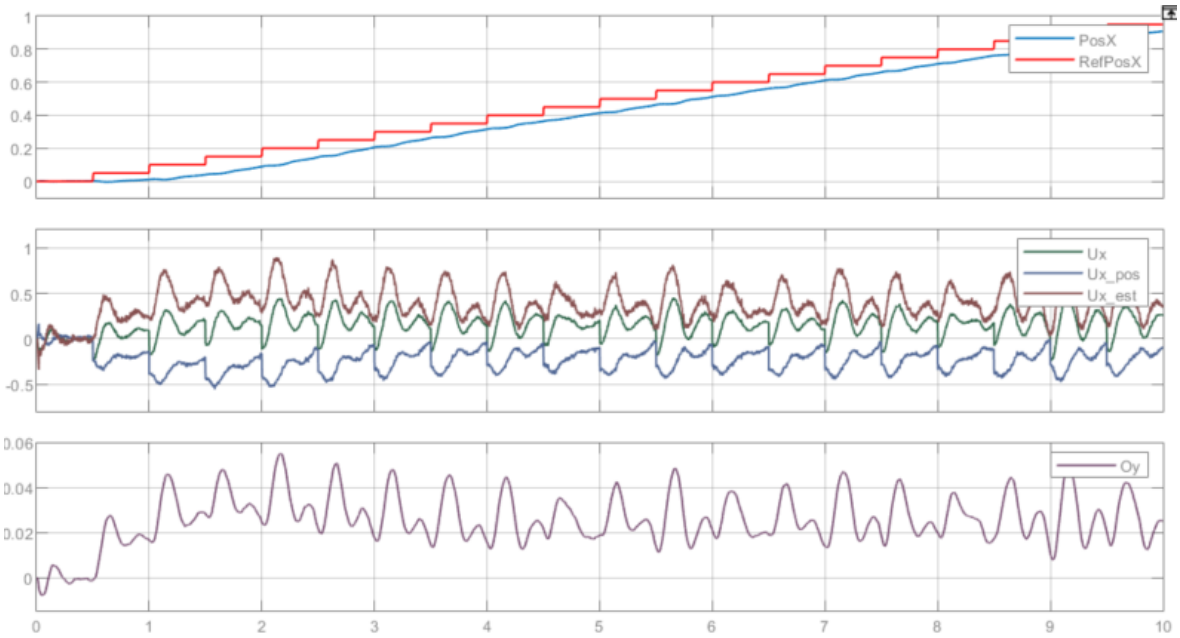


Figura 4.18.- Trayectoria recta

Como se puede ver el resultado es satisfactorio. Se puede analizar más detalladamente con los datos de cada eje (posición vs referencia, acciones de control y ángulo de inclinación del robot):



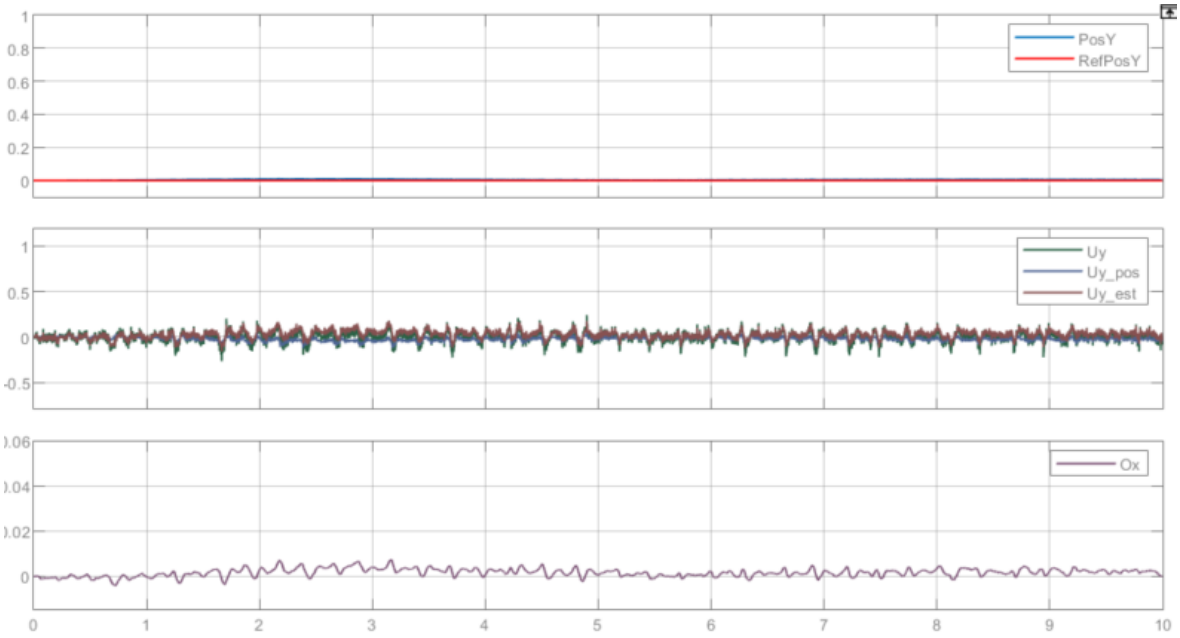


Figura 4.19.- Trayectoria recta. Información detallada

El resultado es muy bueno, prácticamente no se desvía de la trayectoria. Existe un ligero error de posición en el eje X debido a que constantemente se está cambiando la referencia, es una señal tipo rampa. Sin embargo, en el eje Y, que es una referencia constante, se ve un error de posición nulo. Por esta razón, además de que aumentaría la inestabilidad del sistema, se ha juzgado innecesario añadir componente integral al controlador.

A continuación, se va a probar un seguimiento de trayectoria con alguna curva, para evaluar el comportamiento del robot en una situación un poco más exigente. Dado que la trayectoria es más complicada es necesario determinar la forma en que se va a dar la referencia en cada instante. Hasta ahora se estaba cambiando la referencia a intervalos fijos de tiempo, pero para que este método funcione bien, los puntos tienen que estar separados teniendo en cuenta que el robot deberá recorrer la distancia entre cada dos puntos en el mismo tiempo. Esto es cierto para puntos equiespaciados en una línea recta, tal y como se ha hecho anteriormente, pero al introducir curvas debe haber una mayor densidad de puntos cuanto más cerrada sea la curva. Una forma sencilla de conseguir este efecto sin tener que introducir manualmente los puntos uno a uno es utilizar funciones senoidales para la referencia tanto en X como en Y. Se conocen como curvas de Lissajous y permiten crear trayectorias muy distintas simplemente variando la frecuencia de las funciones senoidales utilizadas. A continuación, se muestran los resultados de la prueba junto con la trayectoria escogida (el robot comienza en (0,0) desplazándose en sentido X positivo Y positivo):

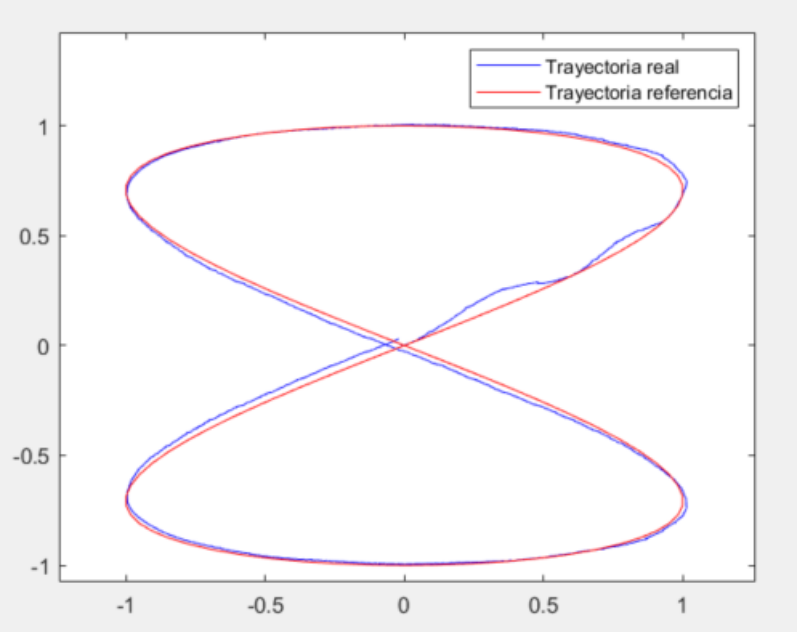
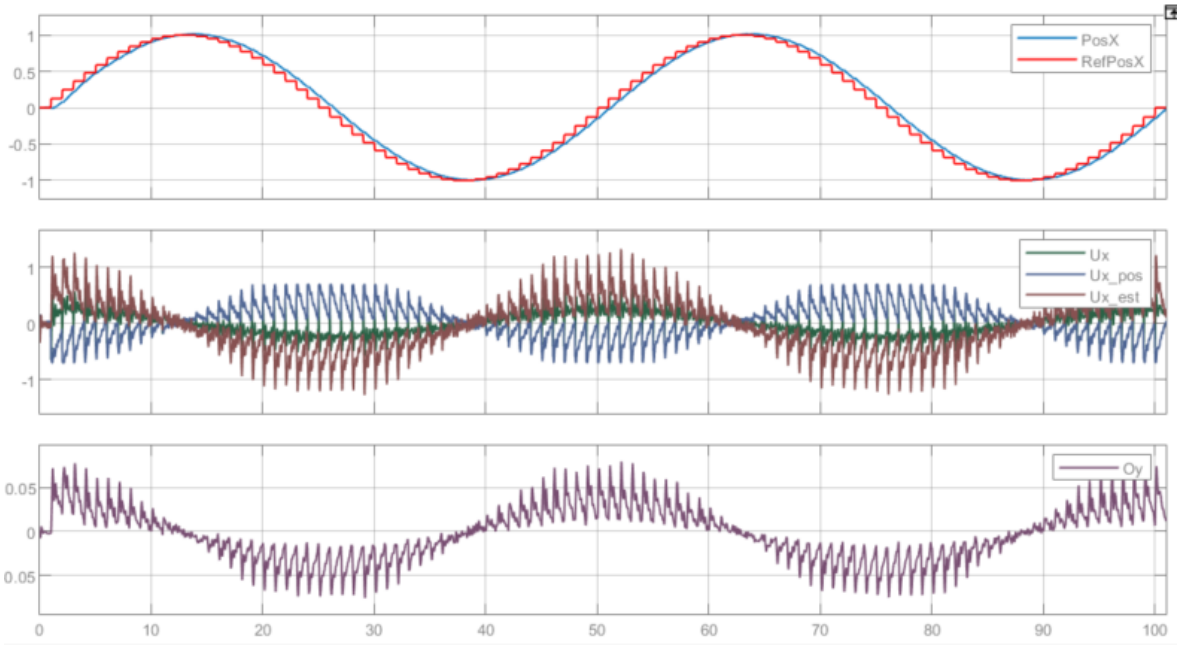


Figura 4.20.- Trayectoria en 8

En este caso el resultado es muy bueno también. El tramo inicial es el que presenta un peor comportamiento, aunque es algo normal teniendo en cuenta que el robot comienza desde parado y hay mucha distancia entre los primeros puntos de la trayectoria. Sin embargo, tras alcanzar al ritmo de la referencia, en la curva superior derecha, el comportamiento es muy fluido y con un error de posición mínimo durante el resto del recorrido, ya que la velocidad del robot es menor. Esto se puede ver con más detalle analizando los datos relativos a cada eje que se muestran a continuación:



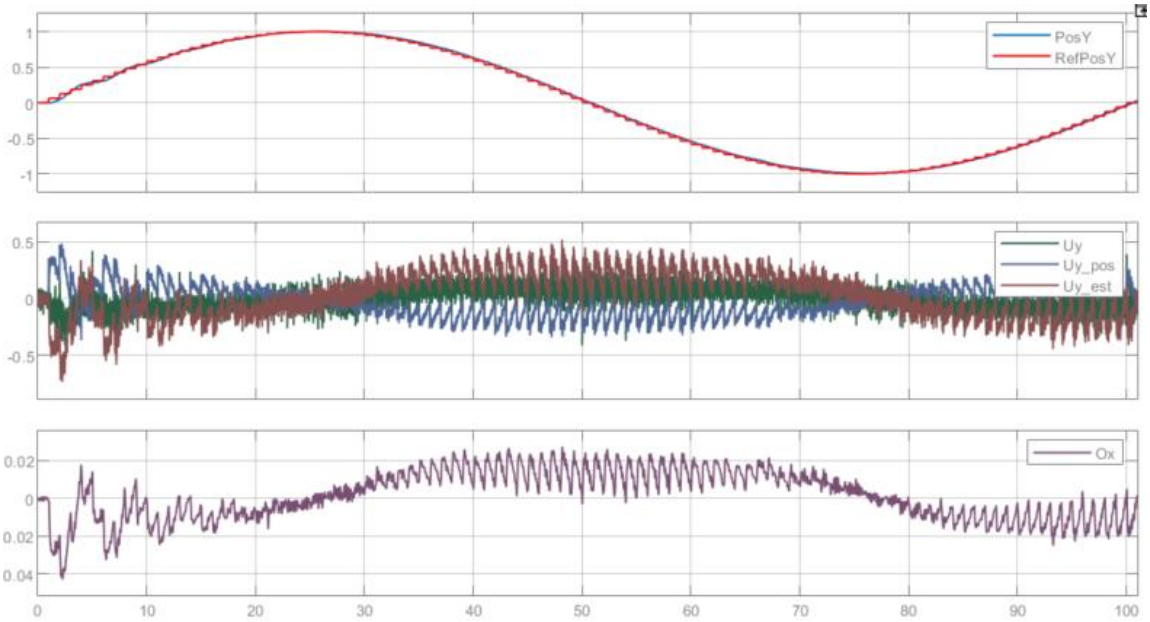


Figura 4.21.- Trayectoria en 8. Información detallada

Se puede ver cómo las acciones de control en el eje X son más grandes, ya que la frecuencia de la onda de referencia de X es el doble que la onda de referencia de Y, llegando incluso a saturar la acción de control de posición para evitar la caída del robot.

Se ha probado también en una trayectoria algo más compleja, y manteniendo el tiempo total de simulación igual. Esto básicamente fuerza al robot a ir más rápido, ya que la trayectoria es más larga y debe ser completada en el mismo tiempo. En la siguiente imagen se puede ver el desempeño del robot:

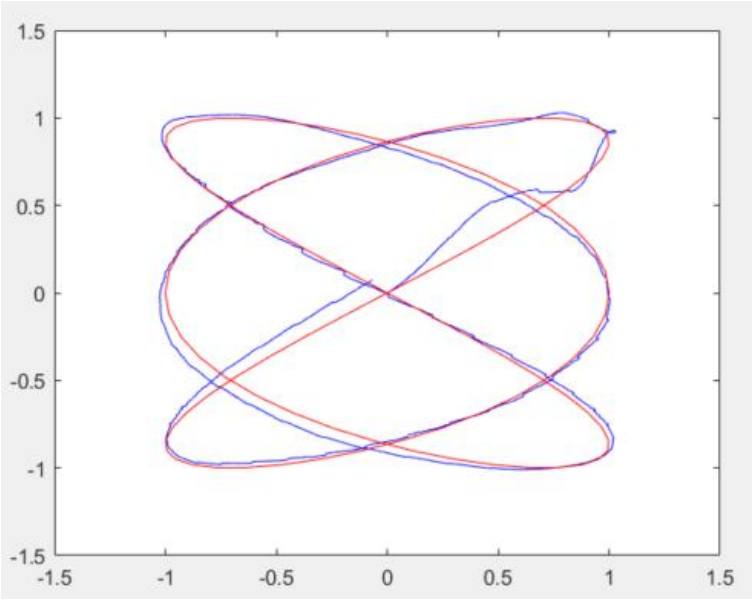


Figura 4.22.- Trayectoria compleja

En efecto, el seguimiento de trayectoria es algo peor, aunque sigue siendo bastante bueno. La parte inicial de la trayectoria sigue teniendo el comportamiento descrito en el ejemplo anterior, al ser la parte en la que el robot va más rápido. En las siguientes gráficas se puede ver más detalladamente los resultados del ensayo:

