Universidad Nacional Autónoma de México

Facultad de Ingeniería

Diseño, modelado y control de un robot móvil con forma esférica

Tesis

que para obtener el título de

Ingeniero Mecatrónico

presentan

Jorge Luis Aguirre Serralde

y

Jakob Culebro Reyes

Director de tesis

M.I. Yukihiro Minami Koyama

Ciudad Universitaria junio, 2015

Investigación realizada gracias al programa UNAM-DGAPA-PAPIME PE104212 “Mejoramiento de la calidad educativa en Ciencias Básicas a través de la Robótica”

Índice general

Capítulo 1

Introducción

1.1 Antecedentes

Debido a los grandes avances tecnológicos y a la ampliación del campo de aplicación de la robótica móvil, actualmente se busca que los robots se desempeñen en situaciones y entornos reales, y no solamente en escenarios controlados. Esto ha ocasionado que los mecanismos de locomoción convencionales ya no sean adecuados.

Dejando fuera ruedas y orugas, que son los mecanismos de locomoción más comunes, dos ideas que han despertado gran interés son los robots con extremidades y los robots modulares, que generalmente tratan de imitar la anatomía y los movimientos de seres vivos. Aunque ambos tienen la ventaja de poder adaptarse de mejor manera a las dificultades de los terrenos, tienen la desventaja de tener diseños más complicados y ser difíciles de controlar. Tomando en cuenta esto último, los robots esféricos tienen el atractivo de ser muy simples en cuanto a su forma y funcionamiento.

Aunque algunos mecanismos para esferas móviles ya se habían inventado desde hace más de un siglo [21], la idea de un robot con forma esférica se comenzó a desarrollar hace algunas décadas, y presenta algunas ventajas importantes sobre diseños tradicionales: al tener una forma simple y no requerir mucho espacio, le permite a algunos robots con este diseño moverse por lugares reducidos; la esfera tiene un solo punto de contacto con el terreno, lo que puede ayudar a minimizar la fricción y reduce la interacción indeseable con obstáculos e irregularidades del te-

rreno; el robot no puede volcarse, por lo que no pierde movilidad en prácticamente ninguna posición y, si es diseñada así, puede tener movimiento omnidireccional; finalmente, la esfera, al rodear a toda la estructura interna del robot, la protege del polvo y de los golpes, lo que hace al robot muy adecuado para tareas de exploración y reconocimiento. Si la esfera se sella, el robot podría funcionar en medios inflamables, lluviosos o sumergidos.

1.2 Planteamiento del problema

Los diferentes sistemas de locomoción para robots esféricos se han puesto a prueba en muchas ocasiones en diferentes partes del mundo, y se ha desarrollado una gran cantidad de prototipos diferentes, pero de forma aislada, con poco seguimiento, y en la mayoría de los casos, únicamente con el objetivo de probar los principios de locomoción y sin crear versiones funcionales y con aplicaciones reales, salvo algunas excepciones. Asimismo, cada mecanismo de locomoción se ha probado de forma separada, habiendo muy pocos diseños que incorporan más de uno de ellos, ocasionando que los robots creados no sean tan versátiles y completos como podrían llegar a serlo.

1.3 Objetivos

1.3.1 Objetivo general

Diseñar y construir un robot móvil con forma esférica e implementar el control de movimiento.

1.3.2 Objetivos particulares

- Proponer una configuración mecánica para el prototipo de robot esférico con base en el análisis de los sistemas de locomoción existentes.
- Obtener el modelo matemático de la configuración propuesta para generar simulaciones que permitan comprender el comportamiento del sistema y obtener algunos parámetros auxiliares para el diseño del prototipo.

- Realizar el diseño mecánico y electrónico a detalle, la manufactura y el ensamble del prototipo.
- Diseñar sistemas de control de movimiento del robot y realizar pruebas sobre el prototipo.

1.4 Hipótesis

El análisis y la evaluación de los diferentes sistemas de locomoción existentes para robots esféricos permitirá proponer una configuración mecánica que, aplicada a un prototipo, resultará en un robot esférico funcional sobre el cual se podrán hacer pruebas de control de movimiento.

1.5 Resumen por capítulos

1.5.1 Capítulo 1, Introducción

Se detallan las generalidades del proyecto, el planteamiento del problema, los objetivos generales y particulares, y se da un breve resumen de los capítulos.

1.5.2 Capítulo 2, Análisis del problema

Se detalla el estado de la técnica de los robots esféricos y sus diferentes sistemas de locomoción. Además, se describe el uso de volantes de inercia como CMG (*Control Moment Gyroscope*) como estrategia para generar grandes cantidades de par, para sobrepasar obstáculos. Se evalúan diferentes propuestas de configuraciones mecánicas y se decide por una.

1.5.3 Capítulo 3, Modelado matemático

Se explica una forma procedural de modelar de forma lagrangiana un sistema mecánico clásico compuestos de cuerpos rígidos sin bucles cinemáticos. Este procedimiento es aplicable para sistemas con movimiento en los tres ejes espaciales y sujetos a rotaciones alrededor de ejes arbitrarios. Las ecuaciones diferenciales generadas se resuelven en un *solver* numérico.

1.5.4 Capítulo 4, Diseño y construcción del prototipo

Se detalla el diseño mecánico del robot, explicando varias de las decisiones de diseño y de disposición de componentes. Se detalla también el arreglo de elementos electrónicos y sensores, y la manera en que estos elementos se integran en una arquitectura de procesamiento.

1.5.5 Capítulo 5, Pruebas de control

Se describe el proceso de linealización del sistema y el uso del sistema simplificado de esta forma para obtener un controlador de segundo orden. Se muestran los resultados de las pruebas al sistema, mediante la asignación de salidas de PWM a los motores (modo manual), y usando dos controladores, uno derivado del modelo, y un controlador PID ajustado manualmente (modo automático).

1.5.6 Capítulo 6, Conclusiones y trabajo a futuro

Se examinan los objetivos presentados al inicio del presente trabajo, y se evalúa si se cumplieron satisfactoriamente. Se concluye sobre el valor obtenido del trabajo. Se detallan los aspectos no esenciales que no pudieron ser cubiertos en el desarrollo del proyecto, para referencia futura.

1.5.7 Apéndices

Se presentan las especificaciones de los elementos mecánicos y electrónicos utilizados, se recolectan todos los planos de fabricación del proyecto, así como el código de Mathematica del procedimiento de modelado y linealización. Se recoge también la parte crítica del código de los programas del robot.

Capítulo 2

Análisis del problema

El análisis de los sistemas de locomoción y las configuraciones mecánicas para robots esféricos se llevó a cabo en tres etapas. La primera fue una descripción de los sistemas de locomoción existentes y una revisión de su estado de la técnica. La segunda etapa consistió en la realización de propuestas de configuraciones mecánicas aplicando los sistemas de locomoción descritos en la etapa anterior, pudiendo incluir cada propuesta uno o más de estos sistemas, para posteriormente llevar a cabo una evaluación de ellas, con el objetivo de seleccionar la más conveniente, de acuerdo con criterios previamente establecidos, para ser utilizada en la siguiente etapa. Finalmente, la tercera etapa consiste en la realización de propuestas de diseño conceptual basadas en la configuración seleccionada para ser evaluadas y seleccionar un diseño que será desarrollado en el prototipo.

2.1 Sistemas de locomoción y estado de la técnica

Después de una revisión del estado de la técnica ([1], [2], [4]), se identificaron cuatro sistemas mecánicos de interés, que se han utilizado para dar movimiento a robots esféricos:

- Unidad móvil interna.
- Péndulo interno.
- Arreglo de masas móviles.
- Giroscopios de control de momento

Los tres primeros sistemas están basados en un principio de desplazamiento del centro de masa, mientras que último está basado en el principio de conservación del momento angular.

Un sistema que no se menciona anteriormente es el de "cuerpo deformable". Este consiste en alterar mediante algún sistema la forma esférica del robot, de forma que alguna secuencia de deformación específica pueda hacer que el robot ruede en cierta dirección. Un ejemplo de robot de este tipo es el propuesto por K. Wait *et ál.* [20], que es muy similar a un balón de fútbol, en el que las secciones pentagonales y hexagonales de la esfera pueden ser infladas y desinfladas. Dependiendo qué secciones son infladas, el robot puede ser llevado por cierta trayectoria.

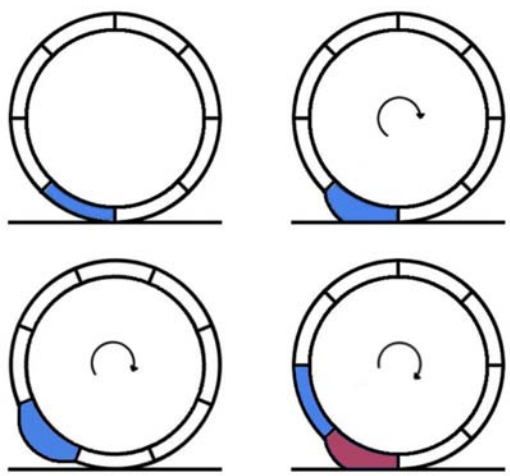


Figura 2.1 Diseño tipo "cuerpo deformable" de K. Wait *et ál.* [20].

Otro tipo de robots esféricos que ha sido desarrollado es el robot impulsado por viento, que aprovecha la energía del viento a través de un diseño especial de la superficie externa de la esfera (figura 2.2). Estos robots han sido creados para la NASA con el fin de ser utilizados para tareas de exploración en Marte, donde los fuertes vientos son un recurso natural altamente aprovechable [3].

Debido a la ausencia de un mecanismo interno de locomoción, estos dos últimos tipos de robots no han sido considerados en el análisis de sistemas de locomoción de este trabajo. Los cuatro sistemas mencionados en un principio serán descritos a detalle a continuación.

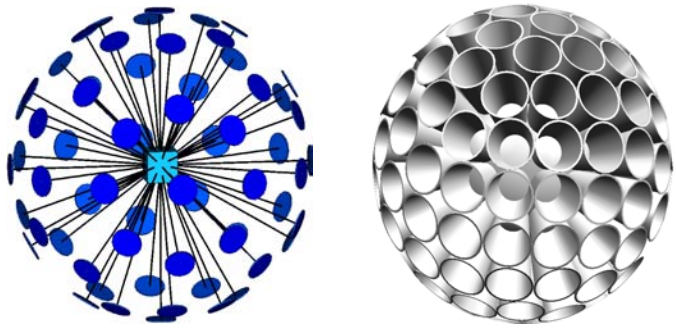


Figura 2.2 Conceptos de robot esférico para exploración en Marte [3].

2.1.1 Unidad móvil interna

La unidad móvil interna (IDU, por sus siglas en inglés) consiste en un dispositivo móvil que transmite movimiento por fricción directamente sobre la superficie interna de la esfera. Al hacerlo, el dispositivo avanza sobre la superficie, creando un cambio en el balance de masa dentro de la esfera, ocasionando que ésta ruede hacia un nuevo punto de equilibrio.

Los primeros robots con esta forma de funcionamiento fueron llamados "rueda de hámster", debido a que la IDU era un vehículo o carro que se mueve libremente sobre la superficie interna de la esfera, similar a un hámster corriendo dentro de su rueda [8]. La desventaja de este diseño es que el carro, al estar libre dentro de la esfera, puede presentar deslizamiento en su contacto con ésta e incluso puede volcarse a altas velocidades.

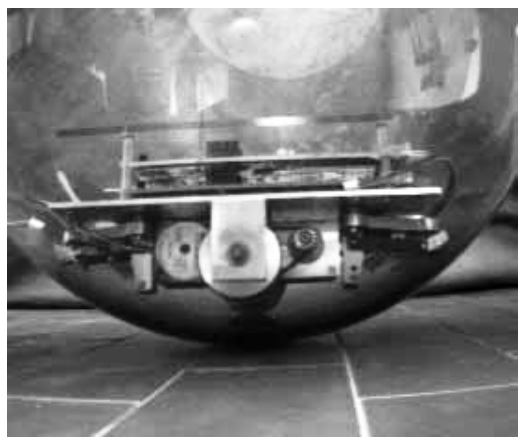


Figura 2.3 Diseño tipo "rueda de hámster" de Bicchi *et ál.* [8].

Para asegurar el contacto entre el vehículo y la esfera en todo momento, se hicieron algunas mejoras al diseño de la IDU, agregándole un eje vertical que en su extremo superior tiene una rueda libre, que es presionada contra la superficie de la esfera mediante un resorte. En la figura 2.4 se muestra una simplificación de este diseño desarrollada por A. Halme *et ál.* [10].

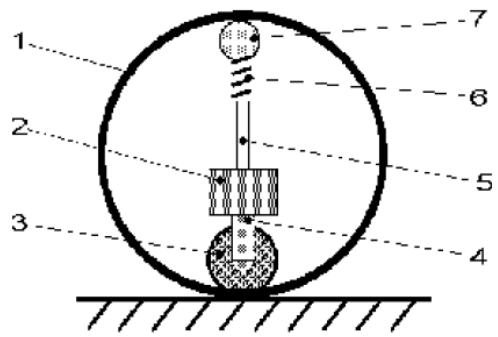


Figura 2.4 Diseño propuesto por A. Halme *et ál.* [10]
1. cuerpo del robot, 2. caja de control, 3. rueda de tracción,
4. eje de dirección, 5. eje de soporte, 6. resorte, 7. rueda de balance.

Otras variantes de este diseño pueden incluir ruedas adicionales que permiten orientar a la IDU, mientras que la rueda de tracción es la que se encarga del avance del robot. Una aplicación de esta idea fue desarrollada por M. Yue *et ál.* en su robot esférico HIT [22], en el cual un mecanismo se encarga de orientar a la IDU haciéndola girar sobre un riel colocado a lo largo del ecuador de la esfera, y otro mecanismo completamente independiente se encarga de hacerla avanzar.

Los robots con IDU tienen la ventaja de ser muy fáciles de controlar, debido a que la esfera tendrá las mismas características de holonomicidad que la IDU. Es decir, si la IDU es un vehículo diferencial, no holonómico por su configuración, la esfera será igualmente no holonómica, y tendrá la misma maniobrabilidad que el vehículo; en cambio, si la IDU es un vehículo omnidireccional, entonces la esfera también lo será. De esta forma, un control del movimiento de la IDU dentro del robot es suficiente, a bajas velocidades, para controlar el movimiento del robot.

El par τ que la IDU es capaz de producir, en condiciones estáticas y confinando el movimiento a un plano, es:

$$\tau = mgr\cos(\theta)$$

Figura 2.5 Sistema de orientación del robot HIT [22].

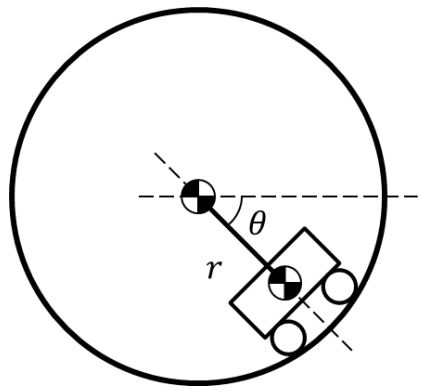


Figura 2.6 Diagrama simplificado de un robot esférico con IDU.

donde m es la masa de la IDU, g es la aceleración de la gravedad, r es la distancia del centro de la esfera al centro de masa de la IDU y θ es el ángulo que se forma entre la horizontal y la recta entre los centros de masa de esfera de IDU.

2.1.2 Péndulo

Un diseño muy utilizado en robots esféricos es el diseño con un péndulo como sistema de locomoción. Este diseño está formado por un eje que pasa por el centro de la esfera sobre el cual cuelga un péndulo. El giro hacia delante o atrás sobre este eje permite que la esfera avance o retroceda, mientras que un giro adicional del péndulo hacia la derecha o izquierda hace que la esfera cambie de dirección.

Aunque en esencia el movimiento del péndulo es omnidireccional, su movilidad está limitada por restricciones mecánicas, debido a que el péndulo sólo puede tener giro continuo sobre uno de sus ejes, mientras que el otro giro está limitado a un cierto ángulo. Esto ocasiona que el robot pierda la holonomicidad y que se puede mover únicamente en línea recta o trazando arcos con cierto radio determinado por la inclinación del péndulo.

Uno de los robots más conocidos que utilizan este sistema de locomoción es el robot de vigilancia GroundBot desarrollado por la compañía Rotundus [17]. El sistema de péndulo de este robot se muestra en la figura 2.7. Para llevar a cabo tareas de vigilancia, este robot tiene incorporadas en el interior de la esfera dos cámaras que permiten transmitir vídeo en 3D, y al estar completamente sellado, puede moverse en lodo, nieve, hielo, arena e incluso es capaz de flotar en agua.