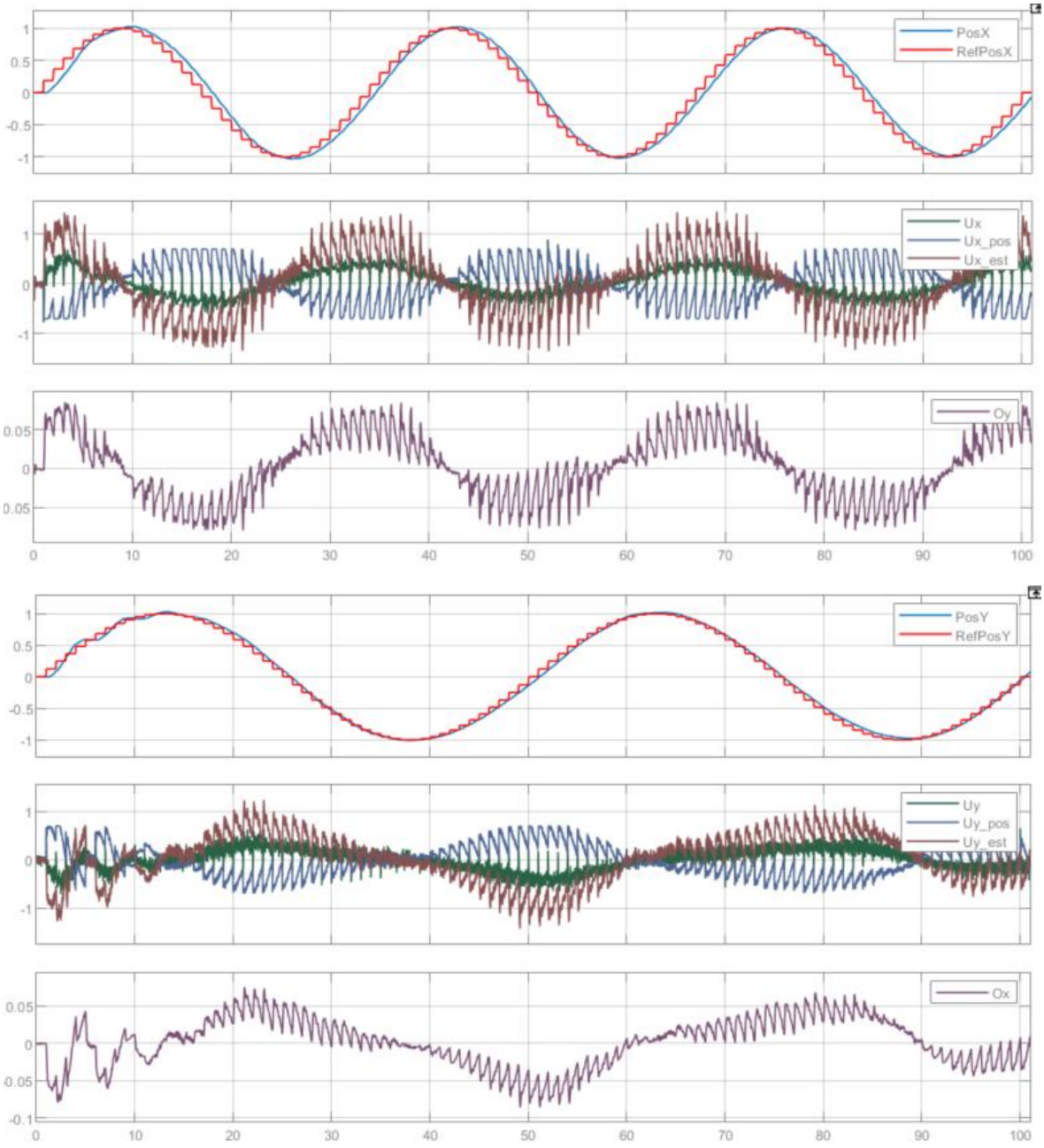


Figura 4.23.- Trayectoria compleja. Información detallada

En este caso se puede ver también cómo la acción de control de posición llega a saturar tanto en el eje Y como en el X, aunque al igual que en la trayectoria anterior el eje X tiene que recorrer más distancia, por tanto, ir a más velocidad, por lo que satura más. Para

intentar mejorar el comportamiento se ha probado a lanzar la simulación manteniendo la misma trayectoria, pero aumentando el tiempo en el que se debe recorrer en un 50%. Esto consigue que el robot vaya más despacio y por tanto con menor error respecto a la trayectoria ideal:

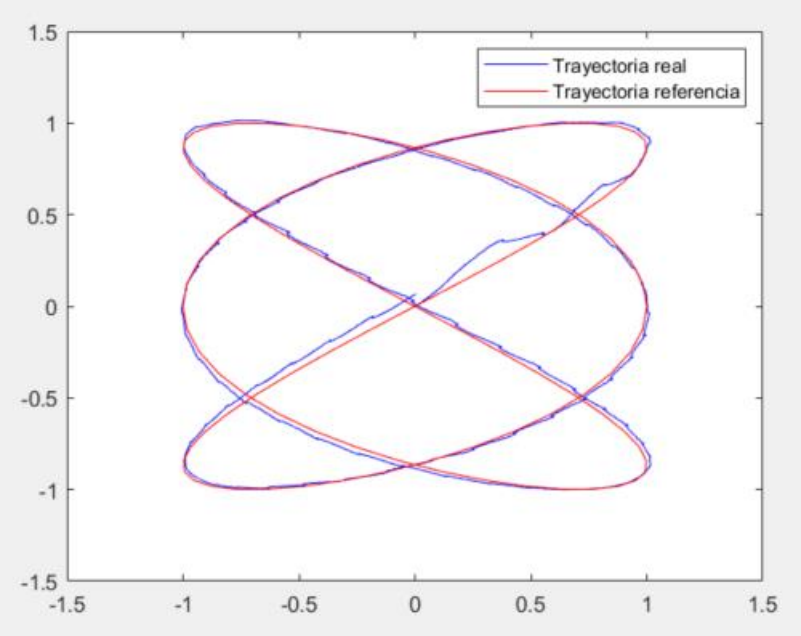
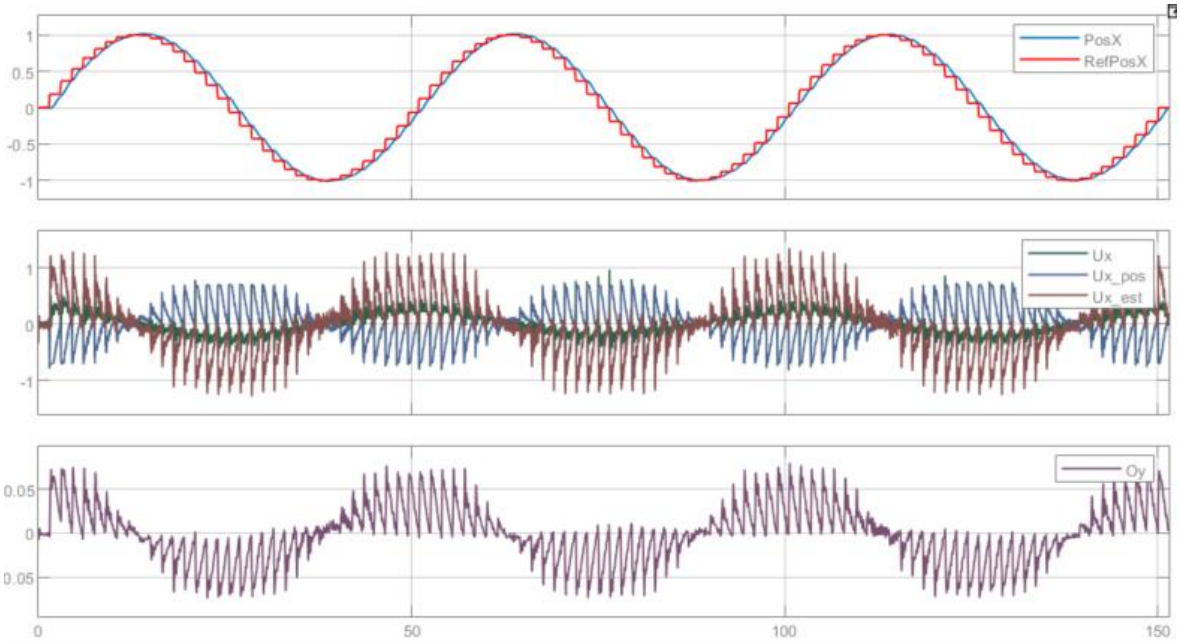


Figura 4.24.- Trayectoria compleja con tiempo aumentado



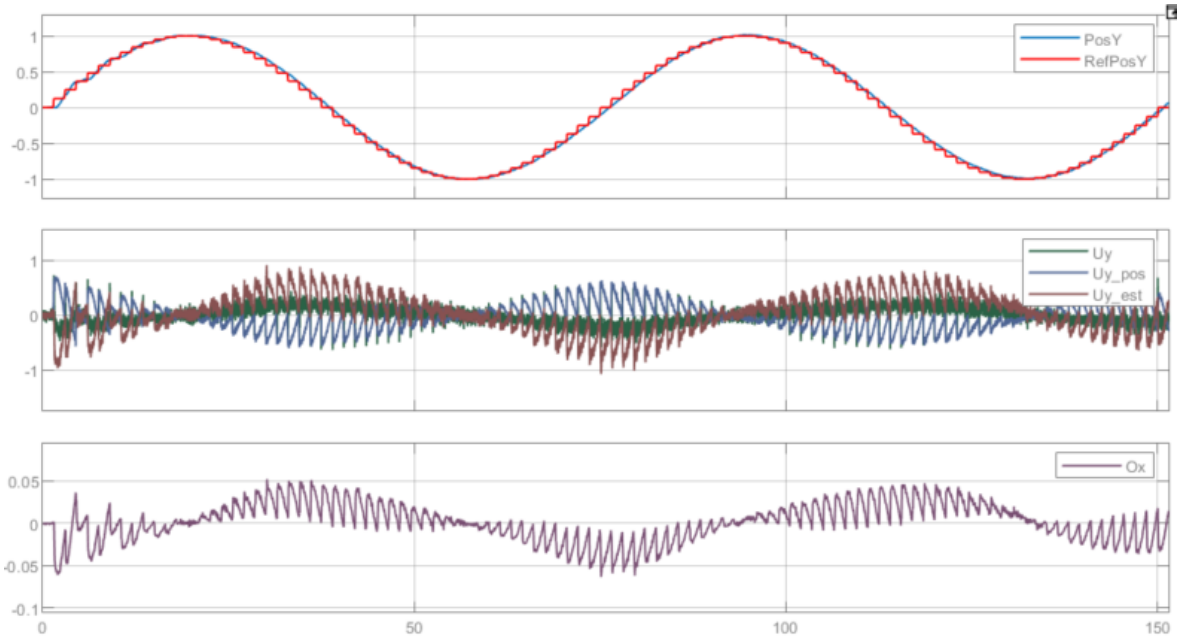


Figura 4.25.- Trayectoria compleja con tiempo aumentado. Información detallada

Se puede ver como al ir más despacio al inicio del movimiento no se desvía tanto de la trayectoria, y también como se llega a saturar considerablemente menos que en el caso anterior. Lo que sí se notan más son los cambios en la referencia, siendo visibles incluso en el gráfico XY.

Dado que se ha habilitado en el archivo Simulink la opción de activar o desactivar el control manual de posición, se ha probado la primera trayectoria también con el control manual para ver la diferencia, y los resultados son los siguientes:

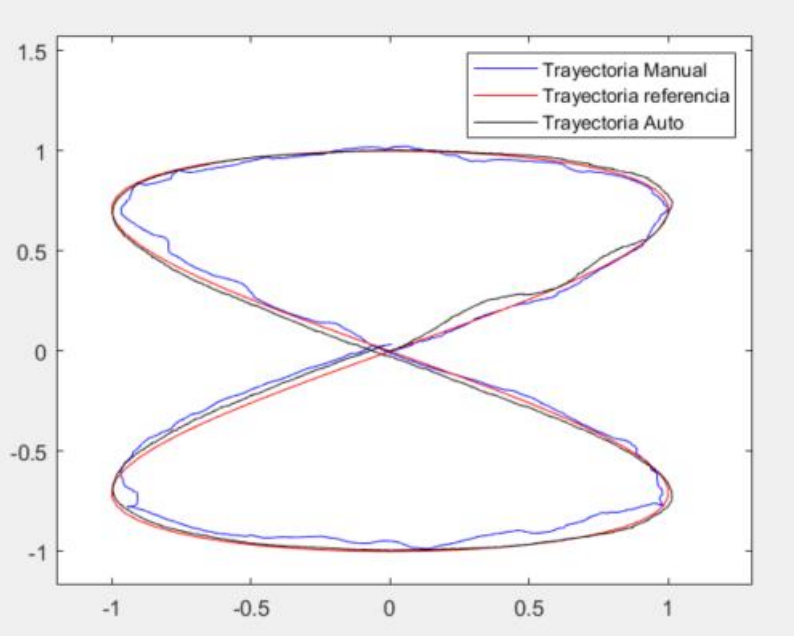


Figura 4.26.- Trayectoria en 8. Comparativa control manual y automático

Claramente el control automático tiene menos error de posición que el manual. Sin embargo, el control manual es considerablemente más rápido. Esto se explica porque se ha limitado la acción de control del controlador de posición para evitar que el robot se incline demasiado, lo que haría que se cayese. Y en cuanto al error de posición del control manual, por supuesto dependerá de la habilidad de la persona que lo maneja, pero en gran parte se debe a que cuando el robot está muy cerca de la trayectoria es complicado ver a ojo si está por encima, por debajo o en la posición correcta.

5. Implementación real

5.1.- CONCEPTO DE DISEÑO

El concepto de diseño del Ballbot está compuesto por dos placas de metacrilato unidas mediante varillas roscadas. En la placa inferior se sitúa el Arduino y el controlador, mientras que en la superior se ha colocado la batería y el IMU. Por su parte, los motores van sujetos mediante sus respectivas bridas a la placa inferior por la parte de abajo a través de unos anclajes.

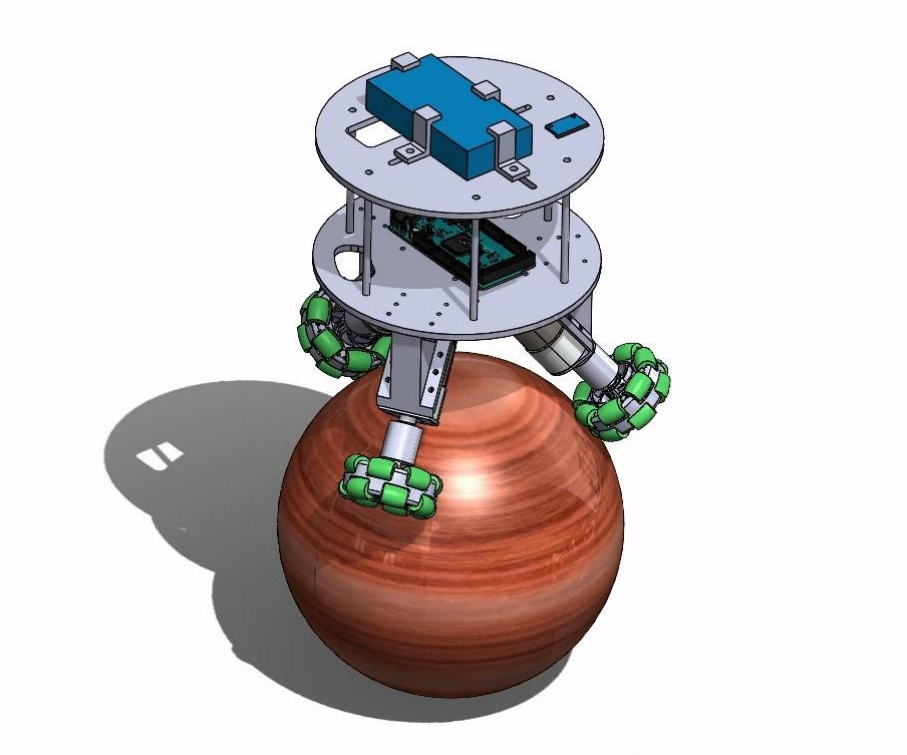


Figura 5.1.- Concepto de diseño del Ballbot

En este diseño mediante SolidWorks no se han incluido el cableado ni la tornillería ya que llevarían bastante tiempo y en realidad no aportan ninguna información adicional.

5.2.- COMPONENTES ELECTRÓNICOS

5.2.1.- Motor

El motor a utilizar será uno de la marca Pololu, de 12V y que incorpora una reductora de 70:1 así como un encoder con resolución de 3200 pulsos por revolución del eje de salida de la reductora (8).

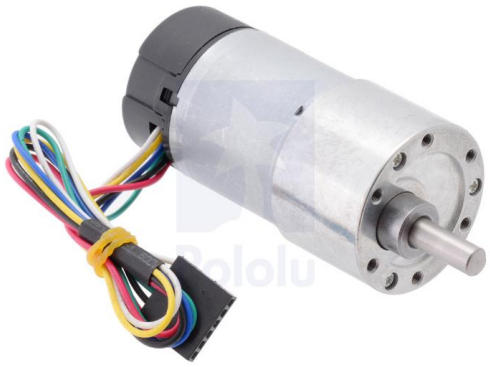


Figura 5.2.- Motor utilizado

Las funciones de cada cable son las siguientes:

- Rojo: Tensión del motor. Conectar al controlador
- Negro: Tensión del motor. Conectar al controlador
- Verde: Masa del encoder
- Azul: Tensión de entrada del encoder (3,5V hasta 20V, requiere como máximo 10 mA)
- Amarillo: Salida 1 del encoder
- Blanco: Salida 2 del encoder

La propia marca ofrece una brida de aluminio (9) para facilitar el anclaje del motor, las cuales se han utilizado también para la construcción del Ballbot.

Figura 5.3.- Bridas para motor

5.2.2.- Microcontrolador

El cerebro del robot será un Arduino Due. Se trata de una versión más potente del archiconocido Arduino Uno y tiene también un mayor número de pines de entrada y salida, así como de comunicación, lo cual será necesario para poder incorporar todos los componentes del Ball robot.

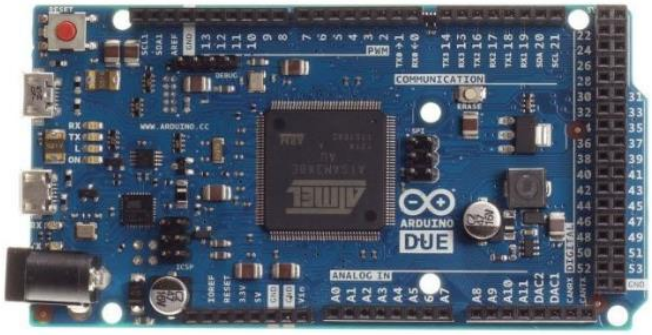


Figura 5.4.- Arduino Due

5.2.3.- Controlador de motor

Se han utilizado tres controladores MZX MSDN470 (10), ya que cada uno es capaz únicamente de controlar un motor. A cambio, este controlador está preparado para alimentar motores DC con intensidades de 20 A, muy superiores a las requeridas por el Ballbot. Además, este controlador no aumenta la zona muerta del motor, permitiendo así al robot realizar pequeñas correcciones. También posee un disipador de forma que no habrá problemas por exceso de temperatura.

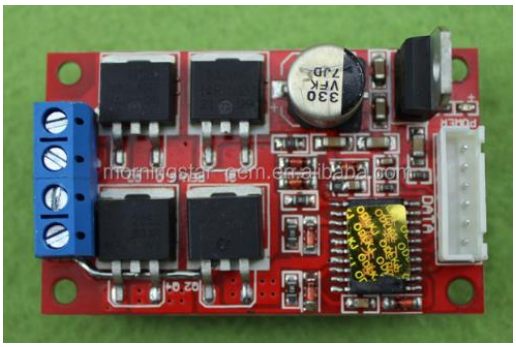


Figura 5.5.- Controlador de motor MZX MSDN470

5.2.4.- IMU (Inertial Measurement Unit)

Para obtener los ángulos respecto a los ejes X e Y se utilizará una unidad de medida inercial, más conocida por sus siglas en inglés IMU. Se ha seleccionado la MPU 9250, que incluye giróscopo, acelerómetro y magnetómetro. Únicamente se buscan los ángulos de inclinación en los ejes X e Y del robot, pero se utiliza toda la información proporcionada por el IMU para, mediante un filtro de Kalman, obtener los ángulos de interés. Esto se hace mediante las librerías MPU9250.h y uNavAHRS.h en el código del Arduino. La comunicación se realiza mediante I2C, aunque la MPU 9250 también permite comunicación mediante SPI.

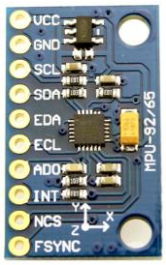


Figura 5.6.- MPU 9250

5.2.5.- Batería principal

Dado que se desea que el robot sea completamente independiente es necesario incorporar una batería para alimentar los drivers. Se ha elegido para ello una batería (11) de 12V, igual que el voltaje máximo de los motores, y una capacidad de 6800 mAh, que debería ser suficiente para hacer funcionar el robot durante el tiempo necesario.

Figura 5.7.- Batería

5.2.6.- Batería secundaria

Debido a que se están utilizando unos drivers cuya salida de 12V no se puede utilizar para alimentar el Arduino (no porque sean 12V sino porque no es estable, en ocasiones se interrumpe y apaga el microcontrolador en plena ejecución), para cumplir el requisito de tener un robot inalámbrico es necesario incluir una segunda batería para alimentar al Arduino. Se ha utilizado una batería portátil genérica, conectada mediante un cable USB.

5.2.7.- Módulo Bluetooth

Para la comunicación con el dispositivo móvil se ha elegido el módulo Bluetooth HC06, que se comunicará mediante puerto serie con el Arduino. En el robot irá colocado a una de las placas mediante cinta adhesiva.

Figura 5.8.- Módulo Bluetooth HC06

5.3.- COMPONENTES MECÁNICOS

Una vez determinados los componentes electrónicos a utilizar y el concepto de diseño, el siguiente paso es elaborar un diseño 3D para poder fabricar las diferentes piezas. Dado que las bridas del motor son componentes comerciales (los vende el propio fabricante del motor) únicamente es necesario fabricar los anclajes del motor, los de la batería y las dos placas de metacrilato.

En cuanto a las placas de metacrilato, se ha elegido metacrilato de extrusión (XS) y un espesor de 5mm dado que la escuela ya disponía de él, y se han elaborado mediante corte láser. Y los anclajes, al no haber grandes cargas en el sistema, más allá del peso propio, no es necesario que posean mucha resistencia, por lo que una simple impresión 3D es suficiente. Concretamente se ha elegido el sinterizado selectivo por láser de polvo de nylon por

comodidad, ya que son piezas pequeñas y se pretende aprovechar la impresión 3D de alguna otra pieza que el taller deba hacer para sinterizar estas también a la vez.

5.3.1.- Bola

Se trata de un componente crucial para el correcto comportamiento del robot, ya que debe agarrar lo suficiente como para que las ruedas no patinen, además de ser rígida para poder aguantar el peso del robot y lisa para facilitar la rodadura. Satisfacer todas estas condiciones es una tarea muy difícil con un presupuesto reducido, por lo que se ha elegido un balón de baloncesto. Cumple con creces el requisito de rigidez, y al estar diseñada para tener contacto con la mano de un jugador el agarre también es excelente. El punto débil es el relativo a la rodadura, ya que todos los balones de baloncesto tienen las características estrías negras, que no dejan de ser pequeñas hendiduras que dificultan la rodadura de los rodillos. Se ha buscado un balón cuyas estrías sean lo menos profundas posible.

Figura 5.9.- Bola de baloncesto

5.3.2.- Ruedas omnidireccionales

Se trata de unas ruedas especiales cuya banda de rodadura está compuesta por una serie de rodillos que pueden girar libremente. De esta forma, las ruedas omnidireccionales permiten libre deslizamiento en la dirección perpendicular a la rodadura (deriva), manteniendo el poder de tracción en la dirección de rodadura.

Figura 5.10.- Rueda omni-wheel

5.3.3.- Acoplamientos rueda-motor

Su función es conseguir que el eje del motor y el eje de la rueda giren solidariamente. Se ha elegido un acoplamiento flexible genérico (12) que la universidad ya tenía de trabajos previos.

Figura 5.11.- Acoplamiento rueda - motor

5.3.4.- Ejes de rueda

Se trata de unos ejes con una parte redonda y otra cuadrada cuya función es que la rueda gire solidaria al acoplamiento. En un principio se utilizaron unos ejes impresos en 3D de otros trabajos fin de estudios, pero tenían una longitud algo superior a la necesaria, lo cual hacía que trabajasen a flexión durante el funcionamiento del robot, llegando a producirse la rotura de uno de ellos. Para evitar que ocurriera lo mismo con los otros ejes se han diseñado e impreso en 3D unos nuevos de longitud inferior, de forma que las ruedas queden más cerca del acoplamiento y así disminuir la carga a la que se ven sometidos los ejes.

Figura 5.12.- Eje de rueda

5.3.5.- Anclajes

Se ha buscado que el ángulo que forman sea el adecuado para que el contacto de las ruedas con la bola sea perfecto para una bola de 23 cm de diámetro (correspondiente al tamaño de un balón de baloncesto). La unión con la placa de metacrilato se realiza mediante tornillos.

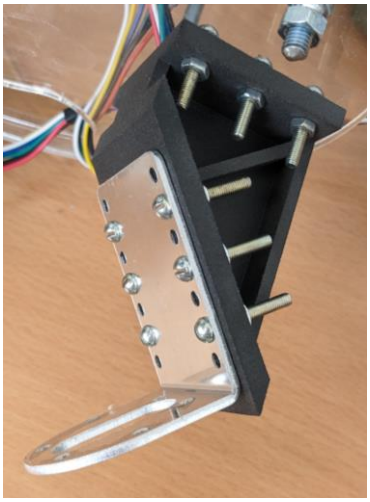


Figura 5.13.- Anclaje

5.3.6.- Placa inferior

Esta placa dispone de agujeros para las varillas roscadas que sujetan la placa superior, agujeros para colocar el Arduino mediante separadores, dos agujeros grandes para poder pasar cables desde los motores hasta el Arduino, y por último agujeros para colocar los anclajes de los motores.

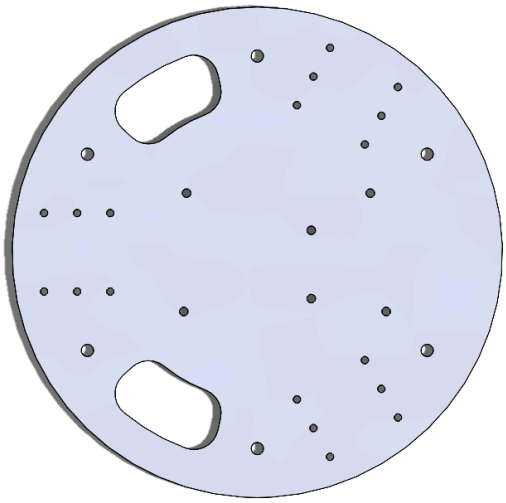


Figura 5.14.- Placa inferior

5.3.7.- Placa superior

En ella se han realizado agujeros para sujetar el acelerómetro mediante tornillos a la propia placa, aprovechando que la superficie inferior es completamente plana. De esta forma se minimiza la posible vibración (y por tanto lectura de ángulos distintos a los reales) causada por colocar separadores u otro soporte.

En la placa superior se encuentra la batería. Tal y como se ha explicado anteriormente, resulta conveniente colocar el centro de gravedad del robot en una posición elevada, razón por la cual se ha utilizado esta configuración. La sujeción de la batería se realiza mediante unas piezas impresas en 3D de forma que la posición de la batería se puede ajustar a voluntad tanto en el eje X como en el eje Y dentro de un rango. La explicación de este sistema de sujeción ligeramente más complejo es que al ser un robot que se basa en el equilibrio es muy importante que el centro de gravedad se encuentre centrado, lo cual resulta más fácil de conseguir si se puede desplazar ligeramente uno de los componentes de mayor peso del sistema.

La placa superior también tiene agujeros para las varillas roscadas que la unirán a la parte inferior del robot y un hueco para pasar los cables de la batería. Se ha tenido en cuenta la posibilidad de que la batería se desplace a la hora de hacer dicho hueco de forma que se garantice que pueda llegar al Arduino. A continuación, se muestra una imagen de la placa superior:

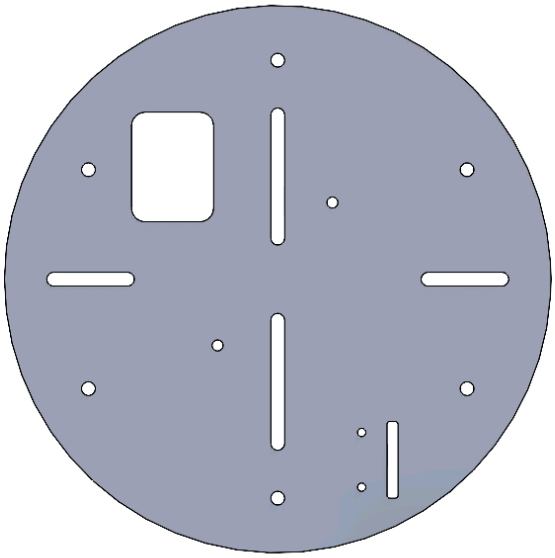


Figura 5.15.- Placa superior

5.3.8.- Anclajes batería

Se trata de cuatro piezas idénticas cuya función, como se ha comentado previamente, es sujetar la batería en la posición deseada. Para ello se apretará el tornillo que las une a la placa superior una vez en la posición correcta. El hecho de que sean cuatro también facilita el proceso de retirar la batería para poder cargarla fuera del robot si fuera necesario, ya que para extraer la batería se deberá aflojar únicamente uno de los anclajes, sirviendo los otros tres para que una vez cargada la batería vuelva exactamente a la misma posición en la que estaba. Respecto al proceso de fabricación, se ha elegido el sinterizado de nylon al igual que los soportes del motor por la misma razón; poder aprovechar para imprimirlas junto con otras piezas.

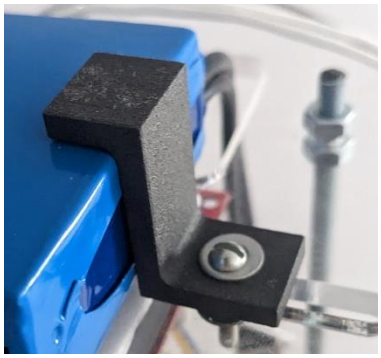


Figura 5.16.- Anclaje batería

5.4.- OTROS COMPONENTES

Esta categoría engloba los demás elementos necesarios para la construcción del Ballbot, tales como cables y tornillería. Si bien no es necesaria una explicación detallada de los mismos ya que se trata de componentes muy simples, es importante mencionarlos para obtener un listado completo de los materiales necesarios.

También se utilizarán dos leds, uno es un led RGB que se utilizará para señalizar que se está realizando el proceso de calibrado (brillará en color morado) y para control de duración de vuelta de bucle, es decir, se encenderá si el Arduino no es capaz de ejecutar el código a la velocidad necesaria (en este caso brillará en color rojo). Se utilizará también un led verde para señalizar que los motores están alimentados.

5.5.- MONTAJE DEL ROBOT

5.5.1.- Montaje final

Para el montaje se ha ido de la parte inferior del robot hacia la superior, comprobando antes de montar que las distintas piezas están bien conectadas y funcionan correctamente (motores, IMU y módulo Bluetooth).

Ha habido algún cambio respecto al concepto de diseño elaborado en SolidWorks. Inicialmente se planteaba que en la placa superior fuese colocado el IMU, y de hecho hay unos agujeros especialmente diseñados para ello, pero se movió a la placa inferior para poder colocarlo de manera centrada, y de esa forma mejorar la lectura del sensor. También hubo un cambio de posición de los controladores ya que en un principio se iba a utilizar un modelo que se montaba directamente sobre el Arduino, pero por razones que se detallarán más adelante en este documento se ha tenido que reemplazar por tres controladores individuales, y el lugar más apropiado para su colocación es la parte inferior de la placa superior, mediante cinta de doble cara. A continuación, se muestra una imagen del robot completamente montado:

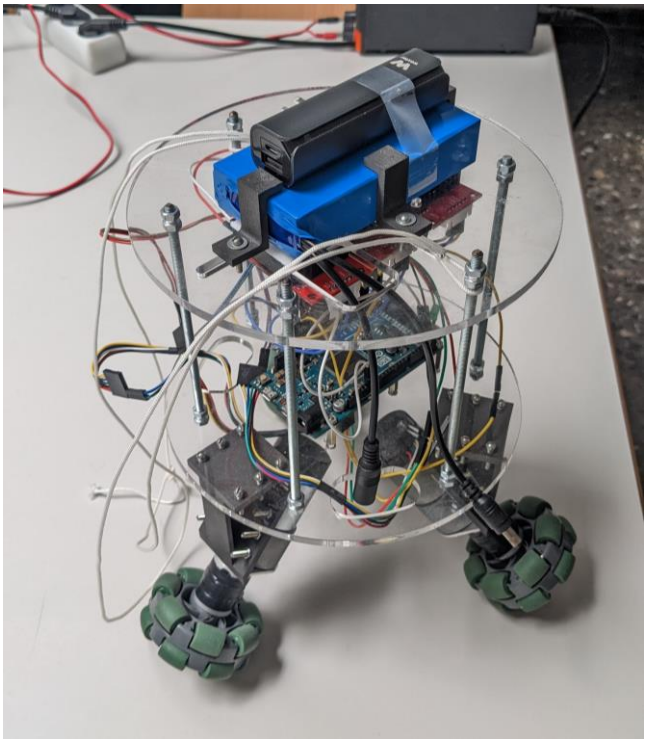


Figura 5.17.- Robot montado

5.5.2.- Ajuste de la posición de la batería

Como se ha comentado previamente es importante que el centro de gravedad del conjunto quede centrado una vez finalizado el montaje. Por ello es necesario realizar un ajuste de la posición de la batería en la placa superior.

Para no complicar las cosas el ajuste se va a realizar en dos partes, primero en un eje y luego en el otro, y se hará sujetando el robot de forma que viendo hacia dónde se inclina el mismo se puede compensar moviendo la batería. No es necesario que el equilibrio sea perfecto, pero sí que esté cerca. Como era de esperar, la posición óptima de la batería es prácticamente centrada.

5.6.- APLICACIÓN MÓVIL

Se va a utilizar una aplicación móvil para controlar la posición del robot, ya que no se ha implementado un control de posición automático. Además, la aplicación servirá para controlar la alimentación de los motores, ajustar parámetros del controlador, entrar en los distintos modos de prueba y establecer los datos a visualizar por el monitor serie del Arduino IDE del ordenador. Para elaborar la interfaz se ha utilizado la aplicación Bluetooth

Electronics, que dispone de varios paneles de control personalizables y ofrece facilidades para la utilización de terminales serie, botones, joysticks y más. En la figura siguiente se muestra el panel de control diseñado:

Figura 5.18.- Panel de control de la aplicación móvil

Se ha diseñado teniendo en cuenta que el móvil estará sujeto por la mano derecha del usuario. Por ello en la parte más cercana a la esquina inferior derecha se encuentran los botones que se pulsan con más frecuencia; el encendido de los motores y los joysticks (posición y giro en Z). En una posición central de la pantalla se encuentran los botones de ajuste de los parámetros del controlador, así como los interruptores de transmisión de datos por puerto serie, que permiten ver los ángulos y velocidades angulares, acciones de control en cada eje y acciones de control en cada rueda. En la parte izquierda se encuentra un terminal serie que permite mandar los comandos no incluidos en ningún otro botón, como por ejemplo la activación del modo de calibración de zona muerta. Por último, se ha incluido un botón amarillo cuya función es establecer los valores iniciales (offset) de los ángulos del robot. La ventaja de tenerlo en el panel de control es que se puede colocar el robot en posición de equilibrio sobre la bola y en ese momento obtener los ángulos, que es un procedimiento más preciso que simplemente establecer los valores iniciales tras el arranque del robot.

Los distintos botones descritos previamente se han configurado en la app Bluetooth Electronics para comunicarse con el Arduino mediante el envío de un comando por Bluetooth. Concretamente los comandos incorporados son los siguientes:

Comando	Descripción
m	Activa los motores
p	Desactiva los motores
vi	Obtener los valores iniciales de los ángulos
calxy/calxyoff	Activar/desactivar el modo calibrar ejes
calzm/calzmoff	Activar/desactivar el modo de calibración de zona muerta
solox	Activar el modo de control únicamente en eje X (la acción de control en eje Y será siempre nula)
soloy	Activar modo de control únicamente en eje Y (la acción de control en eje X será siempre nula)
dosejes	Activar modo de control en ambos ejes
a	Activar transmisión por puerto serie de ángulos y velocidades angulares del robot
ae	Activar transmisión por puerto serie de las acciones de control en cada eje (parte proporcional, parte derivativa, parte integral y total para cada eje)
ar	Activar transmisión por puerto serie de las acciones de control de cada motor
ns	Desactivar todas las transmisiones por puerto serie
kpu/kpd	Aumentar/disminuir el valor de Kp en 0,02
kdu/kdd	Aumentar/disminuir el valor de Kd en 0,002
kiu/kid	Aumentar/disminuir el valor de Ki en 0,2
vdu/vdd	Aumentar/disminuir el número de vueltas de bucle que se espera para mandar datos (para usar con el serial plotter del Arduino IDE como si fuese el ajuste de eje tiempos)
zmu/zmd	Aumentar/disminuir el valor de la zona muerta en 0,1V
acu/acd	Aumentar/disminuir la acción de control aplicada a los motores en el modo calibración de zona muerta en 0,1V
acuu/acdd	Aumentar/disminuir la acción de control aplicada a los motores en el modo calibración de zona muerta en 0,5V

cczu/cczd	Aumentar/disminuir el coeficiente de corrección Z en 0,3 (este coeficiente es el que sirve para añadir un giro en Z proporcional a la acción de control del eje Y)
auyu/auyd	Aumentar/disminuir el ajuste Uy en 0,1 (este coeficiente es el que sirve para que al robot se desplace a la misma velocidad en los ejes X e Y ante una misma acción de control)

Tabla 5.1.- Comandos Bluetooth

En cuanto a los joysticks, se han configurado para que si están pulsados manden cada 100 ms la posición en la que están de la manera "X12Y34" donde 12 representa la posición en el eje horizontal y 34 en el eje vertical. Para diferenciar el joystick que controla el giro sobre el eje Z se ha configurado de tal manera que manda la información precedida de la letra "r", es decir, de la forma "rX56Y78", y se utiliza únicamente el valor de la posición en el eje horizontal como señal de giro en Z a mandar al robot. El valor de la posición en el eje vertical es irrelevante.