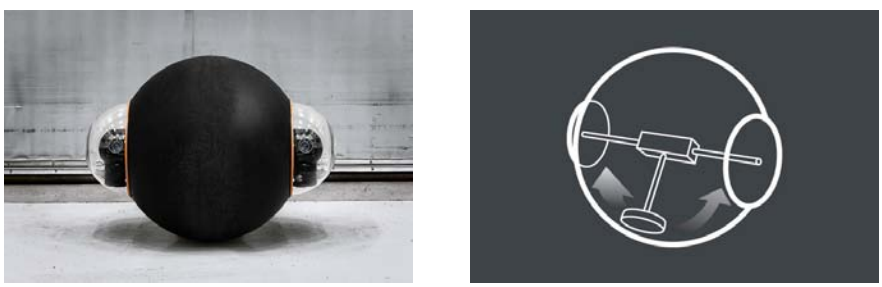


Figura 2.7 Robot GroundBot, desarrollado por Rotundus (izquierda) y su configuración interna (derecha) [17].

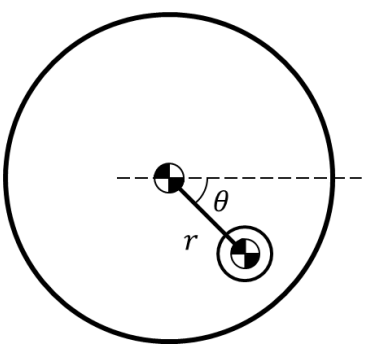


Figura 2.8 Diagrama simplificado de un robot esférico con péndulo.

El par que puede producir el péndulo, siguiendo las mismas convenciones que para

la figura 2.6, se describe con la expresión:

$$\tau = mgr\cos(\theta)$$

2.1.3 Arreglo de masas móviles

Este sistema, al igual que la IDU o el péndulo, busca redistribuir la masa dentro de la esfera para provocar un movimiento, pero lo hace de una forma un poco diferente. R. Mukherjee *et ál.* propusieron para su robot Spherobot [16] un diseño en el que se tiene un arreglo de cuatro masas que se pueden desplazar a lo largo de cuatro barras radiales. Al cambiar la posición de las masas a lo largo de las barras, el centro de masa de la esfera va cambiando también, produciendo un movimiento. Además, este robot cuenta con patas telescópicas que lo sostienen cuando está en reposo y una cámara que permite grabar vídeo.

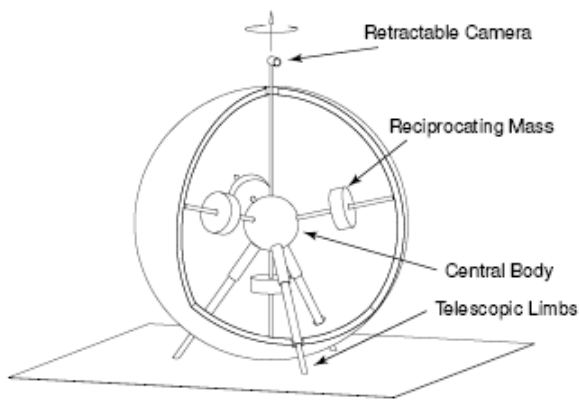


Figura 2.9 Robot Spherobot, desarrollado por R. Mukherjee *et ál.* [16].

Otra propuesta es la desarrollada por A. Javadi y P. Mojabi con el robot August [13]. El arreglo de cuatro masas móviles es similar al del Spherobot, y las masas son desplazadas utilizando motores paso a paso. Una ventaja de este robot sobre el Spherobot es que, al no tener aditamentos como las patas o la cámara, el cuerpo del robot está completamente balanceado cuando las masas se encuentran en una misma posición respecto al centro de la esfera, lo que hace el control del robot mucho más sencillo.

Un diseño alternativo propuesto por J. Lux [15] consiste en una única masa colocada al centro de la esfera y sostenida por cuatro cables. La masa la conforman

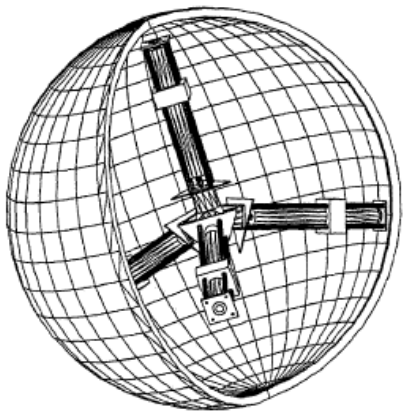


Figura 2.10 Robot August, desarrollado por A. Javadi y P. Mojabi [13].

los sistemas de control del robot y un mecanismo capaz de retraer o aflojar los cables, de forma que al hacerlo de manera coordinada, la masa puede desplazarse del centro de la esfera, generando un par que la mueve.

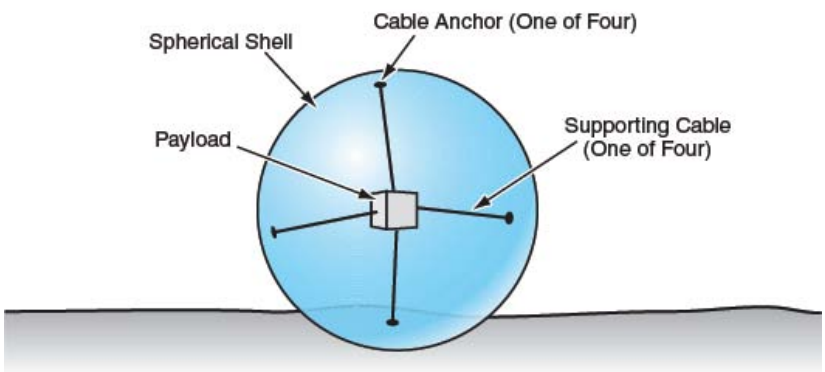


Figura 2.11 Diseño propuesto por J. Lux [15].

Para el arreglo de masas móviles, donde m_i es la masa del elemento i del arreglo, g es la aceleración de la gravedad, y $\mathbf{r}_i$ es un vector que va del centro de la esfera al centro de masa de cada maso móvil i

$$\boldsymbol{\tau} = \sum \mathbf{r}_i \times \begin{bmatrix} 0 \\ 0 \\ -m_i g \end{bmatrix}$$

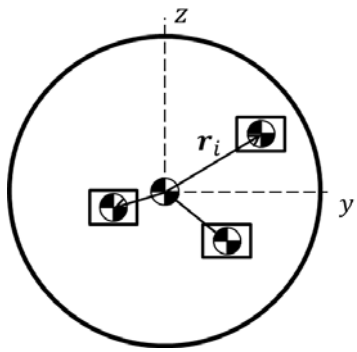


Figura 2.12 Esquema simplificado de un robot con arreglo de masas móviles.

2.1.4 Giroscopios de control de momento

Las investigaciones más recientes sobre robots esféricos han explorado el uso de volantes de inercia para controlar su movimiento. El robot Gyrover, desarrollado por H. Brown y Y. Xu [9], aunque no tiene forma esférica, sino más bien de elipsoide, es un "disco" vertical que tiene por dentro un volante de inercia que cuelga sobre el eje horizontal del cuerpo del robot a manera de péndulo. Un motor hace avanzar la esfera mediante una transmisión de engranes y mantiene el giro del volante de inercia, mientras que otro sistema produce una inclinación en el volante que, por conservación de momento angular, genera un par de precesión que hace que el robot gire hacia la derecha o izquierda, logrando así cambiar la dirección del robot.

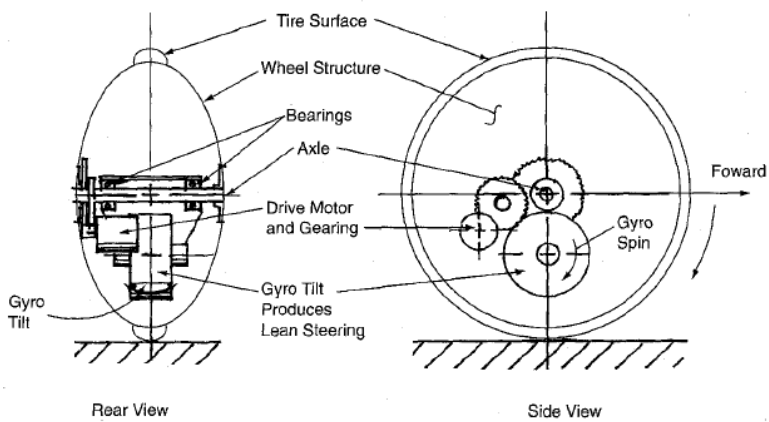


Figura 2.13 Robot Gyrover, desarrollado por H. Brown y Y. Xu [9].

Para entender de mejor forma el funcionamiento de este robot se presentan las ecuaciones de movimiento rotacional de cuerpos rígidos.

$$\sum \mathbf{M}_G = \dot{\mathbf{H}}_G \tag{2.1}$$

establece que la suma de todos los pares externos que actúan sobre el cuerpo alrededor de un punto fijo (en este caso el centro de masa G) son iguales a la razón de cambio del momento angular total alrededor de ese mismo punto. Si se introduce un sistema de referencia móvil xyz que se mueve con velocidad angular $\mathbf{\Omega}$ respecto al sistema de referencia inercial, la ecuación 2.1 cambia a:

$$\sum \mathbf{M}_G = (\dot{\mathbf{H}}_G)_{xyz} + \mathbf{\Omega} \times \mathbf{H}_G \tag{2.2}$$

donde $(\dot{\mathbf{H}}_G)_{xyz}$ es la razón de cambio de $\mathbf{H}$ alrededor de G respecto al sistema de referencia móvil xyz. Desarrollando la ecuación 2.2 y descomponiéndola sobre los ejes x, y, z se obtienen las siguientes ecuaciones de movimiento rotacional:

$$\sum M_x = I_x(\dot{\omega}_x)_{xyz} - I_y\Omega_z\omega_y + I_z\Omega_y\omega_z \tag{2.3}$$

$$\sum M_y = I_y(\dot{\omega}_y)_{xyz} - I_z\Omega_x\omega_z + I_x\Omega_z\omega_x \tag{2.4}$$

$$\sum M_z = I_z(\dot{\omega}_z)_{xyz} - I_x\Omega_y\omega_x + I_y\Omega_x\omega_y \tag{2.5}$$

donde I_x, I_y, I_z son los momentos de inercia del cuerpo, asumiendo que los ejes x, y, z son ejes principales de inercia; $(\dot{\omega}_x)_{xyz}$, $(\dot{\omega}_y)_{xyz}$, $(\dot{\omega}_z)_{xyz}$ son las aceleraciones angulares del cuerpo calculadas respecto al sistema de referencia móvil xyz; ω_x, ω_y, ω_z representan la velocidad angular del cuerpo con respecto al sistema inercial y Ω_x, Ω_y, Ω_z la velocidad angular del sistema inercial con respecto al sistema acoplado al cuerpo. Los detalles de la derivación de estas ecuaciones se pueden consultar en el capítulo 21 del libro *Engineering Mechanics: Dynamics* de R. C. Hibbeler [12].

Tomando como ejemplo el volante del robot *Gyrover*, cuyo diagrama simplificado se muestra en la figura 2.14, se seleccionó un sistema de referencia xyz con origen en el centro de masa del volante y que se mueve con él, únicamente teniendo giro sobre el eje y. Si se asume que el volante tiene una componente de velocidad angular sobre y constante, y gracias a la selección del sistema de referencia móvil,

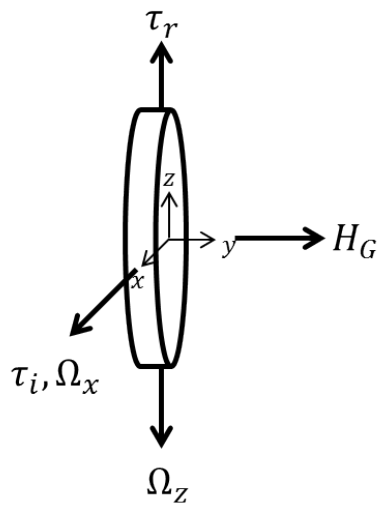


Figura 2.14 Diagrama de un volante de inercia vertical.

el primer término de las ecuaciones 2.3, 2.4 y 2.5 se puede eliminar, resultando en las ecuaciones simplificadas:

$$\sum M_x = -I_y \Omega_z \omega_y + I_z \Omega_y \omega_z \tag{2.6}$$

$$\sum M_y = -I_z \Omega_x \omega_z + I_x \Omega_z \omega_x \tag{2.7}$$

$$\sum M_z = -I_x \Omega_y \omega_x + I_y \Omega_x \omega_y \tag{2.8}$$

Si se aplica un par de inclinación τ_i sobre el eje x, de la ecuación 2.6 se obtiene:

$$\sum M_x = \tau_i = -I_y \Omega_z \omega_y$$

Esto quiere decir que, al aplicar un par de inclinación sobre x a un volante con giro en y, si el volante esta sólo y anclado en su centro a algún punto fijo sobre el que puede rotar libremente, va a reaccionar con una velocidad en z, que se puede calcular como:

$$\Omega_z = -\frac{\tau_i}{I_y \omega_y}$$

Si al producirse este "efecto" en z, existe alguna resistencia externa (como sucede en el sistema real), como fricción o la oposición de algún obstáculo, que se aplica

sobre el cuerpo como un par τ_r, entonces de la ecuación 2.8 se tiene que:

$$\sum M_z = \tau_r = I_y \Omega_x \omega_y$$

El par de reacción del sistema ante esta resistencia es el par que se puede aprovechar para darle movimiento a la esfera del robot, pero por equivalencia, esto implica que se producirá el mismo efecto sobre el eje x, con la misma dirección que el par τ_i, de manera que también se tiene que:

$$\Omega_x = \frac{\tau_r}{I_y \omega_y}$$

Al conjunto de un volante de inercia y un sistema mecánico capaz de inclinarlo sobre un eje perpendicular a su eje de giro se le conoce como giroscopio de control de momento (CMG, por sus siglas en inglés), pudiendo ser éste simple, si la inclinación es sobre un eje, o doble, si la inclinación es sobre dos.

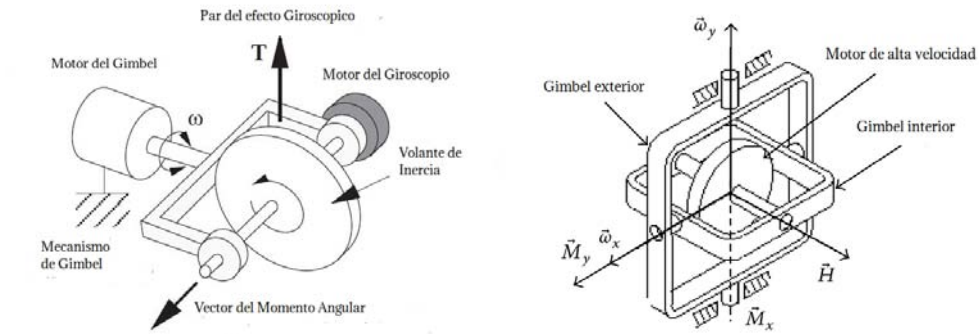


Figura 2.15 CMG simple (izquierda) y CMG doble (derecha).

Una desventaja del uso del CMG es que el par de salida aprovechable únicamente tiene la dirección deseada por un instante, ya que conforme el volante es inclinado, la dirección del par cambia, haciendo que la componente en la dirección deseada disminuya, mientras que aumenta la componente en otra dirección perpendicular, causando efectos indeseados en la esfera. Además, el ángulo máximo que se puede inclinar el volante es 90°, ya que a partir de ese ángulo el par de salida comienza a crecer en sentido opuesto.

Una solución a este problema fue propuesta por G. Schroll [18] en su prototipo de robot esférico, y consiste en utilizar dos CMG en lugar de uno, cuyos volantes al girar con la misma velocidad pero en sentidos opuestos y al ser inclinados en

direcciones opuestas también, tienen un par de salida cuyas componentes se suman en la dirección deseada, pero que se contrarrestan en la dirección no deseada.