5.7.- CÓDIGO DE CONTROL

Para la elaboración del código se ha utilizado el Arduino IDE, que además del entorno de programación permite el uso de un monitor serie y un plotter, que permitirá ver la evolución de distintas variables de interés como pueden ser ángulos o acciones de control. Para usar el plotter se deben mandar los distintos datos a graficar separados por un tabulador ("\t") y añadir un fin de línea ("\n") al final del último dato.

5.7.1.- Librerías utilizadas

A la hora de establecer la comunicación con algunos componentes no tiene sentido programar el código desde cero ya que es un problema al que otras personas ya se han enfrentado y existen librerías específicas para esta tarea. Concretamente en este proyecto se ha utilizado la librería uNavAHRS.h (uNav Attitude and Heading Reference System) para la comunicación con el IMU, así como un filtrado de la señal mediante un filtro de Kalman extendido, utilizando los acelerómetros y magnetómetros además del giroscopio para dar una medida más precisa.

Para el correcto funcionamiento de la librería uNavAHRS hay que instalar también las librerías MPU9250.h y Eigen.h.

5.7.2.- Estructura del código

Se ha buscado utilizar funciones para evitar poner muchas líneas de código en el setup y el loop. De esta forma resulta más sencillo encontrar las distintas partes del código para realizar modificaciones al mismo. También se ha procurado dejar abundantes comentarios, que siempre son útiles cuando se quiere modificar el código. La estructura general del código se muestra en el siguiente flujograma:

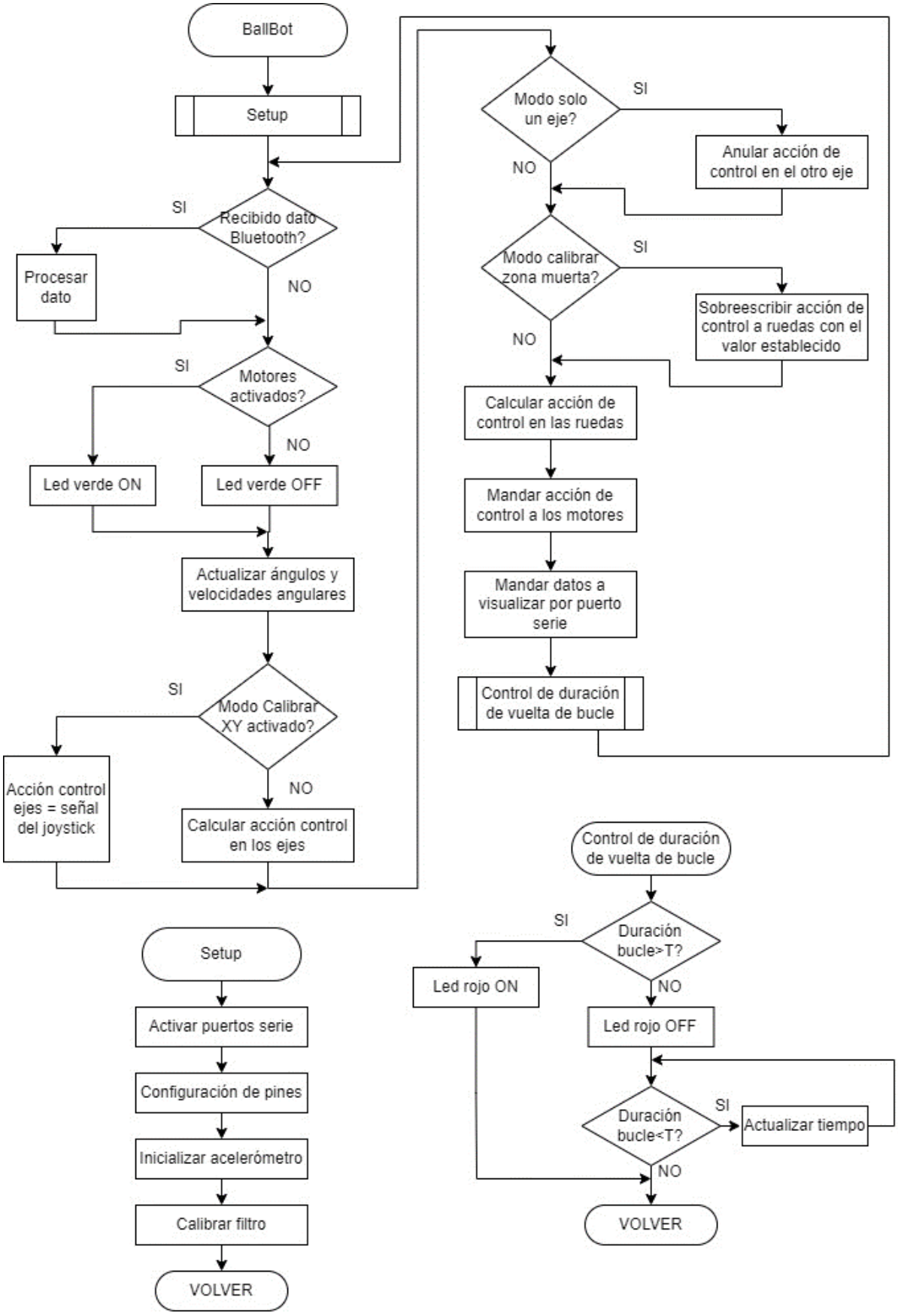


Figura 5.19.- Flujograma del código de control

5.7.3.- Modos especiales de funcionamiento

Se han incluido algunos modos de funcionamiento específicos para realizar pruebas concretas. A continuación, se ofrece una breve descripción de cada uno de ellos:

- Modo de calibración XY: Este modo anula las acciones de control relativas a la estabilidad, mandando a los motores únicamente la acción de control determinada por el usuario mediante el joystick.
- Modo de calibración de zona muerta: Este modo manda a los tres motores una misma acción de control, por lo que permite ver diferencias en su comportamiento, principalmente el tamaño de su zona muerta.
- Modo de control solo en eje X / solo en eje Y: Pensado para facilitar la calibración del controlador al eliminar completamente la acción de control relativa a uno de los ejes. Sin embargo, resultó no ser muy útil ya que requiere acción constante por parte del usuario para evitar que el robot caiga en el eje no controlado, lo cual complica la realización de pruebas.

5.8.- PRUEBAS INICIALES

Antes de probar el robot en movimiento se ha juzgado necesario disponer de un sistema que evite la caída del robot al suelo, evitando de esta forma posibles daños. Para ello se ha utilizado una cuerda, que atada a la parte superior del robot servirá para que el usuario la pueda sujetar, dejándola normalmente suelta, pero pudiendo tirar de ella para así levantar el robot si ve que hay riesgo de caída.

Como primera prueba se ha juzgado conveniente evaluar el comportamiento de los motores ante la aplicación de un determinado voltaje a los mismos, para detectar posibles diferencias y poder compensarlas mediante software. Para ello se ha utilizado la aplicación móvil, activando el modo de calibración de zona muerta y midiendo mediante un voltímetro la tensión real aplicada a cada motor. Los resultados se recogen en la siguiente gráfica:

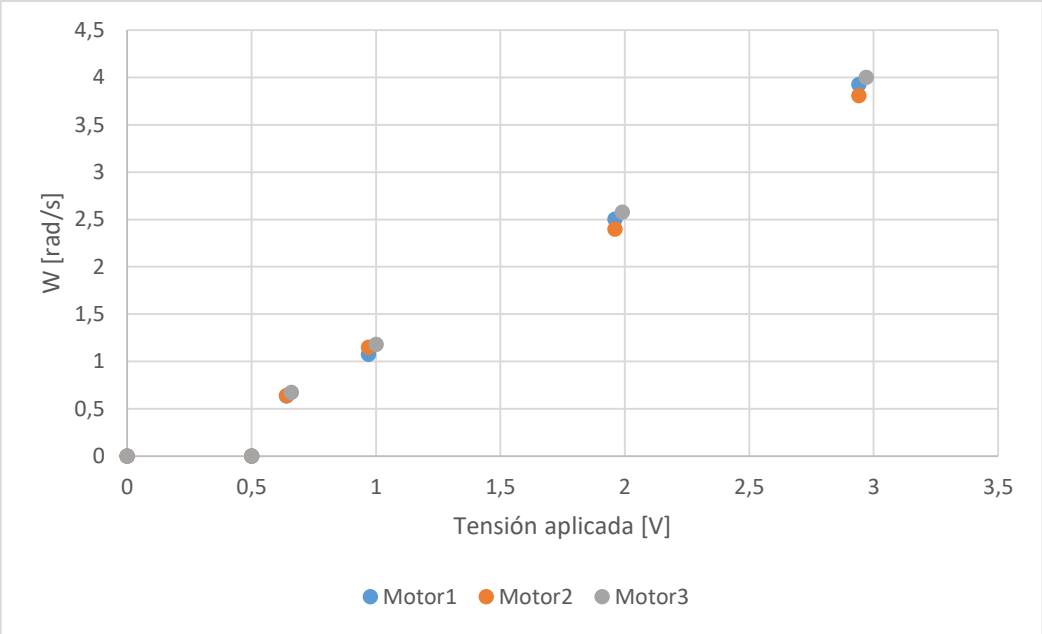


Figura 5.20.- Comportamiento de los motores

Como se puede ver el comportamiento de los tres motores es prácticamente idéntico, por lo que no será necesario hacer ningún ajuste adicional. Asimismo, la zona muerta se sitúa en 0,6V, lo cual corresponde con los datos que se conocían de los motores. El siguiente paso consiste en comprobar que las acciones de control en cada eje se corresponden en efecto con un movimiento en esa dirección. Cuando se estaba realizando esta misma prueba en la etapa de simulación se restringió el giro del robot, de forma que era imposible que se cayera, pero con el robot físico esto no es posible, por lo que la única solución es colocar el robot boca abajo con la bola encima. Esta solución no es tan buena, ya que a simple vista no es tan fácil detectar si el movimiento es correcto, pero aun así se puede realizar la comprobación grosso modo. Para ello, se ha activado el modo calibración XY desde la aplicación móvil.

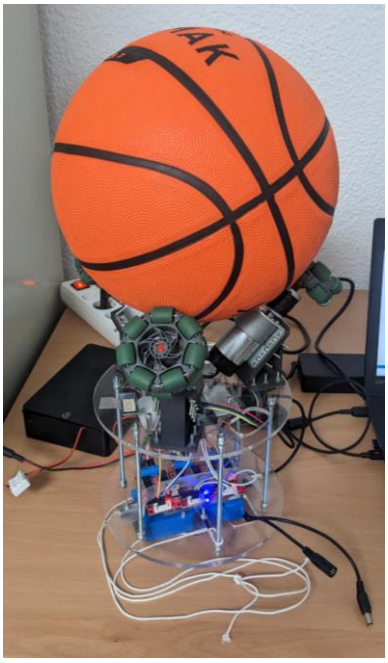


Figura 5.21.- Comprobación de acciones de control en X e Y

Como era de esperar, el resultado es el mismo que en la simulación; el movimiento en el eje X es correcto pero el movimiento en el eje Y no. Para solucionarlo se ha actuado una vez más igual que en la etapa de simulación, es decir, añadiendo un giro en el eje Z proporcional a la acción de control en Y, y también un coeficiente de corrección de forma que ante el mismo valor de acción de control el robot se desplace a la misma velocidad en ambos ejes. Así los parámetros del controlador serán los mismos para ambos ejes.

5.9.- CALIBRACIÓN DE CONTROLADOR

Una vez realizadas las pruebas de funcionamiento iniciales es momento de calibrar el controlador. La idea es comenzar con Kd y Ki nulas, e incrementar Kp hasta que al dejar caer el robot hacia un lado el controlador sea capaz de corregir, aunque se llegue a caer por el otro lado. Una vez conseguido esto, se debe aumentar la Kd para amortiguar esa sobreoscilación. La parte integral no debería ser necesaria ya que el control manual de posición es el encargado de corregir la tendencia del robot a desplazarse en una dirección, aunque se incluye en el código para poder probar su efecto.

Por comodidad, la calibración se ha realizado con el Arduino conectado al ordenador para ver datos por puerto serie y con drivers alimentados por un transformador, y no a la batería, para no tener que estar pendiente del estado de carga de la misma. También se ha

aprovechado la aplicación móvil diseñada para facilitar este proceso, permitiendo modificar los valores de todas las constantes durante la ejecución del código, así como comprobar que las acciones de control no saturan.

Tras numerosas iteraciones se ha llegado a los valores Kp = 0,58 y Kd = 0,014. La imagen siguiente muestra el proceso de calibrado del controlador, con el cable de conexión del Arduino con el ordenador y el cable de alimentación de los drivers:

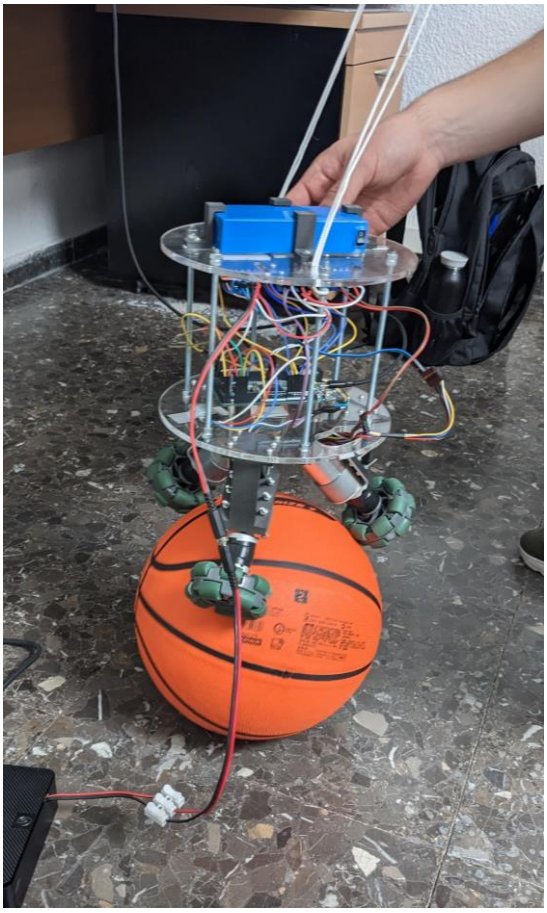


Figura 5.22.- Calibración controlador

5.10.- INCIDENCIAS

5.10.1.- Cambio de bola

En un principio se probó con una pelota de pilates, ya que su superficie es completamente lisa excepto el tapón de la válvula, de forma que los rodillos de las ruedas omnidireccionales podrían girar sin atascarse, y la superficie debería agarrar lo suficiente también.

Figura 5.23.- Pelota de pilates utilizada inicialmente

Sin embargo, ya en las primeras pruebas se vio que el robot se hundía demasiado en la bola, y en ocasiones las ruedas resbalaban al girar demasiado rápido. Por esta razón se acabó utilizando una pelota de baloncesto, que ofrece más agarre y rigidez, aunque presenta los característicos surcos que no favorecen la rodadura. En las sucesivas pruebas se ha visto que, si bien al rodar por encima de los surcos en ocasiones pega un ligero bote, en general presenta un comportamiento mejor que con la pelota de pilates.

5.10.2.- Cambio de controlador de motor

Inicialmente se estaba usando el controlador DK Electronics L293D Motor Shield (13), un controlador de 4 motores diseñado para ser utilizado con Arduino Due (aunque se necesiten controlar únicamente 3 motores). Se trata de una placa diseñada para controlar motores de corriente continua de entre 4,5V y 25V, y es capaz de proporcionar una corriente máxima de 600mA. Además de que la corriente máxima no es muy elevada, durante la calibración de controladores se observó que tardaba demasiado en reaccionar, es decir, no era capaz de hacer pequeñas correcciones cuando el ángulo era pequeño, y si se intentaba subir la parte proporcional del controlador el sistema se volvía inestable. Se hizo una caracterización de uno de los motores con este controlador para obtener su zona muerta, y salía que era de unos 2V, según se ve en la siguiente gráfica:

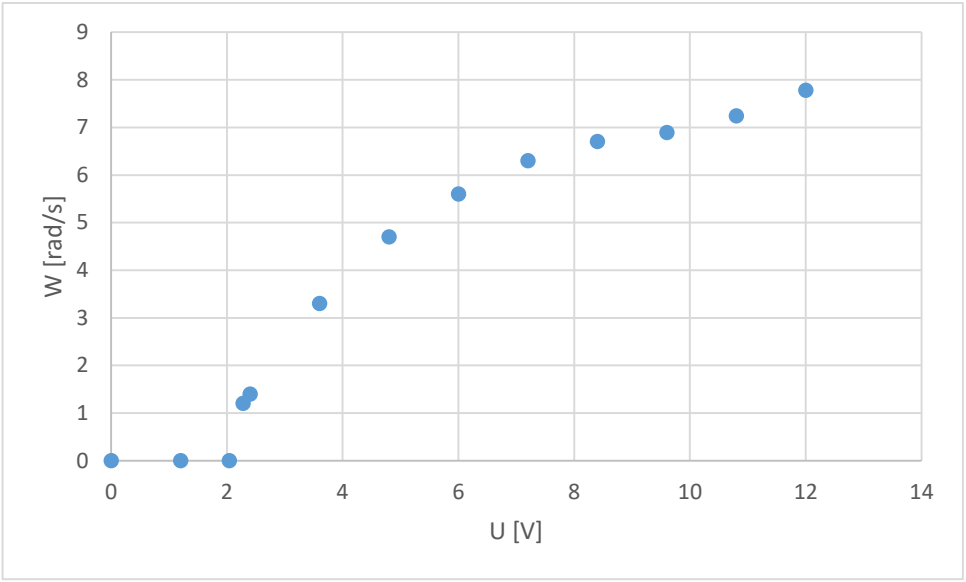


Figura 5.24.- Caracterización motor con driver L293D

A simple vista se puede ver que la zona muerta es excesivamente grande, además de que para acciones de control superiores al valor de 2V correspondiente a la zona muerta el comportamiento no es lineal, cuando debería serlo. Aun así, se probó a implementar una compensación de zona muerta, pero el robot no era capaz de mantener el equilibrio por lo que se cambió al controlador Motor Shield V2.0 (14). Con este driver se volvió a realizar una caracterización del motor, y en esta ocasión los resultados sí se corresponden con lo que cabía esperar:

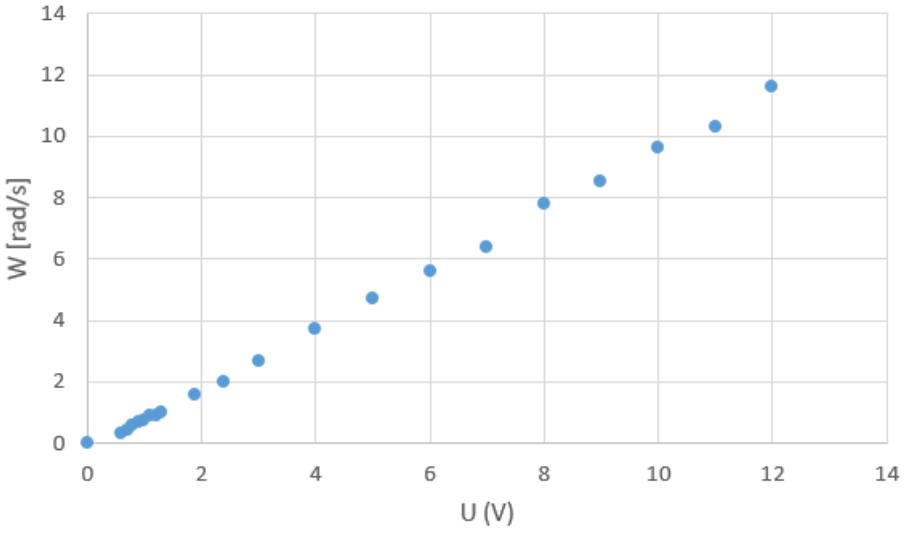


Figura 5.25.- Caracterización motor con driver MotorShieldV2.0

El comportamiento es lineal y la zona muerta es de 0,6V. Sin embargo, realizando esta misma prueba con los tres motores a la vez se ha observado que el motor 1 gira considerablemente más deprisa que el 2, y este a su vez gira algo más rápido que el 3. Concretamente, para una acción de control máxima, es decir, 12V, el motor 1 gira a 12 rad/s, el motor 2 a 10,8 rad/s y el motor 3 a 9,2 rad/s. Además, durante las pruebas para ajustar el controlador de estabilidad se quemó el driver, por lo que fue necesario realizar otro cambio.

En esta ocasión se ha optado por utilizar dos drivers L298N (15), ya que poseen un disipador mayor que debería eliminar los problemas de temperatura por exceso de corriente. Se utilizan dos drivers porque cada uno únicamente es capaz de controlar dos motores, y para esta aplicación es necesario controlar tres. Sin embargo, una vez más, el comportamiento de los motores no es adecuado, presentando una zona muerta aún mayor que con el L293D, ya que hasta 3,1V no se encendían los tres motores. Además, ante la misma acción de control, el voltaje aplicado a cada motor variaba considerablemente, como se puede ver en la siguiente gráfica:

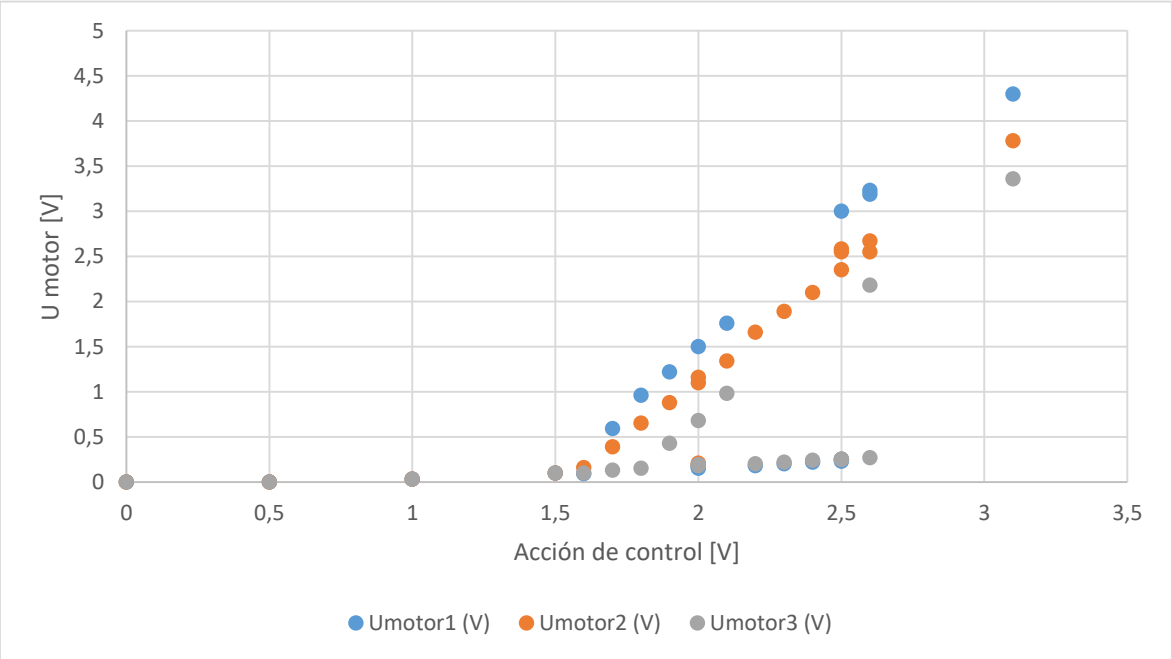


Figura 5.26.- Caracterización motores con driver L298N

A la vista de este comportamiento se ha juzgado necesario realizar otro cambio de driver. El modelo seleccionado es el MZX MSDN470 (10) y se trata de un driver de un solo motor, por lo que son necesarias tres unidades. Este driver está preparado para alimentar motores DC con intensidades de 20 A, muy superiores a las requeridas por el Ballbot, y el

mayor inconveniente de su utilización es encontrar un hueco lo suficientemente grande en el robot para poder colocarlos. La ubicación utilizada ha sido la parte inferior de la placa superior, de forma que queden lo más centrados posible. Por supuesto, antes de montarlos se ha hecho una prueba para comprobar que el comportamiento es el adecuado, es decir, que a los motores se les aplica la acción de control deseada. En efecto eso es lo que se observa en los resultados de la prueba:

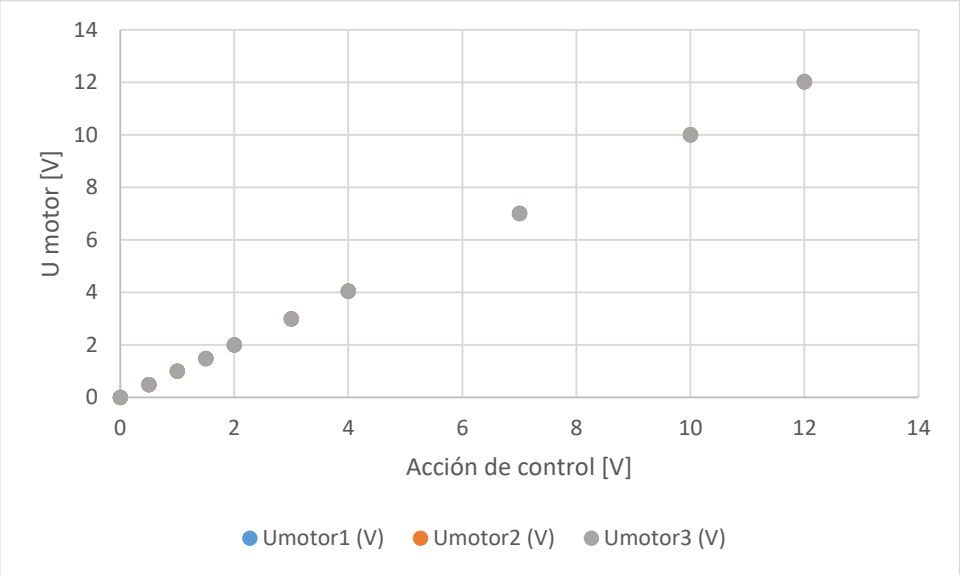


Figura 5.27.- Caracterización motores con driver MZX MSDN470

De hecho, como las acciones de control de los tres motores son idénticas, los puntos de la gráfica se superponen. Eso explica por qué se ven únicamente los puntos grises correspondientes al motor 3. La zona muerta de los motores es ligeramente inferior a 0,6V.

5.11.- RESULTADOS

Se ha conseguido que el robot tenga un comportamiento bastante correcto, realizando acciones correctivas cuando detecta que se está cayendo. Sin embargo, no se ha conseguido que se mantenga en equilibrio sin contacto con el usuario, si bien es cierto que se requiere únicamente un ligero contacto con el robot, lo suficiente para reducir un poco la velocidad del sistema, para que el control de estabilidad sea capaz de mantener el robot en equilibrio.

Por supuesto se han intentado realizar diversos cambios para intentar mejorar el resultado. Lo primero que se ha intentado al ver que el controlador era demasiado lento es subir Kp y Kd. En ambos casos el comportamiento empeora ya que se producen oscilaciones

cuando el robot se encuentra en la posición de equilibrio, oscilaciones que van aumentando hasta hacer que el robot se caiga. Se ha probado también a introducir acción integral pero el sistema se vuelve más inestable, aun incluyendo antiwindup. Otra idea que se ha probado es calibrar el controlador solo en una dirección, es decir, activando el modo que únicamente aplica acción de control a uno de los ejes. El problema en este caso es que al no poder restringir físicamente el movimiento del robot en el otro eje hay que estar sujetándolo para que no se caiga, por lo que no se puede calibrar correctamente el controlador de esta manera.

Se ha probado también a modificar otras partes del código, no solo el controlador. Por ejemplo, se ha probado a reducir el tiempo de vuelta de bucle hasta el mínimo posible (siempre garantizando que se cumple, por supuesto), siendo este mínimo de 8 ms en vez de los 10 ms utilizados inicialmente. Sin embargo, no se ha observado una mejoría notable. Se ha probado a implementar una compensación de zona muerta de los motores, lo cual sí que ha introducido una pequeña mejoría en el comportamiento, y también se ha cambiado la rutina de calibración para que la toma de valores iniciales de los ángulos ocurra a petición del usuario desde un botón del panel de control del móvil. De esta forma se puede buscar manualmente el punto de equilibrio del robot sobre la bola y establecer esos ángulos como los iniciales, eliminando así el pequeño error que resultaba de hacer la calibración con el robot en posición horizontal. Este cambio también ha afectado positivamente al comportamiento del robot, aunque aún después de todas las modificaciones comentadas, el robot ha seguido siendo incapaz de mantener el equilibrio sin nada de ayuda.

Por último, se ha probado a modificar físicamente el sistema, en vista de que las modificaciones de software no son suficientes para lograr el comportamiento deseado. Dado que el problema parece ser que el sistema es demasiado rápido, las modificaciones han ido buscando hacer el sistema más lento. Primeramente, se ha buscado aumentar la inercia de la bola, para que ruede más despacio. Se ha probado a rellenar la pelota de pilates utilizada inicialmente de agua, pero al no poder rellenarse completamente y haber un poco de aire, el agua se desplaza durante la rodadura haciendo que esta vaya a golpes, imposibilitando así el equilibrio del robot. Se ha probado también con una solución un poco más profesional; la utilización de un balón medicinal.

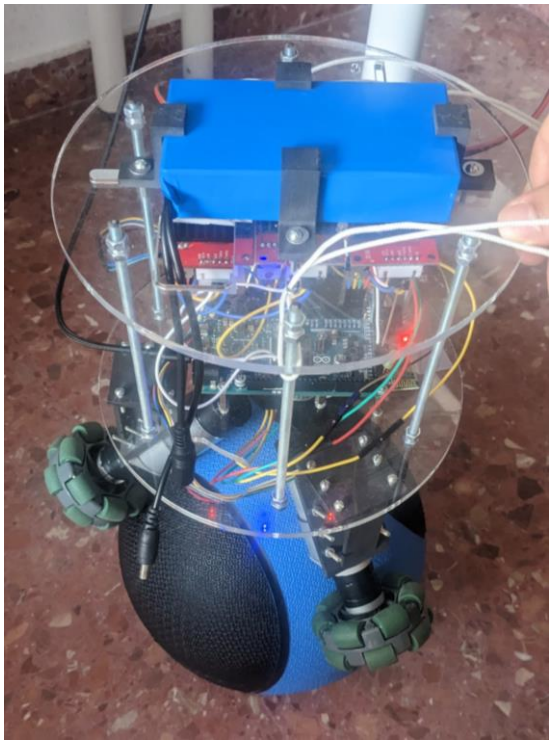


Figura 5.28.- Ballbot sobre balón medicinal

Sin embargo, el recubrimiento exterior del balón es más resbaladizo que el del balón de baloncesto, y las ruedas patinan al intentar corregir la posición del robot, por lo que también se ha descartado la idea. Queda por tanto únicamente la posibilidad de aumentar la inercia del robot, para que caiga más lentamente. Para ello se puede aumentar su masa o desplazarla hacia la parte superior. Esto puede resultar un poco anti intuitivo, pero es similar a por qué es más fácil equilibrar en la palma de la mano un paraguas que un lápiz; el lápiz es demasiado rápido. La solución a probar en el robot ha sido incorporar una pesa a su parte superior, contribuyendo así tanto al aumento de la masa como a la elevación del centro de gravedad.

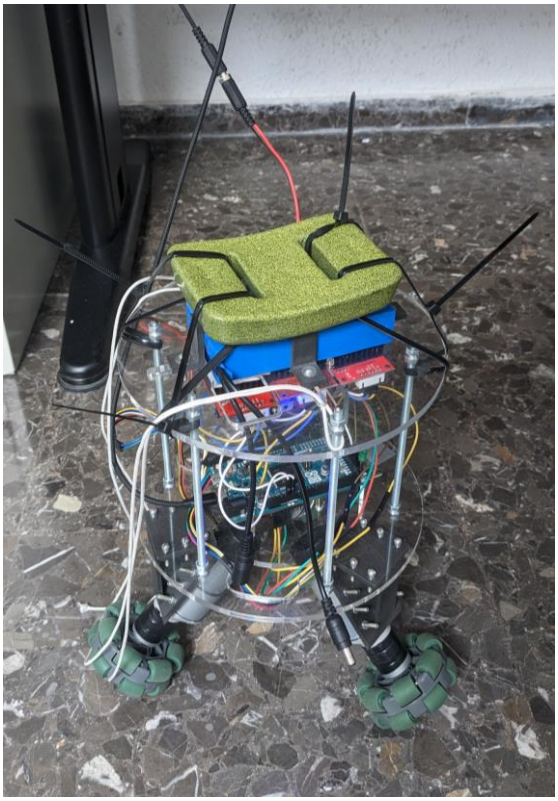


Figura 5.29.- Ballbot con pesa

Sin embargo, al probarlo se ha visto que el peso era demasiado grande, ya que el acoplamiento deslizaba, lo cual hacía que, aunque el eje del motor girase, el eje de la rueda se mantuviese parado. Además, no se ha notado una reducción significativa de la rapidez del sistema.

Por último, aunque la lectura de ángulos parecía ser correcta, se ha probado a mover el IMU de su posición original en la placa superior a una posición centrada en el robot, por si las vibraciones y movimientos del robot afectaban a la lectura de ángulos. Si se utilizase únicamente el giroscopio no debería haber problema con la posición del IMU en el robot, pero dado que se utiliza un filtro de Kalman con la información del acelerómetro se pensó que sería conveniente una posición más centrada para evitar posibles errores. Tras realizar el cambio no se notó un cambio significativo en el comportamiento del robot.

6. Pliego de condiciones

6.1.- OBJETO

Este documento tiene como objeto determinar los requisitos y especificaciones técnicas a satisfacer por el proyecto "Diseño, implementación y control de un prototipo de Ballbot". Se trata de un proyecto de índole académica y con fines educativos, por lo que no se incluyen requisitos relativos a ninguna normativa específica.

El proyecto debe constar de dos partes principales; simulación e implementación real. En la parte de simulación, a realizar en Simulink mediante la herramienta Simscape multibody, se deberá modelar de manera realista el robot, así como su interacción con la bola. La parte de implementación real consta de la selección de componentes, diseño de piezas, montaje y elaboración del código de control.

6.2.- REQUISITOS SIMULACIÓN

1. Se debe implementar un control de estabilidad que sea capaz de corregir las pequeñas desviaciones del robot sobre la bola y absorber perturbaciones ligeras.
2. Se debe implementar un control de posición para evitar que el conjunto robot-bola se desplace demasiado mientras se estabiliza, y de esta forma evitar su caída.
3. El control de posición tendrá dos versiones, una manual, en la que el usuario controlará la posición del robot por medio de un gamepad, y una automática, en la que el propio robot medirá la posición y corregirá la misma adecuadamente.
4. Se realizará un seguimiento de trayectoria utilizando el control de posición descrito anteriormente, tanto la versión manual como la automática.

6.3.- REQUISITOS IMPLEMENTACIÓN REAL

1. Se debe implementar un control de estabilidad que sea capaz de corregir las pequeñas desviaciones del robot sobre la bola y absorber perturbaciones ligeras.
2. Se debe implementar un control de posición para evitar que el conjunto robot-bola se desplace demasiado mientras se estabiliza, y de esta forma evitar su caída. A diferencia de la etapa de simulación, en este caso únicamente se implementará

un control manual de posición debido a la dificultad de obtener una medida de posición precisa.