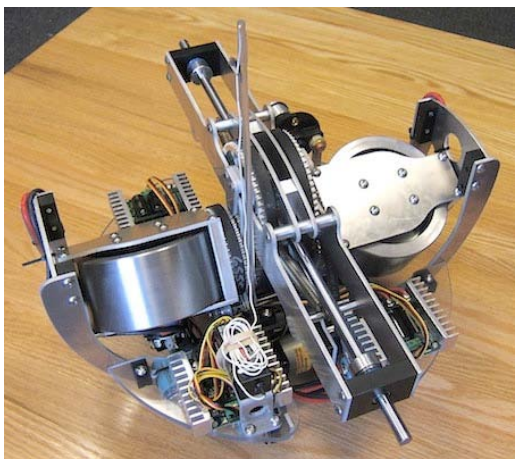


Figura 2.16 Ensamble de los CMG del prototipo de G. Schroll [19].

El prototipo de G. Schroll utiliza una combinación de péndulo con CMG, en la que el péndulo se encarga de darle movimiento de avance y giro al robot. Como se puede ver en la figura 2.16, los volantes giran, en su posición inicial, alrededor del eje z, lo que produce un par en la esfera en el sentido de avance, y es utilizado para crear impulsos de par en el robot que le permitan superar obstáculos e incluso subir escalones. Aunque el problema de los efectos indeseados fue resuelto con este prototipo, el par de salida en la dirección deseada sigue siendo limitado, por lo que estos impulsos no tienen mucha duración, y para poder utilizar los volantes para crear un nuevo impulso es necesario regresarlos lentamente a su posición inicial, y de manera que el par generado no sea capaz de superar la fricción entre la esfera y el suelo, y la esfera mantenga su posición.

Una forma alternativa de utilizar volantes de inercia se desarrolló en el prototipo de V. Joshi *et ál.* [14], que está formado por dos volantes de inercia montados perpendicularmente sobre la superficie interna de la esfera. Para que el centro de masa de la esfera coincida con su centro geométrico, se agregó peso muerto en una posición diametralmente opuesta a cada uno de los volantes. La principal diferencia entre este diseño y uno que utiliza CMG es que el par necesario para mover la esfera no se genera inclinando el eje de los volantes de inercia, sino al variar la velocidad de giro de los volantes, de forma que estos producirán un par

de reacción que se opone al cambio en el momento angular.

Figura 2.17 Prototipo de V. Joshi *et al.* [14].

2.2 Propuestas de configuración mecánica

Para realizar la selección de una de las propuestas de configuración mecánica, se determinaron ciertos criterios de selección, a los que se les dio una ponderación según su importancia. Dichos criterios se muestran en la tabla 2.1.

Se realizaron cinco propuestas de configuraciones mecánicas para el prototipo que incorporan uno o más sistemas de locomoción diferentes. Se procuró que no fueran propuestas redundantes, sino que presentaran algunas diferencias importantes en cuanto diseño y funcionamiento, y también que todas ellas fueran capaces de darle a la esfera al menos un movimiento simple sobre un plano. Las propuestas son las siguientes:

- Propuesta 1. **Tres CMG** dispuestos a la misma altura en un arreglo que puede ser triangular o en estrella. Esta configuración es capaz de darle movimiento omnidireccional a la esfera, ya que cuenta con el mínimo número de CMG necesarios para hacerlo.

- Propuesta 2. **Cuatro CMG** acomodados sobre cada una de las caras de un tetraedro inscrito en la esfera, en un arreglo "radial" o sobre las cuatro caras laterales de un cubo inscrito en la esfera. La introducción de cuatro CMG da

Tabla 2.1 Criterios de selección para las configuraciones mecánicas.

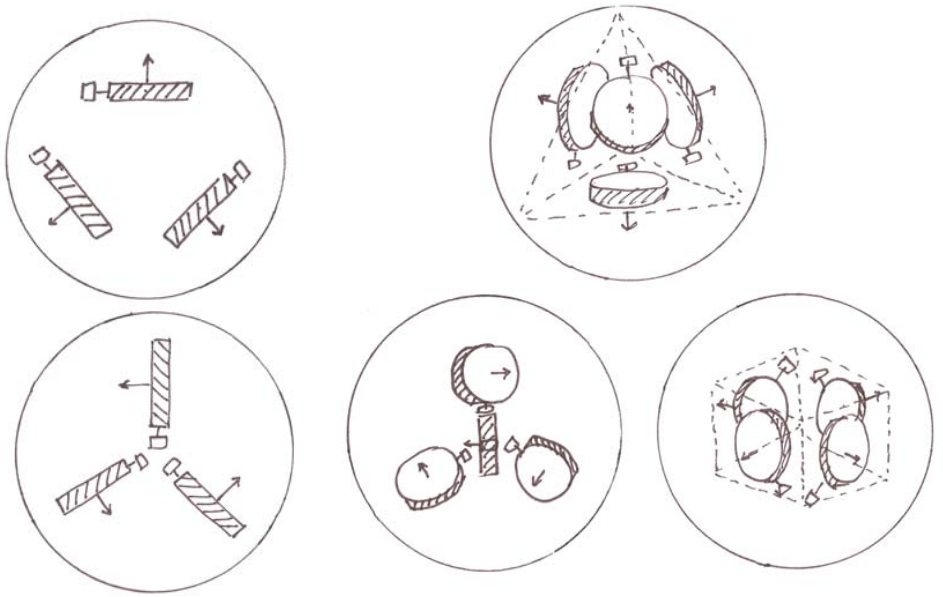
Criterio			Importancia 1 a 5
Control		Estabilidad intrínseca	4
	Automático	Linealizabilidad	3
		Observabilidad	5
		Número de GDL	4
		Economía	2
	Manual	Teleoperabilidad	5
Manufactura		Número de anillos de deslizamiento	5
		Número de ejes	3
		Número de volantes de inercia	5
		Simplicidad de la mecánica interna	5
		Facilidad de ensamble	3
Operación		Movilidad	5
		Autonomía	4
		Holonomicidad	2
		Funcionamiento continuo	4
		Paso de obstáculos	5

a la esfera movimiento omnidireccional, formando un sistema sobreactuado, pero facilita considerablemente el control del sistema.

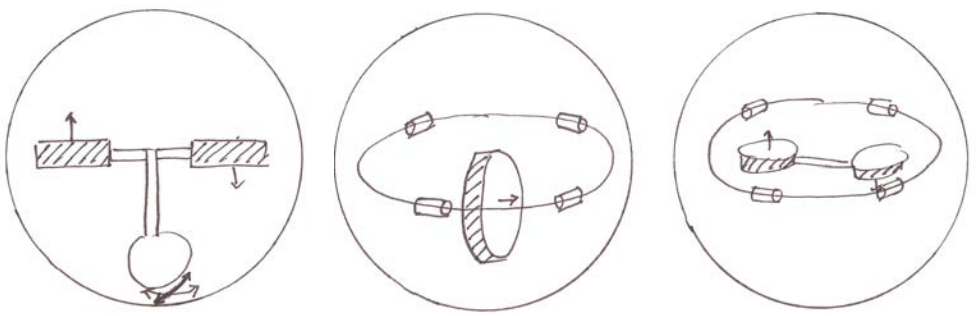
- Propuesta 3. **Un péndulo** de dos grados de libertad (GDL) que proporciona movimiento (no omnidireccional) sobre un plano y **dos CMG** que pueden utilizarse para superar obstáculos.
- Propuesta 4. **Una IDU** de un GDL para movimiento de avance y **un CMG** para controlar la dirección. Otra variante de esta misma configuración es una IDU de dos GDL y un CMG para estabilización.
- Propuesta 5. **Una IDU** de dos GDL para movimiento sobre un plano y **dos CMG** para superar obstáculos.

En la figura 2.18 se muestran los bocetos que se dibujaron para cada una de las propuestas mencionadas.

Considerando cada uno de los criterios establecidos, cada propuesta fue calificada con un valor entero entre -2 y 2, tomando en cuenta si el criterio se desea maximizar o minimizar. La suma de los productos de la importancia del criterio por la calificación asignada a cada propuesta es igual al puntaje total de la propuesta.



(a) Configuración 1 (b) Configuración 2



(c) Configuración 3 (d) Configuración 4 (e) Configuración 5

Figura 2.18 Bocetos de las configuraciones mecánicas propuestas.

Los resultados de la evaluación se muestran en la tabla 2.2.

La propuesta de configuración con mayor puntaje fue la número 3, que consiste en un péndulo de dos GDL y dos CMG. Las propuestas de diseño mecánico, que se presentarán más adelante, se harán utilizando esta configuración. La configuración con el segundo puntaje más alto fue la número 5, que consiste en una IDU de dos GDL y dos CMG. Debido a las similitudes entre las dos configuraciones y al hecho de que el sistema con péndulo y la IDU tienen un comportamiento muy parecido, se decidió hacer las propuestas de diseño mecánico con base en cualquiera de las dos configuraciones, permitiéndonos así ampliar las posibilidades de diseño.

Tabla 2.2 Resultados de la evaluación de las configuraciones propuestas.

Criterio	Importancia	Objetivo	Config. 1	Config. 2	Config. 3	Config. 4	Config. 5
Estabilidad intrínseca	4	+	0	-1	2	1	1
Linealizabilidad	3	+	-1	-2	2	2	2
Observabilidad	5	+	2	2	0	0	0
Número de GDL	4	−	-1	-2	2	0	2
Economía	2	+	-1	-2	2	1	1
Teleoperabilidad	5	+	-2	-2	1	2	1
Número de anillos de deslizamiento	4	−	-1	-2	0	1	0
Número de ejes	3	−	-1	-2	1	1	1
Número de volantes de inercia	5	−	-1	-2	0	1	0
Simplicidad de la mecánica interna	5	−	-1	-2	1	1	1
Facilidad de ensamble	3	+	0	-2	0	1	1
Movilidad	5	+	-2	-2	2	2	2
Autonomía	4	+	0	0	1	2	1
Holonomicidad	2	+	1	2	1	0	1
Funcionamiento continuo	4	+	-1	-1	1	1	1
Paso de obstáculos	5	+	2	2	1	-2	1
Total	63		-28	-62	64	54	57

2.3 Propuestas de diseño mecánico

Después de explorar muchas ideas, se formularon tres propuestas de diseño mecánico para ser evaluadas de forma similar a la evaluación hecha para las configuraciones. La propuesta seleccionada después de esta evaluación, será la que se desarrollará a detalle para el prototipo. Los criterios seleccionados para esta nueva evaluación y su importancia se muestran en la tabla 2.3.

Tabla 2.3 Criterios de selección para las propuestas de diseño mecánico.

Criterio	Importancia 1 a 5
Número de componentes	2
Costo de componentes	2
Facilidad de manufactura	5
Facilidad de ensamble	3
Necesidad de balanceo	4
Factibilidad mecánica	3
Accesibilidad de componentes	4
Mantenimiento	2

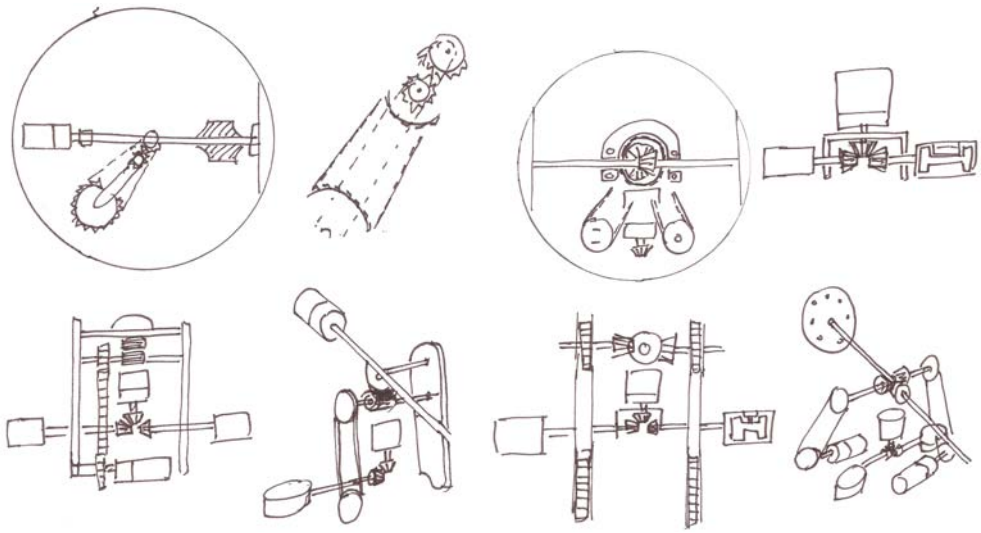
Las propuestas de diseño mecánico son las siguientes:

- **Propuesta 1.** Consiste en un péndulo de dos GDL en el que el primer GDL es actuado directamente sobre el eje del péndulo a través de un motor montado sobre la esfera. Se requiere una masa colocada en posición opuesta al motor para balancear la esfera. El segundo GDL se actúa mediante un motor colocado en la parte inferior del péndulo y mediante una transmisión por bandas y engranes. Los dos CMG se colocan en la parte inferior del péndulo y son actuados mediante un motor colocado de forma vertical y una transmisión de engranes cónicos.

- **Propuesta 2.** Consiste en un péndulo de dos GDL en el que los dos GDL son actuados simultáneamente mediante dos motores montados en la parte de abajo del péndulo y a través de una transmisión por bandas y un arreglo de engranes cónicos. Los CMG se colocan en la parte inferior del péndulo y son actuados mediante un motor y engranes cónicos.

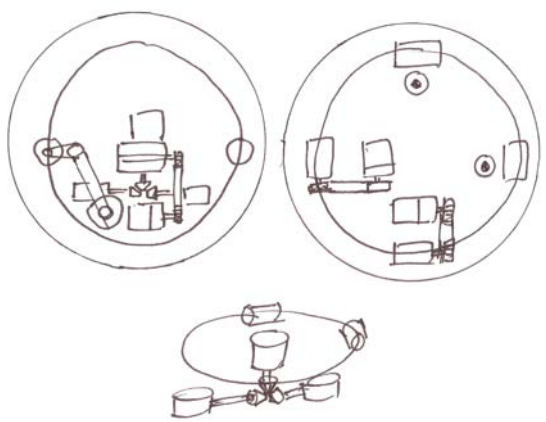
- **Propuesta 3.** Consiste en una IDU de 2 GDL que se mueve mediante dos rodillos en contacto con la esfera. Los rodillos son actuados mediante motores colocados en la parte inferior de la IDU y mediante una transmisión por bandas. Los CMG son colocados en la parte inferior de la IDU y son actuados mediante un motor y engranes cónicos.

En la figura 2.19 se muestran los bocetos dibujados para cada una de las propuestas.



(a) Diseño 1

(b) Diseño 2



(c) Diseño 3

Figura 2.19 Bocetos de las propuestas de diseño mecánico.

Los resultados de la evaluación realizada se muestran en la tabla 2.4.

Tabla 2.4 Evaluación de las propuestas de diseño mecánico.

Criterio	Importancia	Objetivo	Diseño 1	Diseño 2	Diseño 3
Número de componentes	2	−	1	0	0
Costo de componentes	2	−	1	0	1
Facilidad de manufactura	5	+	1	1	0
Facilidad de ensamble	3	+	1	0	2
Necesidad de balanceo	4	−	-1	2	-2
Factibilidad mecánica	3	+	2	2	2
Accesibilidad de componentes	4	+	2	2	2
Mantenimiento	2	−	1	1	1
Total	25		24	29	16

La propuesta con el mayor puntaje es la propuesta 2, que consiste en un péndulo de dos GDL actuados simultáneamente mediante dos motores montados en la parte inferior del péndulo y con una transmisión por bandas y un arreglo de engranes cónicos. Esta propuesta de diseño mecánico será utilizada para diseñar el prototipo de robot esférico, proceso que se detalla en el capítulo 4.

Capítulo 3

Modelado matemático

El modelado matemático de los sistemas mecánicos es un tema de profundo interés para la física y la ingeniería. Un modelo puede "calcularse" de manera previa a alguna prueba, o antes de elaborar un diseño, permitiendo "simular" el sistema en estudio sin necesidad de implementarlo físicamente, y hacer evaluaciones con base en el resultado de tales simulaciones. Esto es útil para obtener parámetros de diseño, estudiar la factibilidad de funcionamiento, y sirve como forma de evaluación y realimentación de los cambios en el diseño del sistema, al poder estudiar su comportamiento y evolución en el tiempo, mediante las simulaciones.