3. El control de posición se realizará mediante una aplicación móvil y comunicación Bluetooth con el robot.
4. Por seguridad se deberá poder cortar la alimentación de los motores desde la aplicación móvil de forma inmediata.
5. El Ballbot debe ser inalámbrico, es decir, debe incorporar su propia batería para poder funcionar sin cables.
6. Se procurará, en la medida de lo posible, utilizar componentes que tenga disponibles el departamento, con el fin de ahorrar costes.
7. En vista de que el proyecto pueda tener una continuación en un momento posterior, el código utilizado para el control del Ballbot debe ser relativamente sencillo e incluir los comentarios pertinentes para facilitar la comprensión del mismo a terceras personas.

7. Presupuesto

Para la elaboración de este presupuesto se ha utilizado el precio de venta de cada componente según sus respectivas páginas web, o bien a través de portales de venta de terceros. Para los componentes de fabricación propia se detalla el cálculo de su precio. Cabe mencionar también que se han omitido los costes de diseño, programación y mano de obra ya que al tratarse de un prototipo hay una gran cantidad de tiempo dedicado a la resolución de problemas que van surgiendo, y no sería representativo.

7.1.- PIEZAS IMPRESAS EN 3D

Para estimar el coste de los ejes de las ruedas se va a utilizar la calculadora diseñada por Prusa3D (16), la empresa de Josef Prusa, pionero en el mundo de la impresión 3D. Asumiendo un peso de 15 gramos por eje y sabiendo que el material es PLA y el tiempo de impresión son 37 minutos por pieza, se obtiene un coste de cada eje de 0,45 euros.

La estimación del coste de las piezas impresas por sinterizado de nylon es más complicada, ya que depende de las otras piezas que hubiese en el lote de impresión, las cuales se desconocen. Usando como referencia las piezas que se muestran en la estimación de precios de la web de filament2print (17) se puede aceptar un coste promedio de 0,4 euros/cm3 IVA incluido, lo cual hace que el coste de cada anclaje sea de 19,28 euros (su volumen es de 48,20 cm2) y el coste de cada anclaje de batería de 1,29 euros (su volumen es de 3.23cm2).

7.2.- PLACAS DE METACRILATO

Respecto a las placas, se tiene en cuenta el coste inicial del material y el coste del corte láser. Se pueden conseguir placas de metacrilato de 22x22cm por 5,95 euros (18), y en cuanto al corte láser, se puede estimar su coste en 1,2 euros/minuto (19). Estimando 10 minutos de mecanizado para la placa superior y 13 minutos para la inferior, ya que tiene más agujeros, el coste del corte láser quedaría en 12,00 euros para la placa superior y 15,6 euros la inferior. Esto hace un total de 17,95 euros para la placa superior y 21,55 euros para la inferior, IVA incluido.

7.3.- TABLA RESUMEN

Categoría	Nombre	Precio unitario (eur)	Cantidad	Precio total (eur)
Componentes electrónicos	Motor Pololu 12V	22,49	3	67,47
	Arduino Due	29,24	1	29,24
	Controlador de motor	21,68	3	65,04
	IMU	4,95	1	4,95
	Batería principal	21,48	1	21,48
	Batería secundaria	6,41	1	6,41
	Módulo Bluetooth HC06	3,75	1	3,75
			Subtotal	198,39
Componentes mecánicos	Pelota baloncesto Tarmak	5,78	1	5,78
	Rueda omnidireccional	19,77	3	59,31
	Acoplamiento	7,68	3	23,04
	Anclajes	15,93	3	47,79
	Placa inferior	17,81	1	17,81
	Placa superior	14,83	1	14,83
	Anclaje batería	1,07	4	4,28
	Eje rueda	0,45	3	1,35
	Brida motor	5,20	3	15,60
			Subtotal	186,79
Otros componentes	Tornillería	-	-	8,55
	Led RGB	0,64	1	0,64
	Cables	-	-	0,46
			Subtotal	9,65
			Total sin IVA	394,78
			IVA 21%	82,90
			TOTAL (euros)	477,68

8. Conclusiones y propuestas de mejora

Durante el desarrollo de este trabajo se ha conseguido realizar una simulación funcional del Ballbot, tanto la versión controlada mediante gamepad como la versión de control automático. Se ha conseguido en ambos casos que siga una trayectoria de forma bastante precisa. También se ha desarrollado el prototipo real, incluyendo el diseño, fabricación y montaje del mismo, así como la elaboración del código de control y una interfaz para controlar el Ballbot desde el móvil mediante Bluetooth. Sin embargo, y pese a todos los esfuerzos realizados al respecto, no se ha conseguido que el robot se mantenga en equilibrio de manera completamente autónoma, ya que requiere un ligero contacto para que el robot se mueva más despacio y de esta forma el controlador sea capaz de mantenerlo en equilibrio.

Una posible explicación es que los motores utilizados tengan una reductora demasiado grande, y quizás para poder lograr la estabilidad es necesario que las ruedas del Ballbot giren más deprisa al aplicar acciones de control pequeñas. En esta línea, quizás habría sido mejor idea utilizar las versiones de estos motores con reductora de 30 o 19, en vez de la versión de 70 que se ha utilizado.

Otra posible razón para las dificultades encontradas es que el sistema sea simplemente demasiado rápido como para ser controlado por una persona. Es decir, que la única forma de mantener el robot en equilibrio sea mediante un control automático de posición, ya que cuando se empieza a desviar en una dirección es importante actuar inmediatamente para evitar que coja velocidad, y un control manual está limitado a la velocidad de reacción del usuario. Si esto fuese cierto, la razón por la que el control manual funcionó en la etapa de simulación sería que esta se estaba ejecutando a velocidad inferior a la real (debido a que el ordenador utilizado no era capaz de procesar a tiempo real), con lo cual la persona que manejaba el gamepad tenía más tiempo para ver cuando el robot se estaba empezando a desplazar en una dirección, y así poder corregirlo adecuadamente. También influye que el control con un gamepad es mucho más preciso que mediante una aplicación móvil.

Por todo ello, la principal propuesta de mejora que se propone es la implementación de un control automático de posición en el prototipo. Si bien para conseguir un seguimiento de trayectoria aceptable será necesario un sistema de posicionamiento preciso mediante sensores exteroceptivos (por ejemplo, mediante balizas), es posible que se pueda conseguir la estabilidad del Ballbot de manera completamente autónoma si se utiliza la lectura de los encoders de los motores para estimar la posición del robot en cada instante, y se aplica una acción de control adecuada para devolverlo a una cierta posición inicial. Otras propuestas de mejora interesantes son:

- Inclusión de sensores en el Ballbot que le permitan evitar obstáculos durante su desplazamiento
- Diseño de un soporte de pruebas que se mueva con el Ballbot, pero sin entorpecerlo, y que sea capaz de sujetar el robot en caso de caída para evitar que contacte con el suelo.
- Búsqueda o elaboración de una bola más adecuada para este uso. Aunque el balón de baloncesto tiene buen agarre, posee los característicos surcos que dificultan la rodadura. Se podría modificar una bola de otro tipo (por ejemplo, un balón medicinal o incluso una bola de bolos) para ponerle una superficie que agarre bien. Sería interesante también ver el efecto de utilizar bolas de distinto tamaño.
- Modificar el robot para que tenga el centro de gravedad más alto. Se pueden utilizar varillas roscadas más largas y separar aún más la placa superior de la inferior. Esto debería hacer que el sistema se comporte ligeramente más lento, facilitando así el control.

A título personal, más allá de los resultados obtenidos, la realización de este trabajo ha supuesto un broche final estupendo a modo de resumen y ampliación de los conocimientos adquiridos a lo largo del máster, tanto en la parte mecánica como en la electrónica y el control. También ha demostrado ser una oportunidad fantástica para iniciarme en el mundo de Arduino y la infinidad de cosas que se pueden hacer con él, al ser este el primer proyecto de cierta envergadura en el que lo utilizo.

9. Bibliografía

1. Nagarajan, Umashankar, y otros. State Transition, Balancing, Station keeping, and Yaw Control for a Dynamically Stable Single Spherical Wheel Mobile Robot. Pittsburgh : Carnegie Mellon University, 2005.

2. Endo, Tatsuro y Nakamura, Yoshihiko. An Omnidirectional Vehicle on a Basketball. Tokio : Universidad de Tokio, 2005.

3. Festo. Bionic Mobile Assistant. [En línea] [Citado el: 08 de 05 de 2023.] https://www.festo.com/es/es/e/sobre-festo/investigacion-y-desarrollo/bionic-learning-network/lo-mas-destacado-de-2018-2021/bionicmobileassistant-id_326923/.

4. Chevalie, Juan Pablo. Preliminary study for the design of a Ballbot. Turín : Politecnico di Torino, 2018.

5. University of Illinois. Hands Free Wheelchair Prototype. [En línea] [Citado el: 08 de 05 de 2023.] https://mechse.illinois.edu/news/54099.

6. Sphero. Sphero robot. [En línea] [Citado el: 08 de 05 de 2023.] https://sphero.com/collections/coding-robots/family_bolt#shopify-section-collection-template.

7. Wheelchair of the future, the ballbot. [En línea] [Citado el: 08 de 05 de 2023.] https://www.youtube.com/watch?v=Y9mBY8DMyGQ.

8. Pololu. 50:1 Metal Gearmotor 37Dx70L mm 12V with 64 CPR Encoder (Spur Pinion). [En línea] [Citado el: 24 de 05 de 2023.] https://www.pololu.com/product/2824.

9. TEM Components. Brida de aluminio. [En línea] [Citado el: 24 de 05 de 2023.] https://www.tme.eu/es/details/pololu-1084/accesorios-para-micromotores/pololu/stamped-aluminum-l-bracket-pair-for-37d/.

10. MZX. MSDN470 Motor Controller. [En línea] [Citado el: 09 de 06 de 2023.] https://www.alibaba.com/product-detail/Perfect-Anti-dead-zone-H-Bridge_60590319410.html.

11. dpwestek. Batería. [En línea] [Citado el: 30 de 05 de 2023.] https://es.aliexpress.com/item/32761089441.html.

12. Aoweital. Acoplamiento 5mm - 8mm. [En línea] [Citado el: 09 de 06 de 2023.] https://www.amazon.com/dp/B0C176ZVV3/ref=sspa_dk_detail_1?pd_rd_i=B0C175S4WF&pd_rd_w=RT8Pi&content-id=amzn1.sym.eb7c1ac5-7c51-4df5-ba34-

ca810f1f119a&pf_rd_p=eb7c1ac5-7c51-4df5-ba34-ca810f1f119a&pf_rd_r=5404DJHBVAAXBPJME825&pd_rd_wg=5S97n&pd_rd_r=145c6e52-.

13. DK Electronics. L293D Motor Control Shield. [En línea] [Citado el: 24 de 05 de 2023.] https://www.ee-diary.com/2020/10/DC-motor-control-L293D-Motor-Shield.html.

14. Adafruit. Motor Shield V2.0. [En línea] [Citado el: 09 de 06 de 2023.] https://www.adafruit.com/product/1438.

15. Prometec. Controlador L298N. [En línea] [Citado el: 09 de 06 de 2023.] https://www.prometec.net/l298n/.

16. Prusa3D. Calculadora de coste de impresión. [En línea] [Citado el: 09 de 06 de 2023.] https://blog.prusa3d.com/es/calculadora-precio-impresion-3d_38905/.

17. Filament2print. Calcular el coste de impresión en SLS. [En línea] [Citado el: 08 de 06 de 2023.] https://filament2print.com/es/blog/121_como-calcular-el-coste-de-piezas-impresas-en-.html.

18. RotulaTuMismo. Placa de metacrilato de 5mm. [En línea] [Citado el: 09 de 06 de 2023.] https://www.rotulatumismo.com/135-placas-de-metacrilato-5mm.html?gclid=CjwKCAjwm4ukBhAuEiwA0zQxk45nDwdH6hehdhhkSYxD36x50tGPNMN9iWgmR9y49_5Y0oIMorkLghoCiNoQAvD_BwE.

19. Atta33. Tarifas de procesos de fabricación. [En línea] [Citado el: 04 de 06 de 2023.] https://atta33.com/servicios-tarifas/.