Un modelo de un sistema es el conjunto de ecuaciones diferenciales que, resueltas generalmente de forma numérica, permiten hacer una evaluación más o menos fiel de la evolución del sistema a lo largo del tiempo (esto es, la "trayectoria" de sus grados de libertad, coordenadas o estados independientes). Por ejemplo, el modelo matemático de un sistema consistente de un péndulo simple, considera sus dimensiones, masa, inercia y fricción. Tal modelo sirve para hacer experimentos simulados sobre el comportamiento del péndulo en diversas circunstancias, como una velocidad o valor inicial de su ángulo, un cambio de la longitud de su eje, o la manipulación de su masa o la fricción. El péndulo físico seguiría, por algún periodo de tiempo apreciable, una pauta de comportamiento bastante similar, o al menos comparable a la simulada, y ayudaría a tomar decisiones de diseño. El estudio del sistema puede extenderse para obtener un controlador que intente gobernar algún aspecto de su comportamiento como, en el caso particular del péndulo simple, mantenerlo invertido.

Existen diferentes formas de modelar un sistema matemáticamente. Una de ellas, la forma de Newton o Newton-Euler, consiste en establecer, para todos los objetos en el sistema, las ecuaciones de balance de fuerzas y pares (y como resultado de la segunda ley de Newton, se involucran las segundas derivadas de los grados de libertad), generalmente sobre coordenadas cartesianas y sus posibles rotaciones en cada cuerpo; y simplificar el sistema resultante. Si bien esta forma es correcta, es difícil cambiar parámetros de diseño bajo tal esquema, o añadir barras adicionales, y generalmente, es más fácil brindar al sistema información de fuerzas, pares y masas, para resolver velocidades y aceleraciones.

El esquema de modelado lagrangiano, por otra parte, consiste en especificar, para cada cuerpo, su energía cinética y potencial en términos de algunas *coordenadas generalizadas* (no necesariamente cartesianas) que permitan reconstruir la totalidad del estado del sistema, y establecer las restricciones e interacciones entre los cuerpos usando tales coordenadas generalizadas. Posteriormente, se impone a la evolución de estas energías la condición de que se minimice una cantidad llamada acción. Las ecuaciones diferenciales de movimiento se derivan de esa imposición, usando las ecuaciones de Euler-Lagrange. En el proceso se involucran, a lo mucho, solamente las primeras derivadas de los grados de libertad espaciales. Esta forma de modelado, aunque algo más difícil de entender intuitivamente, es más apropiada para resolver fuerzas y pares, dadas condiciones de posiciones, ángulos y velocidades, y una elección conveniente de las coordenadas generalizadas permite obtener datos de interés del sistema directamente. La adición de cuerpos adicionales es sencilla, incluso dentro de cadenas cinemáticas ya establecidas. Sin embargo, las ecuaciones generadas bajo éste método son muy largas y difíciles de manipular, lo que hace necesario el uso de una herramienta computacional.

Se puede demostrar que ambos esquemas son matemáticamente equivalentes, capaces de capturar los mismos detalles de los sistemas físicos clásicos, y la elección de uno sobre otro se reduce a detalles de conveniencia. Sin embargo, ambas formas ignoran algunos aspectos de utilidad para la ingeniería, como el costo computacional de la evaluación del modelo, que es con frecuencia muy alto. No se exploran análogos en tiempo discreto para las ecuaciones dinámicas, que podrían reducir abaratar la simulación y control del sistema. Además, como se asume que la tierra es un sistema inercial, se ignoran las dinámicas del efecto Coriolis asociadas. En otras escalas o sistemas, tales efectos no deberían despreciarse.

En el proyecto, se usó el esquema de modelado lagrangiano en tres dimensiones para obtener parámetros de diseño de ingeniería, y para propósitos de control. Las razones para ello se resumen en lo siguiente:

- Como todos los cuerpos se consideran rígidos y se desprecian las fricciones, no hay ventaja inherente al uso de un esquema newtoniano, en el cual las fricciones son (generalmente) más sencillas de incluir.
- La posibilidad de hacer cambios rápidos en el diseño, como la adición de barras y volantes, especificando sólo las relaciones entre cuerpos como composiciones de desplazamientos y rotaciones.
- Es más fácil resolver directamente las fuerzas y pares de interés, como los demandados de los motores, usando una elección conveniente de las coordenadas generalizadas.
- Las ecuaciones diferenciales no se escriben directamente. En su lugar, se derivan de las expresiones de energía, que generalmente son más sencillas de expresar.

3.1 Modelado lagrangiano

El modelado lagrangiano necesita de las energías de todos los elementos del sistema. Se considera que todos los cuerpos son rígidos y que tienen dos energías bien definidas, una cinética y otra potencial. Existe un conjunto de ecuaciones conocido como *ecuaciones de Euler-Lagrange*. La energía se conserva, y las ecuaciones de Euler-Lagrange se limitan a describir el flujo entre las energías de los cuerpos del sistema, bajo la condición de que se minimice una cantidad llamada *acción*. El sistema quizá tenga algunas contribuciones adicionales de energía en la forma de pares, fuerzas y fricciones, que también se consideran.

Una expresión general para la energía cinética de un cuerpo rígido con masa m, vector de velocidad lineal $\mathbf{v}$, vector de velocidad angular $\boldsymbol{\omega}$ e inercia rotacional representada por la matriz $\mathbf{I}$ es

$$E_c = \frac{1}{2} m \mathbf{v} \cdot \mathbf{v} + \frac{1}{2} \boldsymbol{\omega}^T \cdot \mathbf{I} \cdot \boldsymbol{\omega} \tag{3.1}$$

Distinguiendo entre las contribuciones lineales y angulares, la energía cinética traslacional y la energía cinética rotacional son:

$$E_{ct} = \frac{1}{2} m \mathbf{v} \cdot \mathbf{v} \tag{3.2}$$

$$E_{cr} = \frac{1}{2} \boldsymbol{\omega}^T \cdot \mathbf{I} \cdot \boldsymbol{\omega} \tag{3.3}$$

Esta distinción se hace por conveniencia. Es posible seccionar el cuerpo en una vasta cantidad de partículas. Para este sistema equivalente, como energía cinética de cada partícula se considera solamente la contribución traslacional, sumando sobre todas las partículas. En el caso límite, con partículas infinitesimales, esta suma puede descomponerse, con algunos cambios de variable apropiados, en las partes traslacional y rotacional del cuerpo original.

$$E_{\mathrm{c}} = \frac{1}{2} \lim_{n \to \infty} \sum_{i=1}^{n} (m_i \mathbf{v}_i \cdot \mathbf{v}_i) = E_{cr} + E_{ct} = \frac{1}{2} \boldsymbol{\omega}^T \cdot \mathbf{I} \cdot \boldsymbol{\omega} + \frac{1}{2} m \mathbf{v} \cdot \mathbf{v} \tag{3.4}$$

Además de las energías cinéticas, deben considerarse las energías potenciales de todos los cuerpos. En particular, la energía potencial gravitatoria. Una expresión general para ésta en un cuerpo rígido de masa m, posición en el sistema origen acoplado a la tierra $\mathbf{s}$ y vector gravitatorio $\mathbf{g}$ (donde $\mathbf{g} = \begin{bmatrix} 0 & 0 & g \end{bmatrix}^T$) es:

$$E_p = m \mathbf{g} \cdot \mathbf{s} \tag{3.5}$$

Las energías potenciales gravitatorias se entienden en relación con algún sistema de referencia origen, asociado con la tierra, cuyo origen brinda el cero de esta medida de energía (algún punto del terreno, por ejemplo). Si bien esto parecería favorecer una elección de coordenadas (cartesianas), estas energías también pueden derivarse del conjunto de coordenadas generalizadas escogidas, pues es necesario (y de todos modos, se asume) que el comportamiento total del sistema se describe por completo con tales coordenadas.

Todas las expresiones de energía deben ser funciones de las coordenadas generalizadas. La elección de tales coordenadas es, por lo general, cuestión de conveniencia, pero debe estar condicionada a que todos los posibles estados del sistema puedan ser "capturados" con tales coordenadas y sus derivadas.

Se obtienen las expresiones simbólicas de ambas energías para todos los cuerpos del sistema, considerando alguna elección conveniente de coordenadas, y se calcula el lagrangiano L, que es la energía cinética total del sistema (T) menos la energía potencial total (V)

$$\begin{aligned} L &= T - V \\ L &= \sum (E_{cti} + E_{cri} - E_{p_i}) \end{aligned} \tag{3.6}$$

Y en cada una de las coordenadas generalizadas q_i, se "aplica" la ecuación de Euler-Lagrange para derivar las ecuaciones diferenciales que rigen al sistema:

$$\begin{aligned} \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} &= F_{q_i} \\ F_{q_i} &= f_{q_i} + \sum a_{ij}\lambda_j - friccion_{q_i} \end{aligned} \tag{3.7}$$

donde F_{q_i} se refiere a la *fuerza generalizada* sobre esa coordenada generalizada en particular, compuesta quizá de varias otras fuerzas f_{q_i} (para lo cual podría ser necesario hacer una descomposición o proyección de componentes, a lo largo de los ejes de las coordenadas relevantes, o añadir o descomponer cuerpos de manera que alguna coordenada generalizada coincida con la dirección de aplicación de la fuerza), y el coeficiente a_{ij} de cada λ_j es la derivada de alguna de las j ecuaciones de restricción (ecuaciones que relacionan varias coordenadas generalizadas, restringiendo o limitando su comportamiento conjunto) sobre cada coordenada generalizada q_i, y restando las posibles fricciones $friccion_{q_i}$, o sus componentes o proyecciones sobre esa coordenada generalizada. Existe un *multiplicador de Lagrange* λ_j por cada ecuación de restricción j. No interesa qué son "realmente" tales λ_j, pero se podrían describir como las "fuerzas de las restricciones" en el sistema, y, en algunos casos, podrían igualarse a fuerzas físicas, pero en general no tiene por qué ser de tal manera. La expresión para determinar cada a_{ij} es

$$a_{ij} = \frac{\partial(restriccion_j)}{\partial q_i} \tag{3.8}$$

Una ecuación de restricción es simplemente una función de todas o algunas de las coordenadas generalizadas, que las relaciona entre sí, de manera que no pueden tomar todas valores arbitrarios. Tales restricciones tienen un tratamiento más profundo y sutil, dependiendo de sus interacciones con los elementos del sistema,

pero no se tratará más que de la manera más elemental. Por simplicidad, se desea que estas ecuaciones de restricción sean siempre iguales a cero. Esto es, en general:

$$restriccion_j = f_j(q_0, q_1, \ldots, q_n) = 0 \tag{3.9}$$

La aplicación de la ecuación de Euler-Lagrange sobre cada coordenada generalizada permite derivar las ecuaciones diferenciales cuya solución describe el comportamiento del sistema modelado. Se necesita además de un conjunto de condiciones iniciales sobre todas las coordenadas generalizadas y sus derivadas, y tantas ecuaciones adicionales de entrada como haya fuerzas de entrada al sistema.

3.2 Problemas de implementación

Si bien el modelado es sencillo en su descripción general de alto nivel, hay varias consideraciones importantes al momento de obtener el modelo de un sistema real. Entre estas:

1. *Cómo escoger las coordenadas generalizadas de forma conveniente.*

   Se ha tomado como regla general escoger como coordenadas generalizadas, donde sea posible, el ángulo de rotación de los ejes mecánicos de los cuerpos (esfera, péndulo y volantes). Por ejemplo, las rotaciones con respecto al centro geométrico de la esfera del péndulo, o los ángulos de ejes de rotación de las cajas de los volante de inercia. Estas coordenadas generalizadas no coinciden necesariamente con los ejes de los motores, pero están dispuestas de tal manera que su relación con estos es lineal. Cada cuerpo tiene acoplado un sistema de referencia que se mueve y rota con el cuerpo, permaneciendo invariante en relación. Estos sistemas están dispuestos de tal forma que el origen de cada uno coincide con el centro de masa o algún punto de interés de cada cuerpo al que está asociado. El sistema de referencia acoplado a la tierra es el sistema origen. En el resto de los casos, se toman como coordenadas generalizadas los desplazamientos o giros en los ejes x, y y z de los sistemas de los cuerpos, o del origen, como sea conveniente.

2. *Cómo establecer ecuaciones de restricción sobre tales coordenadas generalizadas cuando se requiera.*

   En particular, ¿cómo establecer alguna ecuación o ecuaciones de restricción sobre la esfera de manera que el contacto entre la esfera y el terreno, que se asume perfectamente plano, sea modelado sin deslizamiento? Las demás posibles restricciones bien pueden expresarse directamente con la expansión de las fuerzas de entrada en las ecuaciones de Euler-Lagrange, quizá como fricciones viscosas con coeficientes arbitrariamente grandes. Siendo así, se puede establecer como única restricción que la velocidad de la superficie de la esfera, en su punto inferior, sea siempre opuesta a la velocidad de su centro geométrico ($\mathbf{v}$), involucrando como coordenadas generalizadas a x y y medidas desde el origen hasta el centro geométrico de la esfera, la velocidad angular (vectorial) de la esfera ($\boldsymbol{\omega}$), y el vector que va de su centro al punto de contacto con el suelo, en términos del sistema inercial, y considerando que la esfera tiene radio r_{esf}. La expresión vectorial resultante, que deberá luego ser descompuesta en un conjunto de ecuaciones escalares, es:

$$\mathbf{v} - \begin{bmatrix} 0 \\ 0 \\ -r_{esf} \end{bmatrix} \times \boldsymbol{\omega} = \mathbf{0} \tag{3.10}$$

3. *Cómo hallar los componentes de velocidad lineal y angular cuando se tienen varios cuerpos encadenados entre sí, algunos de los cuales rotan sobre los tres ejes coordenados, algunos cuerpos con respecto a algún sistema acoplado a otro cuerpo, y así sucesivamente, en una cadena cinemática.*

   Se ha escogido un paradigma más bien práctico. Se especifica la rotación "total" de cada cuerpo por medio de la multiplicación de matrices de rotación parciales, por conveniencia, sobre los tres ejes x, y, z, y cada desplazamiento por medio de vectores de distancia. La división de la energía de cada cuerpo en una componente lineal y una rotacional permite, para la parte lineal, obtener la posición del centro de masa, por medio de la composición y suma de tales matrices y vectores. La velocidad angular puede derivarse de la matriz de rotación total.

   Otros esquemas podrían hacer uso de cuaterniones, matrices homogéneas capaces de representar al mismo tiempo rotaciones y desplazamientos, o dividir

todos los cuerpos en un vasto número de partículas masivas, y considerar solamente sus contribuciones de energía cinética lineal y energía potencial.

El vector de velocidad angular de un cuerpo puede derivarse de la matriz de rotación "total" $\mathbf{A}(t)$, que representa la transformación lineal desde vectores dados en términos del sistema acoplado al cuerpo "final", a vectores en términos del sistema origen acoplado a la tierra. La relación usada es:

$$\mathbf{W} = \frac{d\mathbf{A}(t)}{dt} \cdot \mathbf{A}(t)^{-1} \tag{3.11}$$

donde $\mathbf{W}$ es el *tensor de velocidad angular*, tres de cuyos componentes son las velocidades angulares del cuerpo sobre los ejes x, y, z:

$$\mathbf{W} = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \tag{3.12}$$

4. *Cómo expresar el tensor de inercia de los cuerpos que rotan, si en general, para la obtención de energías, este tensor debe escribirse en términos de algún sistema origen, y generalmente, sólo se dispone del tensor diagonal sobre los ejes principales de cada cuerpo, hallado con algún sistema CAD.*

   Se aprovecha que el resultado de energía es el mismo si, en lugar de considerar la velocidad angular del cuerpo con respecto al sistema origen acoplado a la tierra, se usa la velocidad angular del sistema origen, con respecto al sistema del cuerpo, que ahora se considera estático. De esta forma se dejan los tensores de inercia intactos.

   Para obtener la velocidad angular del sistema origen en términos del sistema acoplado al cuerpo, se necesita la matriz de rotación que representa la transformación de vectores dados en la base del sistema origen, a vectores en la base del sistema acoplado al cuerpo "final". Puede demostrarse que esta transformación es la transpuesta de la matriz $\mathbf{A}(t)$ dada en 3.11. El tensor de velocidad angular aplicado a esta matriz permite recuperar la velocidad angular del sistema origen, que en realidad permanece estático, en términos del sistema del cuerpo, que se considera fijo solamente para este paso. Hay que considerar una corrección por cambio de signo, quedando la expresión final como se muestra en 3.13.

$$\mathbf{W}_{O_{sys}} = -\frac{d\mathbf{A}(t)^T}{dt} \cdot \left(\mathbf{A}(t)^T\right)^{-1} \tag{3.13}$$

5. *Cómo aplicar la ecuación de Euler-Lagrange, si las expresiones de energía son todas simbólicas, quizá de considerable longitud, y resolver numéricamente las ecuaciones algebraicas y diferenciales resultantes.*

   Se hace uso de Mathematica para el manejo de las expresiones simbólicas. Las ecuaciones de movimiento generadas también se resuelven numéricamente usando el mismo software.

3.3 Descripción del sistema

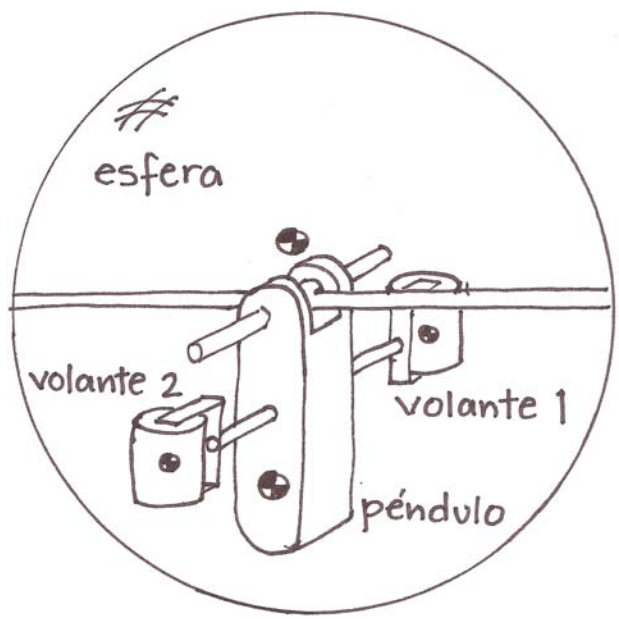


Figura 3.1 Diagrama del sistema del robot.

El sistema del robot consiste, primero, de una esfera de contención, que para el modelado se asume perfecta, rígida y simétrica, por lo que tiene un tensor de inercia diagonal. La esfera puede girar alrededor de cualquiera de los ejes (x, y, z) del sistema de referencia acoplado a la misma, expresando la rotación "total" como una composición de tres rotaciones parciales. Entre la esfera y el suelo no hay deslizamiento (aunque la esfera puede rotar libremente alrededor del eje z

del sistema origen acoplado a la tierra, quizá con alguna fricción en oposición). Uno de los diámetros de la esfera corresponde a un eje metálico fijo a ésta. Un péndulo actuado sobre dos grados de libertad parte del centro del eje diametral de la esfera.

El péndulo tiene dos grados de libertad rotacionales. Los ejes de estos grados de libertad son: el eje diametral de la esfera, y una ortogonal a tal eje y a la línea que parte del centro geométrico de la esfera y termina en el centro de masa del péndulo. También se asume que el péndulo tiene un tensor de inercia diagonal.

Al cuerpo del péndulo se acoplan las "cajas" de los dos volantes a través de ejes actuados. Un único motor rota las dos cajas al mismo tiempo y en direcciones opuestas (aunque para el modelado, se asume que existe un motor para cada caja, pero ambos motores se mueven en oposición), para mitigar los efectos indeseados de la rotación de giroscopios. Dentro de cada una de las cajas de los volantes, un motor trifásico sin escobillas controlado en velocidad, en adelante motor *brushless*, rota los volantes a una velocidad angular elevada (superior a varios miles de rpm) necesaria para su uso efectivo.

Los ángulos (las coordenadas generalizadas) y su relación con los sistemas coordenados acoplados a cada cuerpo se detallan en la siguiente tabla:

Tabla 3.1 Ángulos y su relación con los sistemas de referencia de cada cuerpo.

Cuerpo	Ángulo	Eje
Esfera	ψ_{esf}	z
Esfera	θ_{esf}	y
Esfera	ϕ_{esf}	x
Péndulo	β_{pend}	x
Péndulo	α_{pend}	y
Volante 1	φ_{vol_1}	y
Volante 1	γ_{vol_1}	z
Volante 2	φ_{vol_2}	y
Volante 2	γ_{vol_2}	z

En el caso de los desplazamientos lineales o distancias:

Tabla 3.2 Distancias y su relación con los sistemas de referencia

Cuerpo	Distancia	Eje
Esfera	x	x
Esfera	y	y

En total, hay 11 grados de libertad o coordenadas generalizadas en el sistema. Para los ángulos que se usan en cadenas cinemáticas, se define un "nivel" en la cadena como un número natural, correspondiente al orden en que suceden las rotaciones desde el sistema inercial hasta cada uno de los cuerpos, a manera de un árbol. Esto es:

Tabla 3.3 Nivel de los ángulos en la cadena cinemática.

CG	Índice	Nivel
x	1	0
y	2	0
ψ_{sph}	3	1
θ_{sph}	4	2
ϕ_{sph}	5	3
β_{pend}	6	4
α_{pend}	7	5
φ_{vol_1}	8	6
φ_{vol_2}	9	6
γ_{vol_1}	10	7
γ_{vol_2}	11	7

Por tanto, la cadena cinemática de las rotaciones de la esfera que va del sistema inercial al sistema de la esfera es:

$$\psi_{sph} \rightarrow \theta_{sph} \rightarrow \phi_{sph} \tag{3.14}$$

Para el péndulo es:

$$\psi_{sph} \rightarrow \theta_{sph} \rightarrow \phi_{sph} \rightarrow \beta_{pend} \rightarrow \alpha_{pend} \tag{3.15}$$

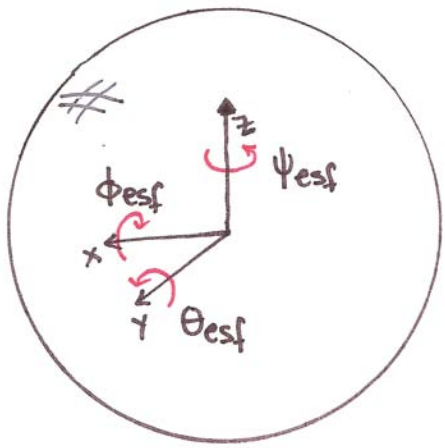
Para el primer volante es:

$$\psi_{sph} \rightarrow \theta_{sph} \rightarrow \phi_{sph} \rightarrow \beta_{pend} \rightarrow \alpha_{pend} \rightarrow \varphi_{vol_1} \rightarrow \gamma_{vol_1} \tag{3.16}$$

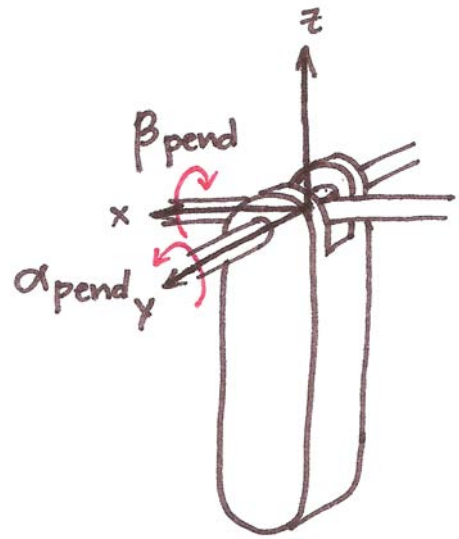
Y para el segundo volante es:

$$\psi_{sph} \rightarrow \theta_{sph} \rightarrow \phi_{sph} \rightarrow \beta_{pend} \rightarrow \alpha_{pend} \rightarrow \varphi_{vol_2} \rightarrow \gamma_{vol_2} \tag{3.17}$$

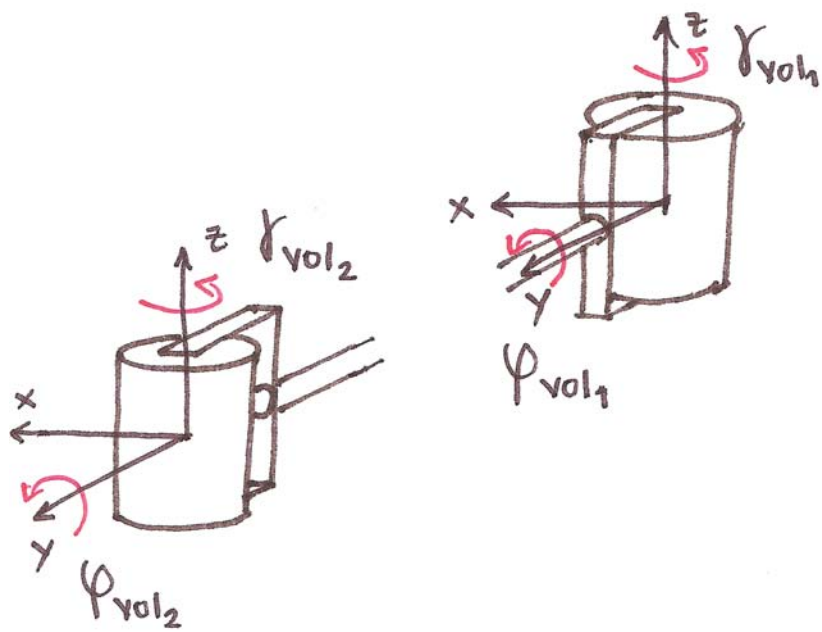
Las rotaciones de cada uno de los cuerpos se muestran en la figura 3.2.



(a) Esfera



(b) Péndulo



(c) Volantes

Figura 3.2 Rotaciones de cada cuerpo del sistema del robot.

3.4 Procedimiento de modelado

1. *Establecer las rotaciones del sistema con matrices de rotación.*

   Se consideran matrices de rotación ortonormales, de determinante unitario (de forma que se preservan ángulos, distancias y orientaciones), sobre los ejes x, y, z. Éstas toman como argumento una función del tiempo $\mu(t)$, que puede ser alguna de las coordenadas generalizadas de las cadenas cinemáticas descritas en 3.3. En general, los vectores se consideran columna, por lo que las matrices de rotación los premultiplican. Si bien podría usarse una única matriz de rotación alrededor de un eje conveniente, el uso señalado es generalmente más práctico.

$$\begin{aligned}
\mathbf{R}_x(\mu(t)) &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\mu(t)) & -\sin(\mu(t)) \\ 0 & \sin(\mu(t)) & \cos(\mu(t)) \end{bmatrix} \\
\mathbf{R}_y(\mu(t)) &= \begin{bmatrix} \cos(\mu(t)) & 0 & \sin(\mu(t)) \\ 0 & 1 & 0 \\ -\sin(\mu(t)) & 0 & \cos(\mu(t)) \end{bmatrix} \\
\mathbf{R}_z(\mu(t)) &= \begin{bmatrix} \cos(\mu(t)) & -\sin(\mu(t)) & 0 \\ \sin(\mu(t)) & \cos(\mu(t)) & 0 \\ 0 & 0 & 1 \end{bmatrix}
\end{aligned} \tag{3.18}$$

   Cada rotación de cada grado de libertad de interés, de cada cuerpo del sistema, puede entenderse como una transformación lineal entre bases de espacios vectoriales, que preserva ángulos, distancias y orientaciones (no hay reflexiones). Existe una base vectorial para el sistema coordenado original acoplado a la tierra, que permanece constante e igual a $\mathbf{I}$, y un origen, que de la misma forma es igual a $\mathbf{O}$. Cada rotación "genera" una nueva base vectorial, que le distingue con respecto a la base "previa" a la aplicación de la rotación. El conjunto de rotaciones que, a lo largo de una cadena cinemática que parte desde el sistema origen acoplado a la tierra, define la orientación de cada cuerpo, genera una base vectorial asociada al cuerpo, contenida en la matriz de rotación pertinente. En cada cuerpo, puede tomarse algún punto, como el centro de masa del cuerpo, o el centro de giro de los actuadores, y usar ese punto junto con la matriz de rotación asociada, para definir un sistema de referencia acoplado al cuerpo. Este sistema de referencia acoplado se mueve

con el cuerpo, permaneciendo invariante en relación al mismo, y en conjunto con parámetros de masa e inercia, para propósitos de modelado, representa al cuerpo. En cadenas cinemáticas que involucran varias rotaciones, el sistema de referencia acoplado al cuerpo final de cada subcadena es resultado de varias "generaciones" de tales sistemas de referencia, partiendo desde el sistema origen. Esto es fácil de tratar cuando todas las cadenas cinemáticas son abiertas. En el presente trabajo, no se consideran cadenas cinemáticas cerradas.

Las matrices de rotación definidas en 3.18 sirven para transformar vectores "dados" en términos de la base del sistema "posterior" (el sistema rotado), a vectores en términos de la base del sistema "anterior" (previo a la rotación). De manera equivalente, pueden verse como operaciones que "tomam" un vector en el sistema de referencia del propio vector, y lo rotan sin cambiar de sistema. La primera forma de entender las matrices de rotación, como cambios de base, es en general más útil.

Se especifican las matrices de rotación para todos los grados de libertad rotacionales del sistema, según 3.3:

$$\begin{aligned}
\mathbf{R}_{esf_1} &= \mathbf{R}_z(\psi_{esf}(t)) \\
\mathbf{R}_{esf_2} &= \mathbf{R}_y(\theta_{esf}(t)) \\
\mathbf{R}_{esf_3} &= \mathbf{R}_x(\phi_{esf}(t)) \\
\mathbf{R}_{pend_1} &= \mathbf{R}_x(\beta_{pend}(t)) \\
\mathbf{R}_{pend_2} &= \mathbf{R}_y(\alpha_{pend}(t)) \\
\mathbf{R}_{vol_1 1} &= \mathbf{R}_y(\varphi_{vol_1}(t)) \\
\mathbf{R}_{vol_1 2} &= \mathbf{R}_z(\gamma_{vol_1}(t)) \\
\mathbf{R}_{vol_2 1} &= \mathbf{R}_y(\varphi_{vol_2}(t)) \\
\mathbf{R}_{vol_2 2} &= \mathbf{R}_z(\gamma_{vol_2}(t))
\end{aligned} \tag{3.19}$$

2. *Hallar las matrices de rotación del sistema inercial al cuerpo.*

   Para cada cuerpo (la esfera, el péndulo y los dos volantes), se determina la composición de matrices de rotación que expresa la transformación entre la base del sistema (de referencia) origen acoplado a la tierra, y la base del sistema acoplado al cuerpo, de forma que vectores dados en la base del sistema

del cuerpo se expresen en la base del sistema origen. Como las transformaciones son sólo cambios entre bases ortonormales, se preservan magnitudes y ángulos. Esto se hace porque en realidad sólo importa conocer la orientación de los cuerpos, ignorando los desplazamientos entre los orígenes de los sistemas de referencia. Cada matriz $\mathbf{R}_i$ representa una transformación que toma vectores en términos del sistema rotado, y devuelve vectores en términos de la base del sistema sin rotar. La "dirección" de estas transformaciones parte del sistema "final" hacia el sistema "inicial", donde el sistema "raíz" es el sistema acoplado a la tierra.

Un vector dado en términos del sistema final debe premultiplicarse, primero, por la matriz que representa la última de las rotaciones, para obtener vectores dados en el sistema del cuerpo, en términos de la base del sistema previo. Esto podría entenderse como "deshacer" el cambio de base de la última rotación, pero preservando la información de ésta, para regresar a una base de menor nivel en la cadena cinemática, y así sucesivamente, hasta la primera de las rotaciones. De esta forma, se obtiene la transformación que toma vectores en términos del sistema de cada cuerpo y devuelve vectores en términos de la base del sistema origen acoplado a la tierra.

Para la esfera, el péndulo, y los dos volantes, las matrices finales son:

$$\begin{aligned}
\mathbf{R}_{esf} &= \mathbf{R}_{esf_1} \cdot \mathbf{R}_{esf_2} \cdot \mathbf{R}_{esf_3} \\
\mathbf{R}_{pend} &= \mathbf{R}_{esf} \cdot \mathbf{R}_{pend_1} \cdot \mathbf{R}_{pend_2} \\
\mathbf{R}_{vol_1} &= \mathbf{R}_{pend} \cdot \mathbf{R}_{vol_1 1} \cdot \mathbf{R}_{vol_1 2} \\
\mathbf{R}_{vol_2} &= \mathbf{R}_{pend} \cdot \mathbf{R}_{vol_2 1} \cdot \mathbf{R}_{vol_2 2}
\end{aligned} \tag{3.20}$$

La transpuesta de una matriz de rotación equivale a la matriz de rotación que representa la transformación "reversa", cambiando la dirección de la transformación entre las bases del sistemas rotado y del sistema sin rotar. Las transformaciones que toman vectores en términos de la base del sistema origen y devuelven vectores en términos de la base del sistema de cada cuerpo, son las transpuestas de las transformaciones anteriores.