10.Anexos

10.1.- CÓDIGO ARDUINO DE LA IMPLEMENTACIÓN

```
#include <Wire.h>
#include <uNavAHRS.h>
#include <MPU9250.h>

//Pines conectados al Arduino
  #define R 25
  #define G 27
  #define B 29
  #define N 23
  #define RT 53

  #define PWM11 8
  #define PWM12 9
  #define PWM21 10
  #define PWM22 11
  #define PWM31 12
  #define PWM32 13
  #define GND1 50

//Conexion Bluetooth
  String bluetoothData ="";//String para guardar el comando recibido por bluetooth

//Control de duracion de vuelta de bucle
  long t1=0, t2=0 ;//t1 y t2 seran los milisegundos de la vuelta de bucle actual y la anterior
  int T=8;  //duracion (en milisegundos) de cada vuelta de bucle
  float Tm=T/1000.0, t=0.0 ; //Tm es la duracion en segundos de cada vuelta de bucle. t es el tiempo total trascurrido en segundos

//Variables motores
```

```
  int SAT = 12; //saturacion del motor. Valor de accion de control que se desea que corresponda con el motor yendo a velocidad maxima, en este caso 12V
  float acc1 = 0.0, acc2 = 0.0, acc3 = 0.0; //acciones de control a mandar a los motores (V)
  float vel1 = 0.0, vel2 = 0.0, vel3 = 0.0; //velocidades de giro de los motores (rad/s)
  float zonaMuerta = 0.2; //Voltios

  float uRueda1, uRueda2, uRueda3, mandoX = 0, mandoY = 0, mandoZ = 0;
  float coefCorreccionZ = 1.5, ajusteUY = 0.65;//coefCorreccionZ genera un giro en Z proporcional a la accion de control en Y. ajusteUY compensa esa acción de control para que el robot se comporte igual en X y en Y
  float k1=1.96, k2=-4.11, k3=2.3;//Coeficientes utilizados para transformar acciones de control en X e Y a acciones de control a cada rueda
  float uX, uY, uZ, uX_p, uX_d, uX_i, uY_p, uY_d, uY_i; //Acciones de control de cada eje
  float kp=0.58, kd=0.014, ki=0.00; //Parametros del controlador PID
  float integralX, integralY;//Para acumular el error en cada eje
  float integralSize = 1;//Tamaño maximo del error acumulado (antiwindup)

//Variables acelerometro
  MPU9250 IMU (Wire,0x68); //Para configurar el IMU
  uNavAHRS filtro ; //Filtro de los angulos de inclinacion
  float oX=0.0, oX_1 = 0.0, oY=0.0, oY_1 = 0.0, wX=0.0, wY=0.0; //Angulos y velocidades angulares de pitch y roll
  float oY_inicial=0.0, oX_inicial=0.0 ; //Valores iniciales de los angulos del robot

//Variables de debug
  bool calibrar = 1; //Determina si se hace calibracion del IMU. Util desactivarla si se hacen pruebas con los motores para no tener que esperar a que acabe la calibracion
  bool activarMotores = 0; //Los motores comienzan desactivados por seguridad
  bool mostrarAccRuedas = 0, mostrarAccEjes = 0, mostrarVelRuedas = 0, mostrarAngulos = 0; //Variables que controlan la informacion a mandar por puerto serie
```

```
  bool modoCalibrarXY = 0, modoCalibrarZonaMuerta = 0;//Modo calibrar XY desactiva el controlador PID dejando unicamente la señal dada por el joystick. Modo calibrar zona muerta aplica la accion de control deseada a los tres motores a la vez
  float accTest = 0.0; //Accion de control a aplicar en el modo calibrar zona muerta. Comienza en cero y se modifica mediante comandos bluetooth
  bool soloX = 0, soloY = 0; //Modo control solo en eje X y modo control solamente en eje Y
  int viewDataCounter = 0, viewDataMax = 1; //Para poder tener unas graficas en el scope que de tiempo a procesarlas visualmente mejor se puede mandar dato cada ciertas vueltas de bucle


void setup() {

  Serial.begin(115200) ; //Activar puerto serie
  Serial1.begin(115200); //activar puerto serie de comunicacion bluetooth
  pinMode(R,OUTPUT) ; pinMode(G,OUTPUT) ; pinMode(B,OUTPUT) ; pinMode(N,OUTPUT) ; pinMode(RT,OUTPUT) ; //Configuracion de pines del led RGB
  pinMode(PWM11,OUTPUT); pinMode(PWM12,OUTPUT); pinMode(PWM21,OUTPUT); pinMode(PWM22,OUTPUT); pinMode(PWM31,OUTPUT); pinMode(PWM32,OUTPUT); //Configuracion pines pwm
  pinMode(GND1,OUTPUT) ; digitalWrite(GND1,LOW); //Se necesita una conexion a masa y todos los pines GND estan ocupados, asi que se habilita el pin GND1 como masa

  //Acelerometro
  IMU.begin(); //Inicializar acelerometro
  IMU.setAccelRange(MPU9250::ACCEL_RANGE_8G) ; // {_2G, _4G, _8G, _16G}
  IMU.setGyroRange(MPU9250::GYRO_RANGE_500DPS); // {_250DPS, _500DPS, _1000DPS, _2000DPS} DPS=Degrees Per Second
  IMU.setDlpfBandwidth(MPU9250::DLPF_BANDWIDTH_5HZ); // {_184HZ, _92HZ, _41HZ, _20HZ, _10HZ, _5HZ}
  IMU.setSrd(0); // Data Output Rate = 1000 / (1 + SRD)
  filtro.setInitializationDuration(30*1000*1000);
```

```
  digitalWrite(R,HIGH) ; digitalWrite(B,HIGH) ;
  if(calibrar){Calibrado();} //Calibrado del filtro y establecimiento de valores iniciales de angulos de pitch y roll (durante esta etapa el led RGB se enciende en colores azul y rojo)
  digitalWrite(R,LOW) ; digitalWrite(B,LOW) ;
}

void loop() {
  //Procesar comando bluetooth
  if (Serial1.available()) {
    char charRecibido = Serial1.read();//Leer un caracter
    if (charRecibido == 'w') {//Uso w como fin de comando porque \n daba problemas
      bluetoothCommand(bluetoothData);//Procesado del comando Bluetooth
      bluetoothData = "";//Resetear el string del comando ya procesado
    } else {
      bluetoothData += charRecibido;//Si el comando no esta completo, añadir el caracter al comando. Despues, ir a leer el siguiente
    }
  }

  //Indicador de marcha
  if(activarMotores==1){
    digitalWrite(RT,HIGH);//Led verde señaliza motores encendidos
  }else{digitalWrite(RT,LOW);}

  //Lectura de angulos de inclinacion (solo si se ha realizado el calibrado) y actualizacion de velocidades angulares y acumulaciones de error
  if (calibrar){
    oX_1 = oX; //Actualizacion de los valores de la vuelta de bucle anterior
    oY_1 = oY;
    if (Muestra()) {//Muestra() actualiza los valores de oX y oY con los valores correspondientes a la vuelta de bucle actual. Si no se pueden actualizar sacar mensaje de error.
```

```
    wX=(oX-oX_1)/Tm; //Calculo de la velocidad angular con el valor del angulo de la vuelta de bucle anterior
    wY=(oY-oY_1)/Tm;
    integralX += oX*Tm; //Actualizacion de la suma del error con el correspondiente a la vuelta de bucle actual
    integralY += oY*Tm;
    if (integralX>integralSize){integralX=integralSize;} else if (integralX<-integralSize){integralX=-integralSize;} //Antiwindup para evitar que la accion integral crezca demasiado
    if (integralY>integralSize){integralY=integralSize;} else if (integralY<-integralSize){integralY=-integralSize;}
   } else {Serial1.println("ERROR: No es posible actualizar los valores de oX y oY") ;}
  }

  //Controlador de estabilidad en ejes X e Y
  if (modoCalibrarXY==1){//En este modo se usan solo los valores que da el controlador de pruebas a mandoX, mandoY y mandoZ
   uX_p = 0; uX_d = 0; uX_i=0;
   uY_p = 0; uY_d = 0; uY_i=0;
  } else{ //Calculo de las componentes proporcional, derivada e integral en cada eje
   uX_p = kp*oY;
   uX_d = kd*wY;
   uX_i = ki*integralY;
   uY_p = kp*oX;
   uY_d = kd*wX;
   uY_i = ki*integralX;
  }
  uX = uX_p + uX_d + uX_i + mandoX; //Calculo de la accion de control en cada eje
  uY = uY_p + uY_d + uY_i + mandoY;
  uZ = 3*mandoZ;//El 3 se ajusto experimentalmente para que gire adecuadamente. Otra opcion es cambiar los valores de mandoZ mandados desde la aplicacion movil
  if (soloX){uY=0;}//En el modo de control unicamente en eje X no se aplica acción de control en Y
```

```
  if (soloY){uX=0;}//En el modo de control unicamente en eje Y no se aplica acción de
control en X

//Calculo acciones de control para cada rueda
  if (modoCalibrarZonaMuerta){ //El modo calibrar zona muerta aplica la misma accion de
control a los tres motores
    acc1 = accTest;
    acc2 = accTest;
    acc3 = accTest;
  } else{ //La justificacion de estas ecuaciones se encuentra detallada en la memoria
    acc1 = (k1+coefCorreccionZ)*uY*ajusteUY + uZ;
    acc2 = (k2+coefCorreccionZ)*uY*ajusteUY - k3*uX + uZ;
    acc3 = (k2+coefCorreccionZ)*uY*ajusteUY + k3*uX + uZ;
  }

//Mandar acciones de control a los motores
  acc2motor();

//Mandar datos para visualizar
  if (viewDataCounter>viewDataMax){
    viewData(); //Si hay alguna visualizacion de datos activada, mandar los datos
correspondientes por el puerto serie
    viewDataCounter = 0; //Resetear el contador de ciclos
  } else{viewDataCounter++;}


//Control de duracion de cada vuelta de bucle. Si dura mas de lo debido se enciende el led
RGB en color rojo
  t2=millis() ;
  if (t2-t1>T) {digitalWrite(R,HIGH) ;}
  else {digitalWrite(R,LOW) ;}
  while (t2-t1<T) {t2=millis() ;}
  t1=millis() ;
```

```
 t=t+Tm ; //tiempo total de ejecucion
}

void bluetoothCommand(String bluetoothData){
 if (bluetoothData.startsWith("X") && bluetoothData.indexOf("Y") != -1) { //Parseado del joystick izquierdo, que controla el movimiento del robot en X e Y (informacion en formato X12Y34)
   int yIndex = bluetoothData.indexOf('Y');//Ver la posicion en la que esta la Y
   String xValueStr = bluetoothData.substring(1, yIndex);//Extraer el valor de la posicion X como string
   String yValueStr = bluetoothData.substring(yIndex + 1, bluetoothData.length());//Extraer el valor de la posicion Y como string (funcionara sin importar el numero de digitos del valor de la posicion X)
   int xValue = xValueStr.toInt();//Convertir los string a int para poder operar con ellos
   int yValue = yValueStr.toInt();
   mandoX = xValue/-80.0;//Reescalado para adecuar los valores a las acciones de control del sistema
   mandoY = yValue/80.0;
 }
 else if (bluetoothData.startsWith("r") && bluetoothData.indexOf("Y") != -1) { //Parseado del joystick derecho, que controla el giro del robot en Z. para diferenciarlo sus mensajes llegan precedidos de la letra r (formato rX12Y34)
   int yIndex = bluetoothData.indexOf('Y');//Ver la posicion en la que esta la Y
   String zValueStr = bluetoothData.substring(2, yIndex);//Extraer el valor de la posicion X del joystick (el valor de la posicion Y es indiferente)
   int zValue = zValueStr.toInt();//Convertir string a int
   mandoZ = zValue/80.0;//Reescalado
 }
 else if (bluetoothData == "kpu") {kp += 0.02; Serial1.print("kp:");Serial1.println(kp);}
 else if (bluetoothData == "kpd") {kp -= 0.02; Serial1.print("kp:");Serial1.println(kp);}
 else if (bluetoothData == "kdu") {kd += 0.002; Serial1.print("kd:");Serial1.println(kd,3);}
 else if (bluetoothData == "kdd") {kd -= 0.002; Serial1.print("kd:");Serial1.println(kd,3);}
 else if (bluetoothData == "kiu") {ki += 0.2; Serial1.print("ki:");Serial1.println(ki);}
```

```
  else if (bluetoothData == "kid") {ki -= 0.2; Serial1.print("ki:");Serial1.println(ki);}
  else if (bluetoothData == "vdu") {viewDataMax += 3; Serial1.print("Numero de vueltas
de bucle para mandar datos:");Serial1.println(viewDataMax);}
  else if (bluetoothData == "vdd") {viewDataMax -= 3;
if(viewDataMax<1){viewDataMax=1;} Serial1.print("Numero de vueltas de bucle para
mandar datos::");Serial1.println(viewDataMax);}
  else if (bluetoothData == "zmu") {zonaMuerta += 0.1; Serial1.print("Zona
muerta:");Serial1.println(zonaMuerta);}
  else if (bluetoothData == "zmd") {zonaMuerta -= 0.1; Serial1.print("Zona
muerta:");Serial1.println(zonaMuerta);}
  else if (bluetoothData == "acu") {accTest += 0.1; Serial1.print("Accion de control de
test:");Serial1.println(accTest);}
  else if (bluetoothData == "acd") {accTest -= 0.1; Serial1.print("Accion de control de
test:");Serial1.println(accTest);}
  else if (bluetoothData == "acuu") {accTest += 0.5; Serial1.print("Accion de control de
test:");Serial1.println(accTest);}
  else if (bluetoothData == "acdd") {accTest -= 0.5; Serial1.print("Accion de control de
test:");Serial1.println(accTest);}
  else if (bluetoothData == "cczu") {coefCorreccionZ += 0.3; Serial1.print("Coeficiente de
correccion Z:");Serial1.println(coefCorreccionZ);}
  else if (bluetoothData == "cczd") {coefCorreccionZ -= 0.3; Serial1.print("Coeficiente de
correccion Z:");Serial1.println(coefCorreccionZ);}
  else if (bluetoothData == "auyu") {ajusteUY += 0.1; Serial1.print("Ajuste
UY:");Serial1.println(ajusteUY);}
  else if (bluetoothData == "auyd") {ajusteUY -= 0.1; Serial1.print("Ajuste
UY:");Serial1.println(ajusteUY);}
  else if (bluetoothData == "m") {activarMotores=1; Serial1.println("Motores activados");}
  else if (bluetoothData == "p") {activarMotores=0; Serial1.println("Motores
desactivados");}
  else if (bluetoothData == "ar") {mostrarAccRuedas=1; Serial1.println("Mostrando accion
de control en las ruedas [V]: acc1 acc2 acc3");}
```

```
  else if (bluetoothData == "ns") {mostrarAccRuedas=0, mostrarAccEjes=0,
mostrarVelRuedas=0, mostrarAngulos=0; Serial1.println("No mostrando informacion por
el puerto serie");}
  else if (bluetoothData == "calxy") {modoCalibrarXY=1; Serial1.println("Modo
calibrarXY activado");}
  else if (bluetoothData == "calxyoff") {modoCalibrarXY=0; Serial1.println("Modo
calibrar XY desactivado");}
  else if (bluetoothData == "calzm") {modoCalibrarZonaMuerta=1; Serial1.println("Modo
calibrar zona muerta activado");}
  else if (bluetoothData == "calzmoff") {modoCalibrarZonaMuerta=0;
Serial1.println("Modo calibrar zona muerta desactivado");}
  else if (bluetoothData == "solox") {soloX=1; Serial1.println("Modo control unicamente
en eje X");}
  else if (bluetoothData == "dosejes") {soloX=0; soloY=0; Serial1.println("Modo control
en ambos ejes");}
  else if (bluetoothData == "soloy") {soloY=1; Serial1.println("Modo control unicamente
en eje Y");}
  else if (bluetoothData == "ae") {mostrarAccEjes=1; Serial1.println("Mostrando accion de
control en los ejes: uX_p uX_d uY_p uY_d uX uY");}
  else if (bluetoothData == "a") {mostrarAngulos=1; Serial1.println("Mostrando angulos
[rad] y vel. angulares [rad/s] del robot: oX oY wX wY");}
  else if (bluetoothData == "vi") {valoresIniciales();}//Obtener los valores iniciales de los
angulos en la posicion de equilibrio, sobre la bola
  else {Serial1.println("Comando Bluetooth desconocido");}

}

void acc2motor(){
  //Compensacion de zona muerta del motor
  if(acc1>0 && acc1<zonaMuerta){acc1+=zonaMuerta;}
  else if(acc1<0 && acc1>-zonaMuerta){acc1-=zonaMuerta;}
  if(acc2>0 && acc2<zonaMuerta){acc2+=zonaMuerta;}
  else if(acc2<0 && acc2>-zonaMuerta){acc2-=zonaMuerta;}
```

```
  if(acc3>0 && acc3<zonaMuerta){acc3+=zonaMuerta;}
  else if(acc3<0 && acc3>-zonaMuerta){acc3-=zonaMuerta;}

  //Saturacion de las acciones de control
  if (acc1>SAT){acc1=SAT;}
  else if(acc1<-SAT){acc1=-SAT;}
  if (acc2>SAT){acc2=SAT;}
  else if(acc2<-SAT){acc2=-SAT;}
  if (acc3>SAT){acc3=SAT;}
  else if(acc3<-SAT){acc3=-SAT;}

//Mandar la accion de control establecida a cada motor
  if (acc1>=0 && activarMotores)
{analogWrite(PWM11,map(abs(acc1*100),0,SAT*100,0,255)); analogWrite(PWM12,0);}
  else if (acc1<0 && activarMotores)
{analogWrite(PWM12,map(abs(acc1*100),0,SAT*100,0,255)); analogWrite(PWM11,0);}
  else {analogWrite(PWM11,0); analogWrite(PWM12,0); }
  if (acc2>=0 && activarMotores)
{analogWrite(PWM22,map(abs(acc2*100),0,SAT*100,0,255)); analogWrite(PWM21,0);}
  else if (acc2<0 && activarMotores)
{analogWrite(PWM21,map(abs(acc2*100),0,SAT*100,0,255)); analogWrite(PWM22,0);}
  else {analogWrite(PWM21,0); analogWrite(PWM22,0);}
  if (acc3>=0 && activarMotores)
{analogWrite(PWM32,map(abs(acc3*100),0,SAT*100,0,255)); analogWrite(PWM31,0);}
  else if (acc3<0 && activarMotores)
{analogWrite(PWM31,map(abs(acc3*100),0,SAT*100,0,255)); analogWrite(PWM32,0);}
  else {analogWrite(PWM31,0); analogWrite(PWM32,0);}
}

void viewData(){

  if (mostrarAccRuedas){
```

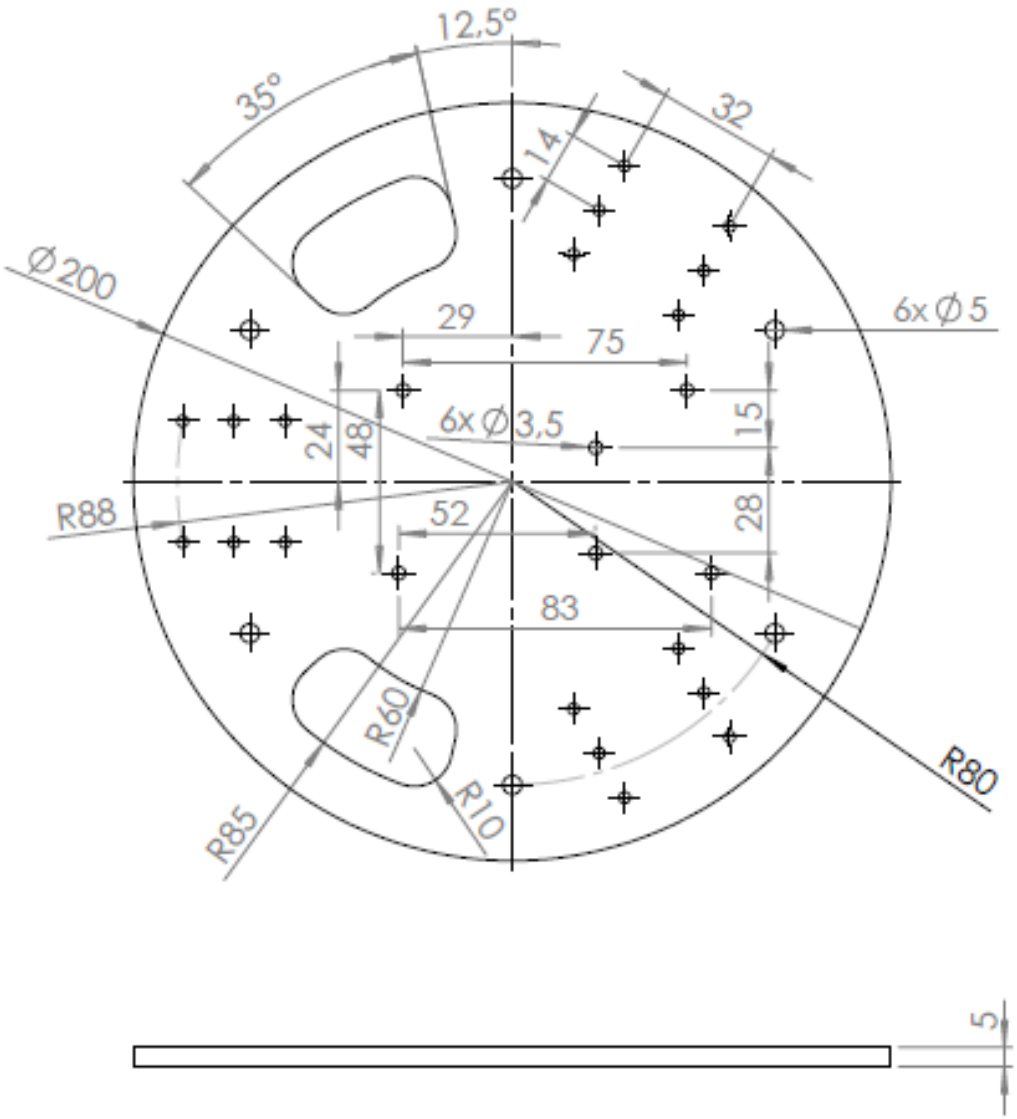
```
    Serial.print(acc1); Serial.print("\t") ; Serial.print(acc2); Serial.print("\t") ;
Serial.print(acc3); Serial.print("\n") ;
  }
  if (mostrarAccEjes){
    Serial.print(uX_p); Serial.print("\t") ;
    Serial.print(uX_d); Serial.print("\t") ;
    Serial.print(uX_i); Serial.print("\t");
    Serial.print(uY_p); Serial.print("\t") ;
    Serial.print(uY_d); Serial.print("\t") ;
    Serial.print(uY_i); Serial.print("\t");
    Serial.print(uX); Serial.print("\t") ;
    Serial.print(uY); Serial.print("\n") ;
  }
  if (mostrarAngulos){
    Serial.print(oX) ; Serial.print("\t") ;
    Serial.print(oY) ; Serial.print("\t") ;
    Serial.print(wX) ; Serial.print("\t") ;
    Serial.print(wY) ; Serial.print("\n") ;
  }
}

void Calibrado() {
  //digitalWrite(RT,HIGH) ;
//  Serial.println("Calibrado del acelerómetro.") ;
//  Serial.println("Eje Z") ; delay(5000) ;
//  IMU.calibrateAccel() ;
//  Serial.println("Eje Y") ; delay(5000) ;
//  IMU.calibrateAccel() ;
//  Serial.println("Eje X") ; delay(5000) ;
//  IMU.calibrateAccel() ;
//  Serial.println("Eje -Z") ; delay(5000) ;
//  IMU.calibrateAccel() ;
//  Serial.println("Eje -Y") ; delay(5000) ;
```

```
//  IMU.calibrateAccel() ;
//  Serial.println("Eje -X") ; delay(5000) ;
//  IMU.calibrateAccel() ;
//  Serial.println("Calibrado del magnetómetro.");
//  delay(5000) ;
//  IMU.calibrateMag() ;
//  Serial.println("----------------------------------------") ;
//  Serial.print("float AccelBiasX =  ") ;
//  Serial.print(IMU.getAccelBiasX_mss(),4) ; Serial.println(";") ;
//  Serial.print("float AccelScaleFactorX =  ") ;
//  Serial.print(IMU.getAccelScaleFactorX(),4) ; Serial.println(";") ;
//  Serial.print("float AccelBiasY =  ") ;
//  Serial.print(IMU.getAccelBiasY_mss(),4) ; Serial.println(";") ;
//  Serial.print("float AccelScaleFactorY =  ") ;
//  Serial.print(IMU.getAccelScaleFactorY(),4) ; Serial.println(";") ;
//  Serial.print("float AccelBiasZ =  ") ;
//  Serial.print(IMU.getAccelBiasZ_mss(),4) ; Serial.println(";") ;
//  Serial.print("float AccelScaleFactorZ =  ") ;
//  Serial.print(IMU.getAccelScaleFactorZ(),4) ; Serial.println(";") ;
//  Serial.print("float MagBiasX =  ") ;
//  Serial.print(IMU.getMagBiasX_uT(),4) ; Serial.println(";") ;
//  Serial.print("float MagScaleFactorX =  ") ;
//  Serial.print(IMU.getMagScaleFactorX(),4) ; Serial.println(";") ;
//  Serial.print("float MagBiasY =  ") ;
//  Serial.print(IMU.getMagBiasY_uT(),4) ; Serial.println(";") ;
//  Serial.print("float MagScaleFactorY =  ") ;
//  Serial.print(IMU.getMagScaleFactorY(),4) ; Serial.println(";") ;
//  Serial.print("float MagBiasZ =  ") ;
//  Serial.print(IMU.getMagBiasZ_uT(),4) ; Serial.println(";") ;
//  Serial.print("float MagScaleFactorZ =  ") ;
//  Serial.print(IMU.getMagScaleFactorZ(),4) ; Serial.println(";") ;
//  Serial.println("----------------------------------------") ;
```