3. *Obtener la velocidad angular del sistema origen en términos del sistema de de cada cuerpo.*

   Se ha escrito una función que toma la matriz de rotación total (dependiente de cada uno de los ángulos de rotación pertinentes, y cada uno de estos, una función del tiempo), genera el tensor de velocidad angular, definido en 3.11 y extrae el vector axial con las componentes deseadas de velocidad angular. Si la matriz de rotación es una que transforma vectores dados en términos del sistema rotado a vectores en términos del sistema acoplado a tierra, la velocidad angular obtenida será la del cuerpo, en términos del sistema origen.

$$\begin{aligned} \boldsymbol{\omega} &= AV(\mathbf{R}(t)) \\ AV(\mathbf{R}(t)) &= \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \end{aligned} \tag{3.21}$$

   Aplicando la función 3.21 a las matrices definidas en 3.20, se obtienen las velocidades angulares en términos del sistema origen acoplado a la tierra:

$$\begin{aligned} \boldsymbol{\omega}_{esf} &= AV(\mathbf{R}_{esf}) \\ \boldsymbol{\omega}_{pend} &= AV(\mathbf{R}_{pend}) \\ \boldsymbol{\omega}_{vol_1} &= AV(\mathbf{R}_{vol_1}) \\ \boldsymbol{\omega}_{vol_2} &= AV(\mathbf{R}_{vol_2}) \end{aligned} \tag{3.22}$$

   Estas velocidades angulares se entienden en relación con la base del sistema origen acoplado a la tierra, y sus asociaciones por medio de productos con otros elementos, como tensores de inercia o vectores de distancia, también deben entenderse en relación con el sistema origen. En una expresión de energía, todos los elementos deben compartir la misma "base" vectorial para que su producto tenga sentido físico. Como los tensores de inercia se obtienen en las bases de los propios cuerpos, es conveniente obtener las velocidades angulares en términos de esas mismas bases, y de esta forma no hacer modificaciones a las inercias. La expresión final es la misma, pues es independiente de la elección de la base. Para cada cuerpo, se obtiene

la velocidad angular del sistema acoplado a tierra, en términos del sistema acoplado al cuerpo, aplicando 3.13:

$$\begin{aligned}\boldsymbol{\Omega}_{esf} &= -AV(\mathbf{R}_{esf}{}^{T}) \\ \boldsymbol{\Omega}_{pend} &= -AV(\mathbf{R}_{pend}{}^{T}) \\ \boldsymbol{\Omega}_{vol_1} &= -AV(\mathbf{R}_{vol_1}{}^{T}) \\ \boldsymbol{\Omega}_{vol_2} &= -AV(\mathbf{R}_{vol_2}{}^{T})\end{aligned} \tag{3.23}$$

4. *Expresar todas las energías cinéticas rotacionales de los cuerpos.*

Se consideran tensores de inercia diagonales para todos los cuerpos, pues se asume que la esfera, el péndulo y los volantes son simétricos con respecto a un plano, por lo que los elementos fuera de la diagonal principal de la matriz de inercia son nulos. Se desprecian las contribuciones inerciales de algunos de los elementos presentes en el prototipo físico del robot, como las cajas de los volantes de inercia y se asume que las contribuciones de los elementos electrónicos pueden agruparse como una placa de material de masa equivalente y espesor tal que los cubra por completo.

$$\begin{aligned}\mathbf{I}_{esf} &= \begin{bmatrix} I_{xxE} & 0 & 0 \\ 0 & I_{yyE} & 0 \\ 0 & 0 & I_{zzE} \end{bmatrix} \\ \mathbf{I}_{pend} &= \begin{bmatrix} I_{xxP} & 0 & 0 \\ 0 & I_{yyP} & 0 \\ 0 & 0 & I_{zzP} \end{bmatrix} \\ \mathbf{I}_{vol} &= \begin{bmatrix} I_{xxV} & 0 & 0 \\ 0 & I_{yyV} & 0 \\ 0 & 0 & I_{zzV} \end{bmatrix}\end{aligned} \tag{3.24}$$

Se obtienen las energías cinéticas rotacionales de los cuerpos, usando la expresión 3.3, en términos de la base de cada cuerpo, para evitar expresar los tensores de inercia en términos de la base del sistema origen acoplado a la tierra.

$$\begin{aligned} E_{cresf} &= \frac{1}{2}{\boldsymbol{\Omega}_{esf}}^T \cdot \mathbf{I}_{esf} \cdot \boldsymbol{\Omega}_{esf} \\ E_{crpend} &= \frac{1}{2}{\boldsymbol{\Omega}_{pend}}^T \cdot \mathbf{I}_{pend} \cdot \boldsymbol{\Omega}_{pend} \\ E_{crvol_1} &= \frac{1}{2}{\boldsymbol{\Omega}_{vol_1}}^T \cdot \mathbf{I}_{vol} \cdot \boldsymbol{\Omega}_{vol_1} \\ E_{crvol_2} &= \frac{1}{2}{\boldsymbol{\Omega}_{vol_2}}^T \cdot \mathbf{I}_{vol} \cdot \boldsymbol{\Omega}_{vol_2} \end{aligned} \tag{3.25}$$

5. *Hallar las posiciones de centros de masa y velocidades lineales de los cuerpos.*

   En este caso, la derivación de la expresión depende de cada cuerpo, y de las formas en que las rotaciones y desplazamientos se encadenan entre sí, desde el origen del sistema acoplado a la tierra, hasta el centro de masa del cuerpo. Una expresión general de la posición del centro de masa de cada cuerpo con desplazamientos $\mathbf{vec}_i$ y rotaciones $\mathbf{rot}_j$ tiene la forma:

$$\mathbf{r} = \mathbf{vec}_0 + \mathbf{rot}_1 \cdot \mathbf{vec}_1 + \mathbf{rot}_2 \cdot \mathbf{vec}_2 + \cdots \tag{3.26}$$

   El centro geométrico de la esfera, de radio r_{esf}, se expresa con el vector:

$$\mathbf{r}_{esf} = \begin{bmatrix} x \\ y \\ r_{esf} \end{bmatrix} \tag{3.27}$$

   En el modelado, la esfera, el péndulo, cada volante y el sistema total se entienden como entidades distintas. La esfera es la carcasa del robot, y aunque los volantes se acoplan al péndulo, el péndulo se considera independiente de estos en masa e inercia. El centro de masa de la esfera se refiere al centro de masa del robot sin péndulo y sin volantes. El centro de masa del péndulo se refiere al centro de masa del robot sin la esfera y sin los volantes. El centro de masa de cada volante se refiere, en cada caso, al del sistema sin otras cosas más que el volante.

   Para el centro de masa de la esfera en reposo, puede existir algún desplazamiento sobre z con respecto al centro geométrico, de magnitud $offset_{esf}$. Si este desplazamiento no fuera sobre z, el sistema no podría estar en reposo:

$$\mathbf{r}_{esf_{cm}} = \mathbf{r}_{esf} + \mathbf{R}_{esf} \cdot \begin{bmatrix} 0 \\ 0 \\ offset_{esf} \end{bmatrix} \tag{3.28}$$

Uno de los extremos del péndulo coincide con el centro geométrico de la esfera, y la distancia de tal punto al centro de masa del péndulo corresponde al parámetro r_{pend}, que en reposo está orientado hacia "abajo".

$$\mathbf{r}_{pend} = \mathbf{r}_{esf} + \mathbf{R}_{pend} \cdot \begin{bmatrix} 0 \\ 0 \\ -r_{pend} \end{bmatrix} \tag{3.29}$$

Y para cada uno de los volantes, por simplicidad, se asume que la posición del centro de masa es independiente de las últimas dos rotaciones de los volantes ($\mathbf{R}_{vol_i\,j}$). Esto es, el centro de masa de los volantes coincide con el eje de giro de ambas rotaciones. Esto no es necesariamente cierto en el prototipo físico, pero las desviaciones deben ser pequeñas, porque generarían vibraciones. Los volantes están dispuestos de forma simétrica, en torno al péndulo, de manera que la línea que une sus centros de masa es ortogonal a la línea entre el centro geométrico de la esfera y el centro de masa del péndulo, a manera de una 'T'. Se consideran desplazamientos desde el centro geométrico de la esfera, con magnitud y_{vol} y z_{vol}, orientados de manera conveniente en la posición de reposo:

$$\begin{aligned} \mathbf{r}_{vol_1} &= \mathbf{r}_{esf} + \mathbf{R}_{pend} \cdot \begin{bmatrix} 0 \\ -y_{vol} \\ -z_{vol} \end{bmatrix} \\ \mathbf{r}_{vol_2} &= \mathbf{r}_{esf} + \mathbf{R}_{pend} \cdot \begin{bmatrix} 0 \\ y_{vol} \\ -z_{vol} \end{bmatrix} \end{aligned} \tag{3.30}$$

Las distancias a cada cuerpo se muestran en la figura 3.3. Se muestra también la disposición simplificada de los cuerpos del sistema, usada para propósitos de modelado.

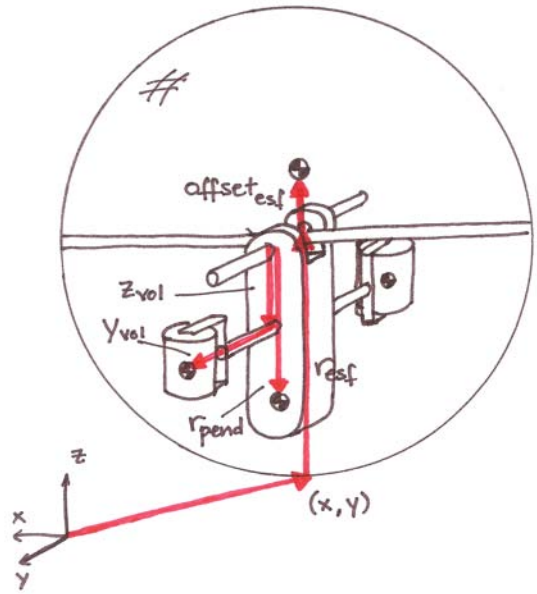


Figura 3.3 Distancias a cada uno de los cuerpos del sistema del robot.

Las velocidades lineales de los centros de masa pueden hallarse derivando sus posiciones con respecto al tiempo:

$$\begin{aligned} \mathbf{v}_{esf} &= \frac{d\mathbf{r}_{esf}}{dt} \\ \mathbf{v}_{pend} &= \frac{d\mathbf{r}_{pend}}{dt} \\ \mathbf{v}_{vol_1} &= \frac{d\mathbf{r}_{vol_1}}{dt} \\ \mathbf{v}_{vol_2} &= \frac{d\mathbf{r}_{vol_2}}{dt} \end{aligned} \tag{3.31}$$

6. *Expresar todas las energías cinéticas lineales de los cuerpos.*

Cada cuerpo tiene una masa especificada como:

Tabla 3.4 Masas de los cuerpos.

Subsistema	Masa
Esfera	m_{esf}
Péndulo	m_{pend}
Volante 1	m_{vol}
Volante 2	m_{vol}

Se obtienen las energías cinéticas traslacionales asociadas, usando la expresión 3.2. Se usa el producto punto entre los vectores:

$$\begin{aligned} E_{ctesf} &= \frac{1}{2} m_{esf} \mathbf{v}_{esf} \cdot \mathbf{v}_{esf} \\ E_{ctpend} &= \frac{1}{2} m_{pend} \mathbf{v}_{pend} \cdot \mathbf{v}_{pend} \\ E_{ctvol_1} &= \frac{1}{2} m_{vol} \mathbf{v}_{vol_1} \cdot \mathbf{v}_{vol_1} \\ E_{ctvol_2} &= \frac{1}{2} m_{vol} \mathbf{v}_{vol_2} \cdot \mathbf{v}_{vol_2} \end{aligned} \tag{3.32}$$

7. *Expresar todas las energías potenciales de los cuerpos.*

   Se considera un vector gravitatorio $\mathbf{g}$:

$$\mathbf{g} = \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \tag{3.33}$$

   Las energías potenciales son cero para cuerpos cuyo centro de masa se encuentra sobre el plano coincidente con el terreno (con $z = 0$). Si se usan los vectores de posición r_i definidos en el punto 5, estas energías pueden expresarse como:

$$\begin{aligned} E_{p_{esf}} &= m_{esf} \mathbf{g} \cdot \mathbf{r}_{esf} \\ E_{p_{pend}} &= m_{pend} \mathbf{g} \cdot \mathbf{r}_{pend} \\ E_{p_{vol_1}} &= m_{vol} \mathbf{g} \cdot \mathbf{r}_{vol_1} \\ E_{p_{vol_2}} &= m_{vol} \mathbf{g} \cdot \mathbf{r}_{vol_2} \end{aligned} \tag{3.34}$$

8. *Obtener el lagrangiano del sistema, y aplicar la parte izquierda de la ecuación de Euler-Lagrange sobre cada coordenada generalizada.*

   Recolectando las expresiones generadas hasta el momento:

$$\begin{gathered} T = E_{cresf} + E_{crpend} + E_{crvol_1} + E_{crvol_2} + E_{ctesf} + E_{ctpend} + E_{ctvol_1} + E_{ctvol_2} \\ V = E_{p_{esf}} + E_{p_{pend}} + E_{p_{vol_1}} + E_{p_{vol_2}} \\ L = T - V \end{gathered} \tag{3.35}$$

Se aplica la parte izquierda de la ecuación de Euler-Lagrange, sobre cada coordenada generalizada:

$$
\begin{aligned}
\text{E-L izq. 1:} = &\quad \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}}\right) - \frac{\partial L}{\partial x} \\
\text{E-L izq. 2:} = &\quad \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{y}}\right) - \frac{\partial L}{\partial y} \\
\text{E-L izq. 3:} = &\quad \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\psi}_{esf}}\right) - \frac{\partial L}{\partial \psi_{esf}} \\
\text{E-L izq. 4:} = &\quad \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_{esf}}\right) - \frac{\partial L}{\partial \theta_{esf}} \\
\text{E-L izq. 5:} = &\quad \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\phi}_{esf}}\right) - \frac{\partial L}{\partial \phi_{esf}} \\
\text{E-L izq. 6:} = &\quad \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\beta}_{pend}}\right) - \frac{\partial L}{\partial \beta_{pend}} \\
\text{E-L izq. 7:} = &\quad \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\alpha}_{pend}}\right) - \frac{\partial L}{\partial \alpha_{pend}} \\
\text{E-L izq. 8:} = &\quad \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\varphi}_{vol_1}}\right) - \frac{\partial L}{\partial \varphi_{vol_1}} \\
\text{E-L izq. 9:} = &\quad \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\gamma}_{vol_1}}\right) - \frac{\partial L}{\partial \gamma_{vol_1}} \\
\text{E-L izq. 10:} = &\quad \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\varphi}_{vol_2}}\right) - \frac{\partial L}{\partial \varphi_{vol_2}} \\
\text{E-L izq. 11:} = &\quad \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\gamma}_{vol_2}}\right) - \frac{\partial L}{\partial \gamma_{vol_2}}
\end{aligned}
\tag{3.36}
$$

9. *Expresar las ecuaciones de restricción del sistema y hallar todos los coeficientes a_{ij}.*

   Las restricciones al movimiento del sistema, como la condición de no deslizamiento, se expresan como funciones de las coordenadas generalizadas, estas funciones deben ser iguales a cero en todo momento de la evolución del sistema. En el caso de la condición de no deslizamiento, se generan dos ecuaciones escalares, una relacionada con x, y otra con y, que resultan de expandir la expresión vectorial 3.10:

$$\begin{aligned}
&\text{Restricción 1:} \quad \dot{x} - r_{esf}[\cos(\psi_{esf})\dot{\theta}_{esf} + \cos(\theta_{esf})\sin(\psi_{esf})\dot{\phi}_{esf}] = 0 \\
&\text{Restricción 2:} \quad \dot{y} + r_{esf}[-\sin(\psi_{esf})\dot{\theta}_{esf} + \cos(\theta_{esf})\cos(\psi_{esf})\dot{\phi}_{esf}] = 0
\end{aligned} \tag{3.37}$$

Donde cada coeficiente a_{ij} se calcula en base a 3.8. Esto es:

$$\begin{aligned}
a_{11} &= \frac{\partial(restriccion_1)}{\partial x} & & \\
a_{12} &= \frac{\partial(restriccion_2)}{\partial x} & a_{71} &= \frac{\partial(restriccion_1)}{\partial \alpha_{pend}} \\
a_{21} &= \frac{\partial(restriccion_1)}{\partial y} & a_{72} &= \frac{\partial(restriccion_2)}{\partial \alpha_{pend}} \\
a_{22} &= \frac{\partial(restriccion_2)}{\partial y} & a_{81} &= \frac{\partial(restriccion_1)}{\partial \varphi_{vol_1}} \\
a_{31} &= \frac{\partial(restriccion_1)}{\partial \psi_{esf}} & a_{82} &= \frac{\partial(restriccion_2)}{\partial \varphi_{vol_1}} \\
a_{32} &= \frac{\partial(restriccion_2)}{\partial \psi_{esf}} & a_{91} &= \frac{\partial(restriccion_1)}{\partial \gamma_{vol_1}} \\
a_{41} &= \frac{\partial(restriccion_1)}{\partial \theta_{esf}} & a_{92} &= \frac{\partial(restriccion_2)}{\partial \gamma_{vol_1}} \\
a_{42} &= \frac{\partial(restriccion_2)}{\partial \theta_{esf}} & a_{101} &= \frac{\partial(restriccion_1)}{\partial \varphi_{vol_2}} \\
a_{51} &= \frac{\partial(restriccion_1)}{\partial \phi_{esf}} & a_{102} &= \frac{\partial(restriccion_2)}{\partial \varphi_{vol_2}} \\
a_{52} &= \frac{\partial(restriccion_2)}{\partial \phi_{esf}} & a_{111} &= \frac{\partial(restriccion_1)}{\partial \gamma_{vol_2}} \\
a_{61} &= \frac{\partial(restriccion_1)}{\partial \beta_{pend}} & a_{112} &= \frac{\partial(restriccion_2)}{\partial \gamma_{vol_2}} \\
a_{62} &= \frac{\partial(restriccion_2)}{\partial \beta_{pend}} & &
\end{aligned} \tag{3.38}$$

10. *Establecer las fuerzas generalizadas para la parte derecha de las ecuaciones de Euler-Lagrange, y sustituir las a_{ij} de las ecuaciones de restricción.*

    Para completar las ecuaciones parciales de Euler-Lagrange generadas en 3.36, se adicionan las fuerzas aplicables sobre cada coordenada generalizada.

Como se consideran fricciones nulas, sólo es necesario expandir los coeficientes a_{ij}, y agregar los pares de entrada τ_{q_i} en las coordenadas generalizadas pertinentes, que se tratan como si fueran variables adicionales.

En particular, se consideran los pares de entrada en los cuerpos y coordenadas generalizadas siguientes:

Tabla 3.5 Pares de entrada

CG	Cuerpo	Par de entrada
β_{pend}	Péndulo	$\tau_{\beta_{pend}}$
α_{pend}	Péndulo	$\tau_{\alpha_{pend}}$
φ_{vol_1}	Volante 1	$\tau_{\varphi_{vol_1}}$
φ_{vol_2}	Volante 2	$\tau_{\varphi_{vol_2}}$
γ_{vol_1}	Volante 1	$\tau_{\gamma_{vol_1}}$
γ_{vol_2}	Volante 2	$\tau_{\gamma_{vol_2}}$

Por lo que las partes derechas de las ecuaciones de Euler-Lagrange, para todas las coordenadas generalizadas, quedan como:

$$\begin{aligned}
\text{E-L der. 1:} &= a_{11}\lambda_1 + a_{12}\lambda_2 \\
\text{E-L der. 2:} &= a_{21}\lambda_1 + a_{22}\lambda_2 \\
\text{E-L der. 3:} &= a_{31}\lambda_1 + a_{32}\lambda_2 \\
\text{E-L der. 4:} &= a_{41}\lambda_1 + a_{42}\lambda_2 \\
\text{E-L der. 5:} &= a_{51}\lambda_1 + a_{52}\lambda_2 \\
\text{E-L der. 6:} &= \tau_{\beta_{pend}} + a_{61}\lambda_1 + a_{62}\lambda_2 \\
\text{E-L der. 7:} &= \tau_{\alpha_{pend}} + a_{71}\lambda_1 + a_{72}\lambda_2 \\
\text{E-L der. 8:} &= \tau_{\varphi_{vol_1}} + a_{81}\lambda_1 + a_{82}\lambda_2 \\
\text{E-L der. 9:} &= \tau_{\gamma_{vol_1}} + a_{91}\lambda_1 + a_{92}\lambda_2 \\
\text{E-L der. 10:} &= \tau_{\varphi_{vol_2}} + a_{10\,1}\lambda_1 + a_{10\,2}\lambda_2 \\
\text{E-L der. 11:} &= \tau_{\gamma_{vol_2}} + a_{11\,1}\lambda_1 + a_{11\,2}\lambda_2
\end{aligned} \tag{3.39}$$

donde cada a_{ij} viene de la expansión de las expresiones en 3.38.

Como hay seis pares de entrada, que se tratan como variables adicionales a las coordenadas generalizadas, se necesitan seis ecuaciones de entrada.

11. *Recolectar las expresiones anteriores, conformar las ecuaciones dinámicas del sistema, y anexar las ecuaciones de restricción y las ecuaciones de entrada.*

Se recolectan las expresiones simbólicas parciales generadas en pasos anteriores, y al desarrollarlas, se construyen las ecuaciones diferenciales que luego deberán ser resueltas numéricamente. Esto es:

$$
\begin{aligned}
&\text{E-L 1:} \quad \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}}\right) - \frac{\partial L}{\partial x} = a_{11}\lambda_1 + a_{12}\lambda_2 \\
&\text{E-L 2:} \quad \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{y}}\right) - \frac{\partial L}{\partial y} = a_{21}\lambda_1 + a_{22}\lambda_2 \\
&\text{E-L 3:} \quad \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\psi}_{esf}}\right) - \frac{\partial L}{\partial \psi_{esf}} = a_{31}\lambda_1 + a_{32}\lambda_2 \\
&\text{E-L 4:} \quad \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_{esf}}\right) - \frac{\partial L}{\partial \theta_{esf}} = a_{41}\lambda_1 + a_{42}\lambda_2 \\
&\text{E-L 5:} \quad \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\phi}_{esf}}\right) - \frac{\partial L}{\partial \phi_{esf}} = a_{51}\lambda_1 + a_{52}\lambda_2 \\
&\text{E-L 6:} \quad \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\beta}_{pend}}\right) - \frac{\partial L}{\partial \beta_{pend}} = \tau_{\beta_{pend}} + a_{61}\lambda_1 + a_{62}\lambda_2 \\
&\text{E-L 7:} \quad \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\alpha}_{pend}}\right) - \frac{\partial L}{\partial \alpha_{pend}} = \tau_{\alpha_{pend}} + a_{71}\lambda_1 + a_{72}\lambda_2 \\
&\text{E-L 8:} \quad \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\varphi}_{vol_1}}\right) - \frac{\partial L}{\partial \varphi_{vol_1}} = \tau_{\varphi_{vol_1}} + a_{81}\lambda_1 + a_{82}\lambda_2 \\
&\text{E-L 9:} \quad \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\gamma}_{vol_1}}\right) - \frac{\partial L}{\partial \gamma_{vol_1}} = \tau_{\gamma_{vol_1}} + a_{91}\lambda_1 + a_{92}\lambda_2 \\
&\text{E-L 10:} \quad \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\varphi}_{vol_2}}\right) - \frac{\partial L}{\partial \varphi_{vol_2}} = \tau_{\varphi_{vol_2}} + a_{10\,1}\lambda_1 + a_{10\,2}\lambda_2 \\
&\text{E-L 11:} \quad \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\gamma}_{vol_2}}\right) - \frac{\partial L}{\partial \gamma_{vol_2}} = \tau_{\gamma_{vol_2}} + a_{11\,1}\lambda_1 + a_{11\,2}\lambda_2
\end{aligned}
\tag{3.40}
$$

De requerirse entradas de voltaje, para propósitos de control, se podrían modelar los motores como sistemas de segundo orden, y agregar sus masas e inercias como cuerpos adicionales. En el modelo, no se consideran las inercias rotaciones de los ejes de los motores, o su engranes reductores.

$$\begin{aligned} \tau_{\beta_{pend}} &= k_{1mot}(V_{\beta_{pend}} - k_{2mot}\,\dot{\beta}_{pend}) \\ \tau_{\alpha pend} &= k_{1mot}(V_{\alpha_{pend}} - k_{2mot}\,\dot{\alpha}_{pend}) \end{aligned} \tag{3.41}$$

donde k_{1mot} y k_{2mot} son constantes del motor, que pueden obtenerse de pruebas, o de hojas de especificaciones, y $V_{\beta_{pend}}$ y $V_{\alpha_{pend}}$ son los voltajes de entrada.

Para resolver numéricamente el sistema, deben considerarse las once ecuaciones de Euler-Lagrange generadas, las dos ecuaciones de restricción, y las seis ecuaciones de entrada. Las ecuaciones de entrada se relacionan directa o indirectamente con las fuerzas de entrada. Con motores modelados, las ecuaciones de entrada podrían ser, por ejemplo:

$$\begin{aligned} &\text{Entrada 1:} \quad V_{\beta_{pend}} = f_1 \\ &\text{Entrada 2:} \quad V_{\alpha_{pend}} = f_2 \\ &\text{Entrada 3:} \quad \dot{\varphi}_{vol_1} = f_3 \\ &\text{Entrada 4:} \quad \dot{\gamma}_{vol_1} = f_4 \\ &\text{Entrada 5:} \quad \dot{\varphi}_{vol_2} = f_5 \\ &\text{Entrada 6:} \quad \dot{\gamma}_{vol_2} = f_6 \end{aligned} \tag{3.42}$$

La elección de ecuaciones de entrada depende de los propósitos de la simulación. Si no se modelan los motores, se podrían reemplazar las dos primeras entradas por restricciones sobre las velocidades de los ángulos del péndulo, por ejemplo:

$$\begin{aligned} &\text{Entrada 1:} \quad \dot{\beta}_{pend} = f_1 \\ &\text{Entrada 2:} \quad \dot{\alpha}_{pend} = f_2 \end{aligned} \tag{3.43}$$

12. *Establecer las condiciones iniciales para las coordenadas generalizadas y sus derivadas, y sustituir todos los parámetros (masas, inercias, distancias) por sus valores numéricos.*

    En la mayoría de los escenarios de prueba, se asume que las condiciones iniciales de ángulos, distancias, velocidades y aceleraciones son cero. La disposición de los desplazamientos es tal que, a condiciones iniciales nulas y tiempo cero, la esfera se encuentra en reposo, con el péndulo orientado hacia abajo.

    Aunque se realizaron pruebas simuladas con una variedad de posibles parámetros, el modelo final utilizó los valores siguientes, obtenidos de los modelos en CAD de las piezas del robot:

Tabla 3.6 Valores de los parámetros del sistema.

Parámetro	Valor	Unidades	Parámetro	Valor	Unidades
I_{zzP}	0.2728	$\mathrm{kg \cdot m^2}$	k_{1mot}	2.5	$\mathrm{N \cdot m/V}$
I_{xxP}	0.1751	$\mathrm{kg \cdot m^2}$	k_{2mot}	0.5305	$\mathrm{V \cdot s/rad}$
I_{yyP}	0.3813	$\mathrm{kg \cdot m^2}$	m_{pend}	5.148	kg
I_{zzS}	0.2262	$\mathrm{kg \cdot m^2}$	m_{esf}	8.597	kg
I_{yyS}	0.1589	$\mathrm{kg \cdot m^2}$	m_{vol}	4.249	kg
I_{xxS}	0.1949	$\mathrm{kg \cdot m^2}$	$offset_{esf}$	0	m
I_{zzV}	0.003004	$\mathrm{kg \cdot m^2}$	r_{esf}	0.2495	m
I_{xxV}	0.006185	$\mathrm{kg \cdot m^2}$	r_{pend}	0.129	m
I_{yyV}	0.005963	$\mathrm{kg \cdot m^2}$	y_{vol}	0.1495	m
g	9.81	$\mathrm{m/s^2}$	z_{vol}	0.0522	m

13. *Resolver numéricamente el sistema generado.*

    En algún software apropiado, en el caso de éste proyecto, se usó el software comercial Mathematica, en su versión 10. Dadas las limitaciones de estos programas, y por la considerable longitud de las ecuaciones generadas con esta forma de modelado, la resolución puede tomar una larga cantidad de tiempo, y ser muy sensible a variaciones en las condiciones iniciales, o el paso de evaluación. Una evaluación a largo plazo del comportamiento del sistema, aunque útil, no necesariamente refleja su evolución verdadera. Se programó una representación gráfica tridimensional del modelo y las simulaciones, para observar más intuitivamente el movimiento simulado.

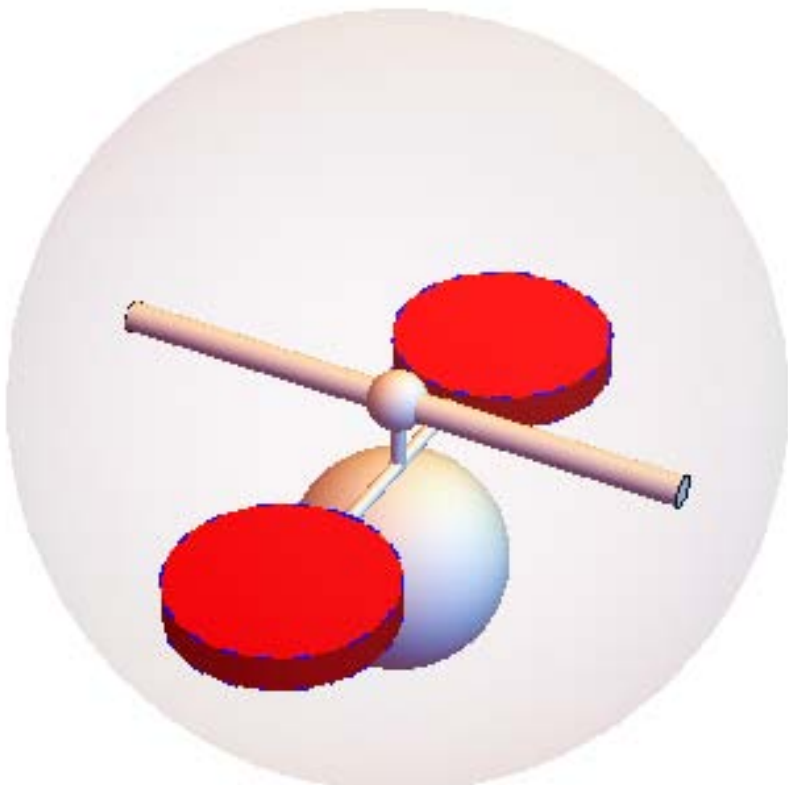


Figura 3.4 Modelo tridimensional en Mathematica.

3.5 Pruebas del modelo

Por supuesto, es necesario realizar algunas pruebas del modelo generado para verificar que el comportamiento simulado asemeje a una situación "real". Se somete el modelo a varios conjuntos de condiciones iniciales y ecuaciones de entrada para estudiar su comportamiento.

3.5.1 Oscilación del péndulo sin entrada

Se desea observar el movimiento de la esfera, influenciado por la presencia del péndulo, al intentar regresar al punto de equilibrio estable (con el péndulo orientado completamente hacia abajo). Se consideran ángulos iniciales para el péndulo de 1 rad en ambos casos (en las coordenadas generalizadas α_{pend} y β_{pend}):

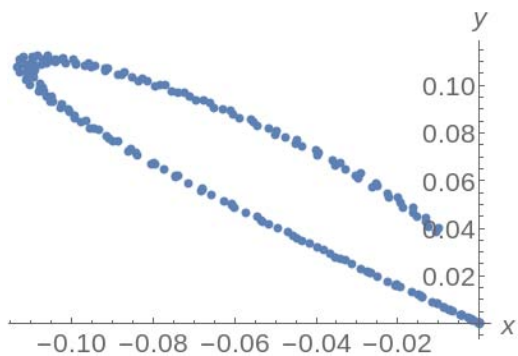
Tabla 3.7 Condiciones iniciales del sistema.