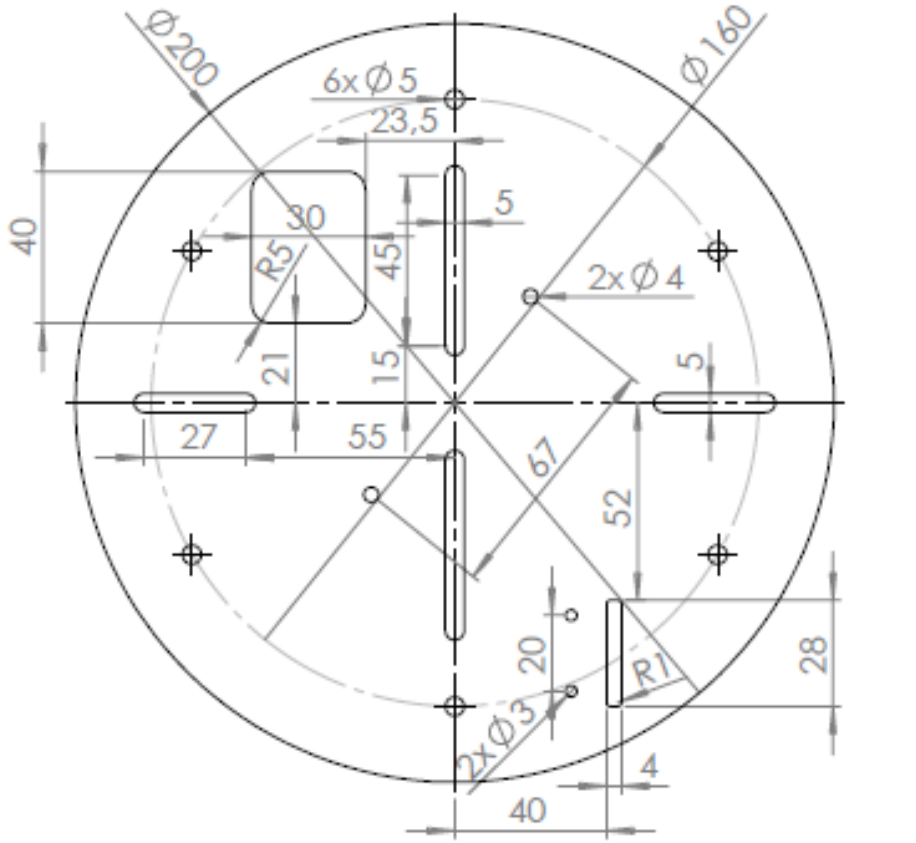
```
//Valores obtenidos para cada parametro. Se precargan para reducir el tiempo de ejecucion del calibrado
  float AccelBiasX =  -0.0066;
  float AccelScaleFactorX =  0.9976;
  float AccelBiasY =  -0.0066;
  float AccelScaleFactorY =  0.9983;
  float AccelBiasZ =  0.0511;
  float AccelScaleFactorZ =  0.9839;
  float MagBiasX =  -5.2452;
  float MagScaleFactorX =  1.2619;
  float MagBiasY =  2.3681;
  float MagScaleFactorY =  2.5835;
  float MagBiasZ =  12.6643;
  float MagScaleFactorZ =  0.5493;

  IMU.setAccelCalX(AccelBiasX,AccelScaleFactorX) ;
  IMU.setAccelCalY(AccelBiasY,AccelScaleFactorY) ;
  IMU.setAccelCalZ(AccelBiasZ,AccelScaleFactorZ) ;
  IMU.setMagCalX(MagBiasX,MagScaleFactorX) ;
  IMU.setMagCalY(MagBiasY,MagScaleFactorY) ;
  IMU.setMagCalZ(MagBiasZ,MagScaleFactorZ) ;

  Serial.print("Inicializando filtro") ;
  int cc=0;
  while (Muestra()==0) { //Se inicializa el filtro. Mientras esta inicializandose Muestra() devuelve un 0, cuando esta inicializado da un 1
    delay(10) ;
    if (cc>=100) {Serial.print(".") ; cc=0 ;} else {cc++ ;}
  }
  Serial.println("\nFiltro inicializado con exito");
}

void valoresIniciales(){
```

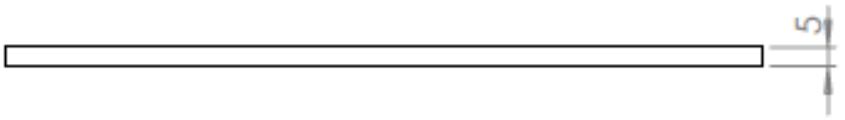
```
  int cc=0;
  Serial.print("\nCalculo de valores iniciales") ; //Calcula la media de 10 medidas de los angulos. Primero suma las 10 medidas y luego divide entre 10
  Serial1.print("\nCalculo de valores iniciales"); //Manda mensaje por bluetooth para poder tener confirmacion cuando se trabaja sin conectar al ordenador
  float acmP=0.0, acmR=0.0 ;
  for (int ii=1;ii<=10;ii++) {
    if (Muestra()) {acmP+=oY+oY_inicial ; acmR+=oX+oX_inicial ;}
    delay(10) ;
    if (cc>=100) {Serial.print(".") ; cc=0 ;} else {cc++ ;}
  }
  oY_inicial=acmP/10.0 ; oX_inicial=acmR/10.0 ;
  Serial.print("\nCabeceo inicial: ") ; Serial.println(oY_inicial) ;
  Serial.print("Alabeo inicial: ") ; Serial.println(oX_inicial) ;
  Serial1.println("Obtencion de valores iniciales finalizada"); //Manda mensaje por bluetooth para poder tener confirmacion cuando se trabaja sin conectar al ordenador
}
bool Muestra() {
  float gX, gY, gZ, aX, aY, aZ, mX, mY, mZ ;
  bool ff ;
  IMU.readSensor() ; //Lectura del sensor
  gX=IMU.getGyroX_rads() ; gY=IMU.getGyroY_rads() ; gZ=IMU.getGyroZ_rads() ;
  aX=IMU.getAccelX_mss() ; aY=IMU.getAccelY_mss() ; aZ=IMU.getAccelZ_mss() ;
  mX=IMU.getMagX_uT() ; mY=IMU.getMagY_uT() ; mZ=IMU.getMagZ_uT() ;
  ff=filtro.update(gX,gY,gZ,aX,aY,aZ,mX,mY,mZ) ;
  if (ff) {
    oY=(filtro.getPitch_rad()*(180.0f/PI))-oY_inicial ;
    oX=(filtro.getRoll_rad()*(180.0f/PI))-oX_inicial ;
  }
  return ff ;
}
```

10.2.- PLANOS



DIBUJADO	ÓSCAR GARCÍA LUENGO	**Nº PLANO**		UNIVERSIDAD POLITECNICA DE VALENCIA	
FECHA DIBUJ.	03/05/2023	Bbot - 1.1			
REVISADO	ÓSCAR GARCÍA LUENGO				
FECHA REVIS.	10/06/2023	**TIPO**	DESPIECE		
ESCALA: 1:5	Placa Inferior			**MATERIAL**	Metacrilato
				TOL. GEN.	ISO 2768-mK
				CANTIDAD	1
				HOJA	1 de 1

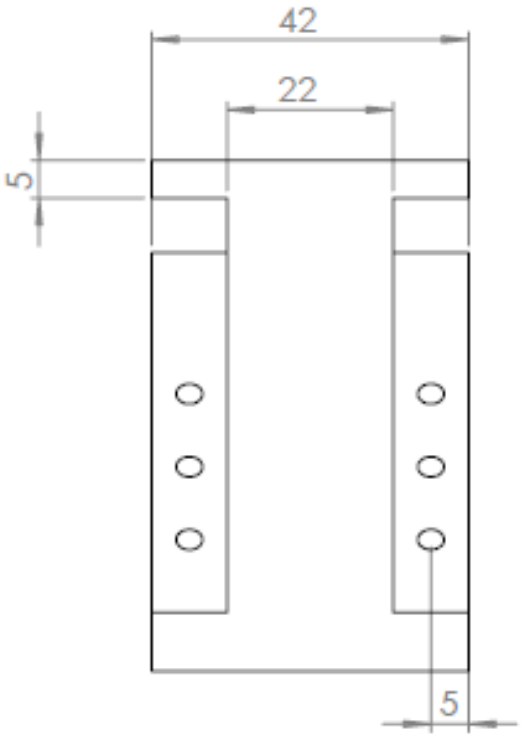


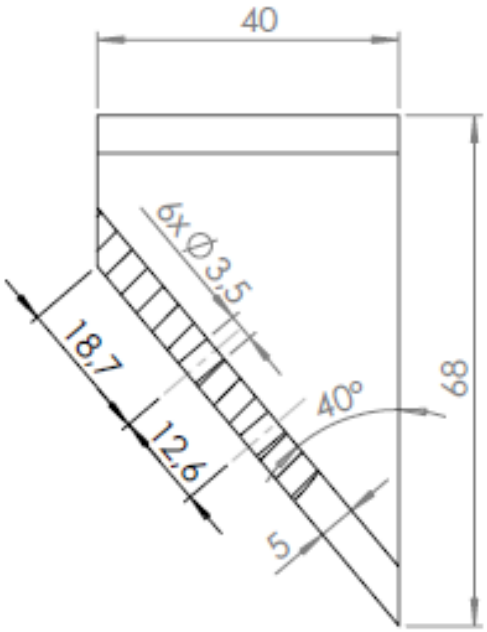


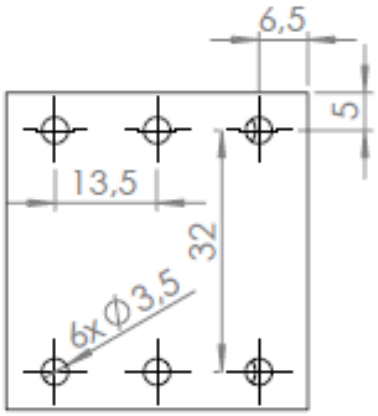
DIBUJADO	ÓSCAR GARCÍA LUENGO	**Nº PLANO**	
FECHA DIBUJ.	15/05/2023	Bbot - 1.2	
REVISADO	ÓSCAR GARCÍA LUENGO		
FECHA REVIS.	10/06/2023	**TIPO**	DESPIECE

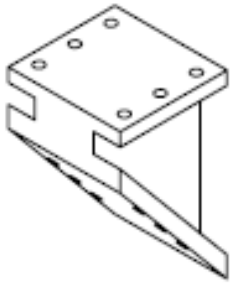
UNIVERSIDAD POLITECNICA DE VALENCIA

ESCALA:	Placa Superior	**MATERIAL**	Metacrilato
1:2		**TOL. GEN.**	ISO 2768-mK
		CANTIDAD	1
		HOJA	1 de 1





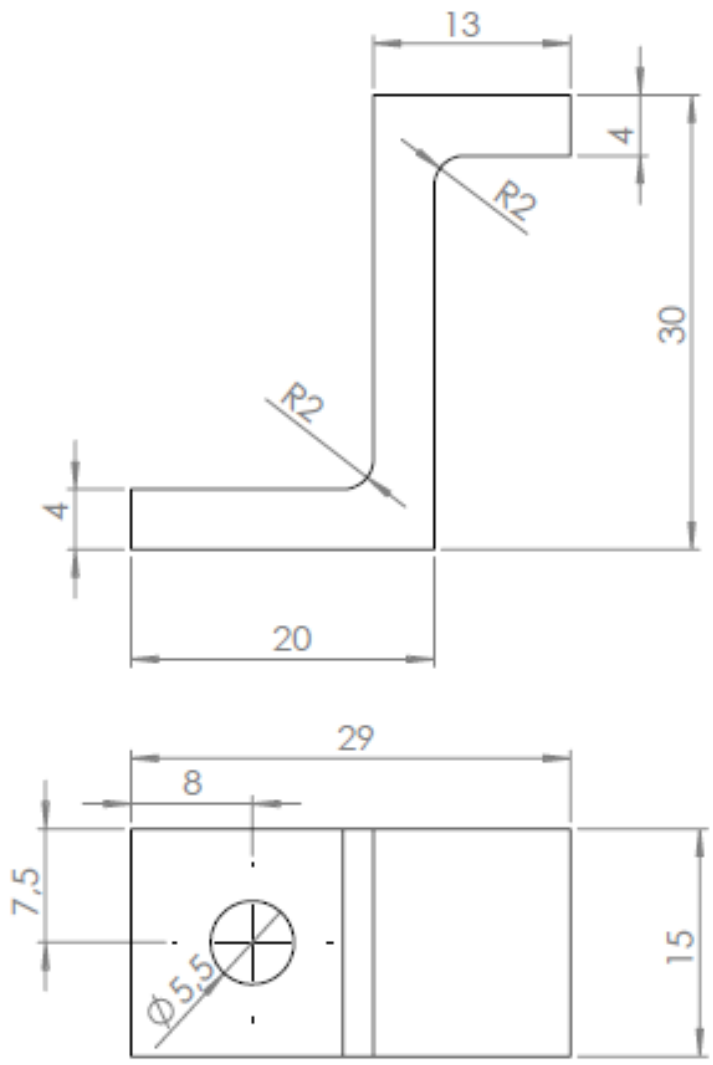


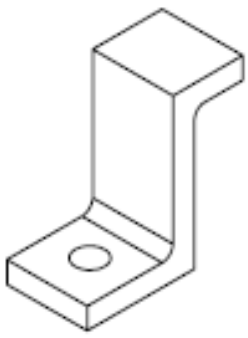


DIBUJADO	ÓSCAR GARCÍA LUENGO	**Nº PLANO**	
FECHA DIBUJ.	03/05/2023	Bbot - 1.4	
REVISADO	ÓSCAR GARCÍA LUENGO		
FECHA REVIS.	09/05/2023	**TIPO**	DESPIECE

UNIVERSIDAD POLITECNICA DE VALENCIA

ESCALA:	Anclaje	**MATERIAL**	Nylon
1:1		**TOL. GEN.**	ISO 2768-mK
		CANTIDAD	3
		HOJA	1 de 1

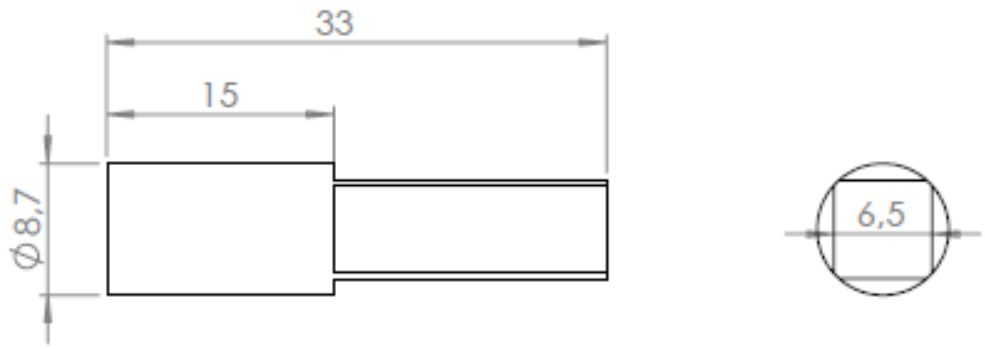


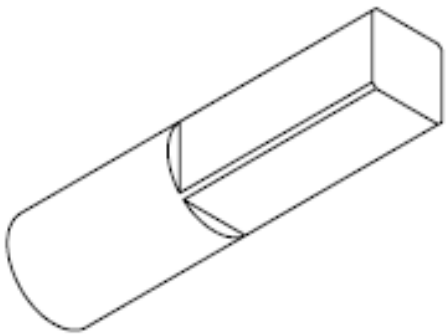


DIBUJADO	ÓSCAR GARCÍA LUENGO	**Nº PLANO**	
FECHA DIBUJ.	03/05/2023	Bbot - 1.3	
REVISADO	ÓSCAR GARCÍA LUENGO		
FECHA REVIS.	09/05/2023	**TIPO**	DESPIECE

UNIVERSIDAD POLITECNICA DE VALENCIA

ESCALA:	Anclaje batería	**MATERIAL**	Nylon
2:1		**TOL. GEN.**	ISO 2768-mK
		CANTIDAD	4
		HOJA	1 de 1





DIBUJADO	ÓSCAR GARCÍA LUENGO	**Nº PLANO**	
FECHA DIBUJ.	09/05/2023	Bbot - 1.5	
REVISADO	ÓSCAR GARCÍA LUENGO		
FECHA REVIS.	10/05/2023	**TIPO**	DESPIECE

UNIVERSIDAD POLITECNICA DE VALENCIA

ESCALA: 2:1	Eje rueda	**MATERIAL**	PLA
		TOL. GEN.	ISO 2768-mK
		CANTIDAD	3
		HOJA	1 de 1