Variable	$f(t_0)$	Variables	$f(t_0)$
x	0	α_{pend}	1
$\dot{x}$	0	$\dot{\alpha}_{pend}$	0
y	0	$\beta pend$	1
$\dot{y}$	0	$\dot{\beta}_{pend}$	0
ψ_{esf}	0	φ_{vol_1}	0
$\dot{\psi}_{esf}$	0	$\dot{\varphi}_{vol_1}$	0
θ_{esf}	0	γ_{vol_1}	0
$\dot{\theta}_{esf}$	0	$\dot{\gamma}_{vol_1}$	0
ϕ_{esf}	0	φ_{vol_2}	0
$\dot{\phi}_{esf}$	0	$\dot{\varphi}_{vol_2}$	0
		γ_{vol_2}	0
		$\dot{\gamma}_{vol_2}$	0

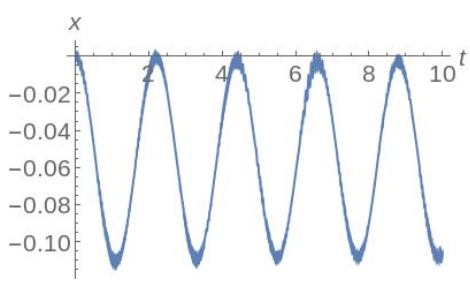
Se esperaría que, en ausencia de entradas al sistema (es decir, manteniendo los ángulos del péndulo y los volantes nulos), la esfera se mueva hacia adelante y atrás (oscilación del péndulo) sobre alguna linea oblicua a los ejes x e y, quizá con desviaciones del movimiento rectilíneo debido a que las matrices de inercia no son iguales en todos sus componentes. Para las ecuaciones de entrada, se considera que ninguno de los ángulos de rotación adicionales a los de la esfera experimenta cambios, lo que equivale a decir que sus velocidades son cero. El conjunto de ecuaciones de entrada es entonces:

$$\begin{aligned}
&\text{Entrada 1:} \quad \dot{\beta}_{pend} = 0 \\
&\text{Entrada 2:} \quad \dot{\alpha}_{pend} = 0 \\
&\text{Entrada 3:} \quad \dot{\varphi}_{vol_1} = 0 \\
&\text{Entrada 4:} \quad \dot{\gamma}_{vol_1} = 0 \\
&\text{Entrada 5:} \quad \dot{\varphi}_{vol_2} = 0 \\
&\text{Entrada 6:} \quad \dot{\gamma}_{vol_2} = 0
\end{aligned} \tag{3.44}$$

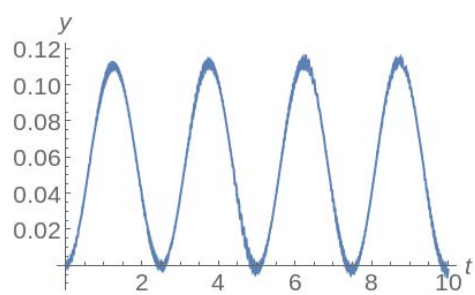
La trayectoria obtenida es razonablemente satisfactoria. Se considera una ventana de tiempo de 3 segundos (no se muestran más puntos porque las trayectorias posteriores son redundantes). El péndulo se mueve y regresa a un punto cercano al origen, aunque no exactamente al origen, presumiblemente por las asimetrías en las matrices de inercia.



(a) Trayectoria en x y y, 200 puntos en 2 s



(b) Trayectoria en x, en 10 s



(c) Trayectoria en y, en 10 s

Figura 3.5

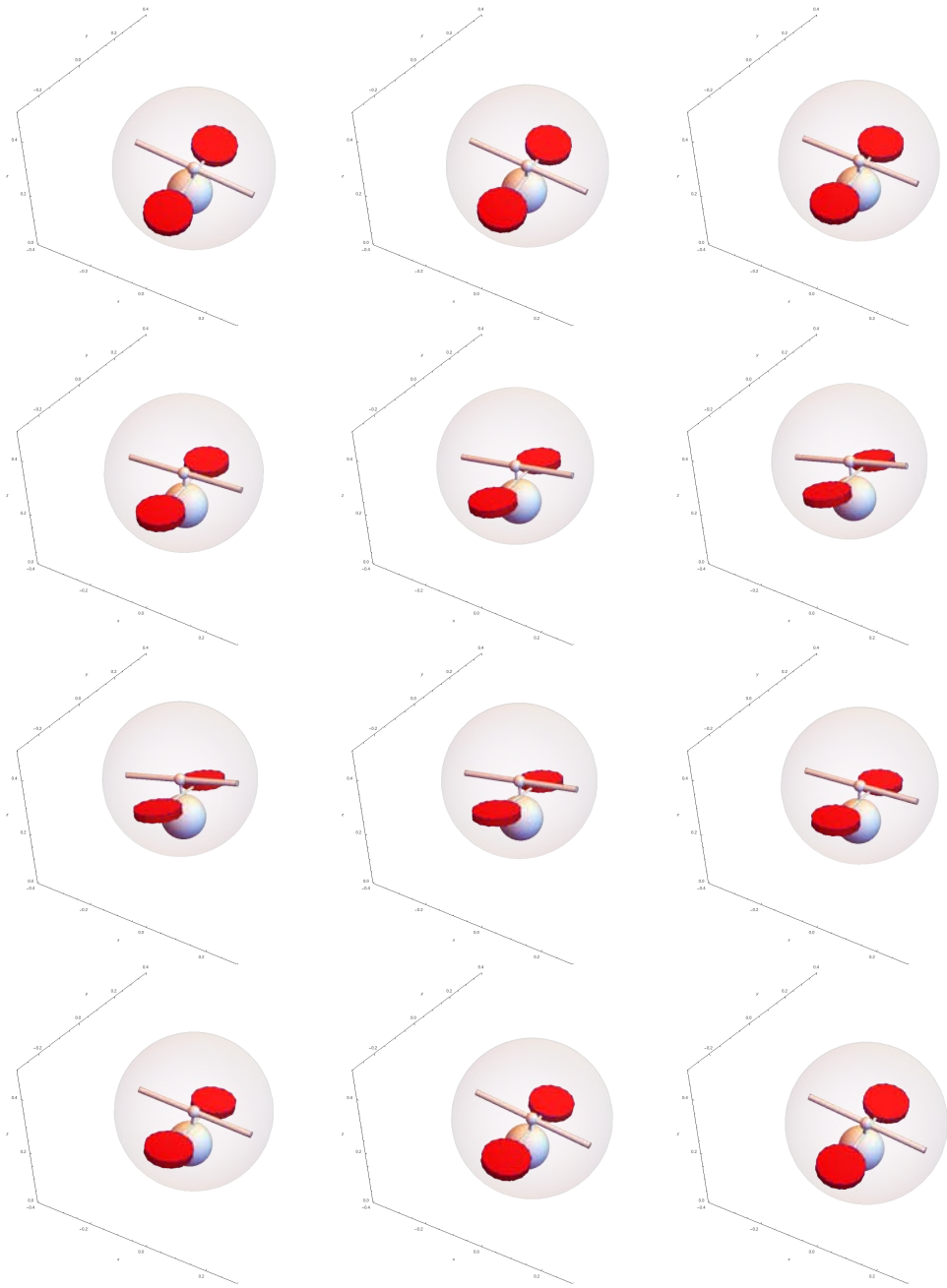


Figura 3.6 Animación del movimiento resultante, a intervalos de 0.2 s.

3.5.2 Péndulo con una entrada de velocidad

Se consideran ángulos iniciales nulos en todas las coordenadas generalizadas. Esta prueba es similar a la anterior, pero forzando a uno de los ángulos del péndulo a incrementarse a velocidad constante.

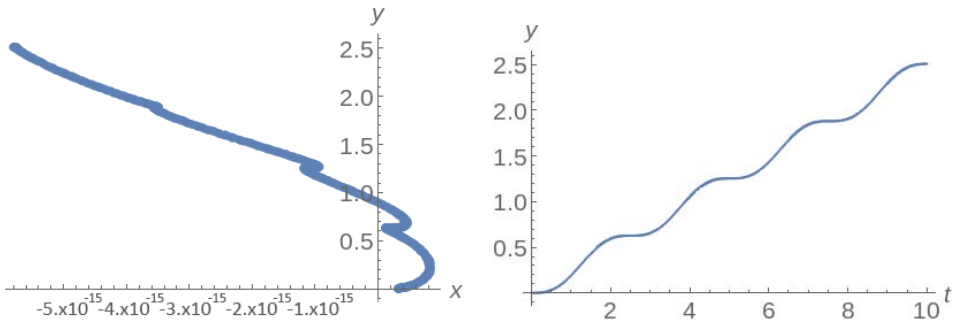
Tabla 3.8 Condiciones iniciales del sistema.

Variable	$f(t_0)$	Variables	$f(t_0)$
x	0	α_{pend}	0
$\dot{x}$	0	$\dot{\alpha}_{pend}$	0
y	0	$\beta pend$	0
$\dot{y}$	0	$\dot{\beta}_{pend}$	0
ψ_{esf}	0	φ_{vol_1}	0
$\dot{\psi}_{esf}$	0	$\dot{\varphi}_{vol_1}$	0
θ_{esf}	0	γ_{vol_1}	0
$\dot{\theta}_{esf}$	0	$\dot{\gamma}_{vol_1}$	0
ϕ_{esf}	0	φ_{vol_2}	0
$\dot{\phi}_{esf}$	0	$\dot{\varphi}_{vol_2}$	0
		γ_{vol_2}	0
		$\dot{\gamma}_{vol_2}$	0

Como condiciones de entrada, se considera una entrada de velocidad constante en uno de los ángulos del péndulo (β_{pend}), y no se usan los modelos de los motores. Se esperaría que, en ausencia de otra entrada al sistema, el péndulo se mueva en una dirección en línea recta (sobre el eje y), quizá con algún vaivén provocado por la oscilación sin mitigar del péndulo.

$$\begin{aligned} &\text{Entrada 1:} \quad \dot{\beta}_{pend} = 1 \\ &\text{Entrada 2:} \quad \dot{\alpha}_{pend} = 0 \\ &\text{Entrada 3:} \quad \dot{\varphi}_{vol_1} = 0 \\ &\text{Entrada 4:} \quad \dot{\gamma}_{vol_1} = 0 \\ &\text{Entrada 5:} \quad \dot{\varphi}_{vol_2} = 0 \\ &\text{Entrada 6:} \quad \dot{\gamma}_{vol_2} = 0 \end{aligned} \tag{3.45}$$

La trayectoria obtenida es razonablemente satisfactoria. El centro geométrico de la esfera se mueve sobre un solo eje (y) suavemente, con una desviación de la velocidad lineal uniforme a manera de vaivén, debido a la oscilación del péndulo.



(a) Trayectoria en x y y, en 10 s (b) Trayectoria en y, en 10 s

Figura 3.7

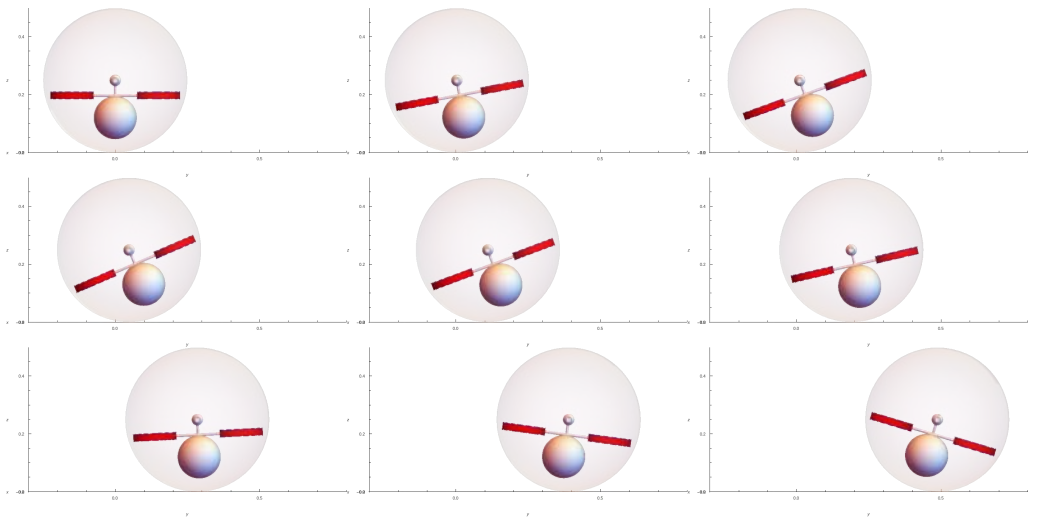


Figura 3.8 Animación del movimiento resultante, a intervalos de 0.2 s

3.5.3 Rotación de los volantes

Los volantes de inercia pueden usarse para intercambiar momento angular, y así generar un par de avance con magnitud suficiente para sobrepasar obstáculos menores, o escapar de depresiones del terreno. En terreno plano, sin obstáculos, mantener girando de forma continua las cajas de los volantes provocaría que la esfera avance en una dirección hasta que la dirección del par resultante cambie, y la esfera invierta su sentido de giro, y así sucesivamente. Se consideran ángulos y velocidades iniciales nulos.

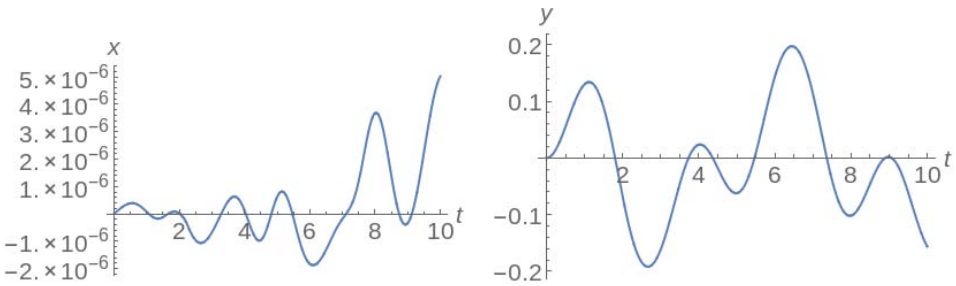
Tabla 3.9 Condiciones iniciales del sistema.

Variable	$f(t_0)$	Variables	$f(t_0)$
x	0	α_{pend}	0
$\dot{x}$	0	$\dot{\alpha}_{pend}$	0
y	0	$\beta pend$	0
$\dot{y}$	0	$\dot{\beta}_{pend}$	0
ψ_{esf}	0	φ_{vol_1}	0
$\dot{\psi_{esf}}$	0	$\dot{\varphi}_{vol_1}$	0
θ_{esf}	0	γ_{vol_1}	0
$\dot{\theta_{esf}}$	0	$\dot{\gamma}_{vol_1}$	0
ϕ_{esf}	0	φ_{vol_2}	0
$\dot{\phi_{esf}}$	0	$\dot{\varphi}_{vol_2}$	0
		γ_{vol_2}	0
		$\dot{\gamma}_{vol_2}$	0

Se establece solamente el giro de los volantes como condiciones de entrada no nulas. Los volantes giran a velocidades opuestas, para asegurar que en el estado inicial la suma de momentos angulares sea nula, y mitigar los efectos indeseados de la rotación de giroscopios. Se esperaría que la esfera se mueva en una sola dirección, en línea recta, en una especie de vaivén hacia adelante y atrás, a medida que el vector del par resultante producido por rotar los CMG cambia periódicamente de dirección.

$$\begin{aligned}
&\text{Entrada 1:} \quad \dot{\beta}_{pend} = 0 \\
&\text{Entrada 2:} \quad \dot{\alpha}_{pend} = 0 \\
&\text{Entrada 3:} \quad \dot{\varphi}_{vol_1} = 1 \\
&\text{Entrada 4:} \quad \dot{\gamma}_{vol_1} = 600 \\
&\text{Entrada 5:} \quad \dot{\varphi}_{vol_2} = -1 \\
&\text{Entrada 6:} \quad \dot{\gamma}_{vol_2} = -600
\end{aligned} \tag{3.46}$$

La trayectoria obtenida es razonablemente satisfactoria. El centro geométrico de la esfera se mueve sobre un solo eje (y) de manera relativamente suave, a manera de un vaivén caprichoso hacia adelante y atrás. Este vaivén es producido por el cambio de dirección del par resultante generado al rotar los volantes como CMG. Debido a este vaivén, el tiempo efectivo de uso continuo de los volantes es limitado, pues eventualmente el sistema se quedará sin momento angular que intercambiar, y habrá que regresar (lentamente) los volantes a la posición inicial.



(a) Trayectoria en x, 10 s (b) Trayectoria en y, en 10 s

Figura 3.9

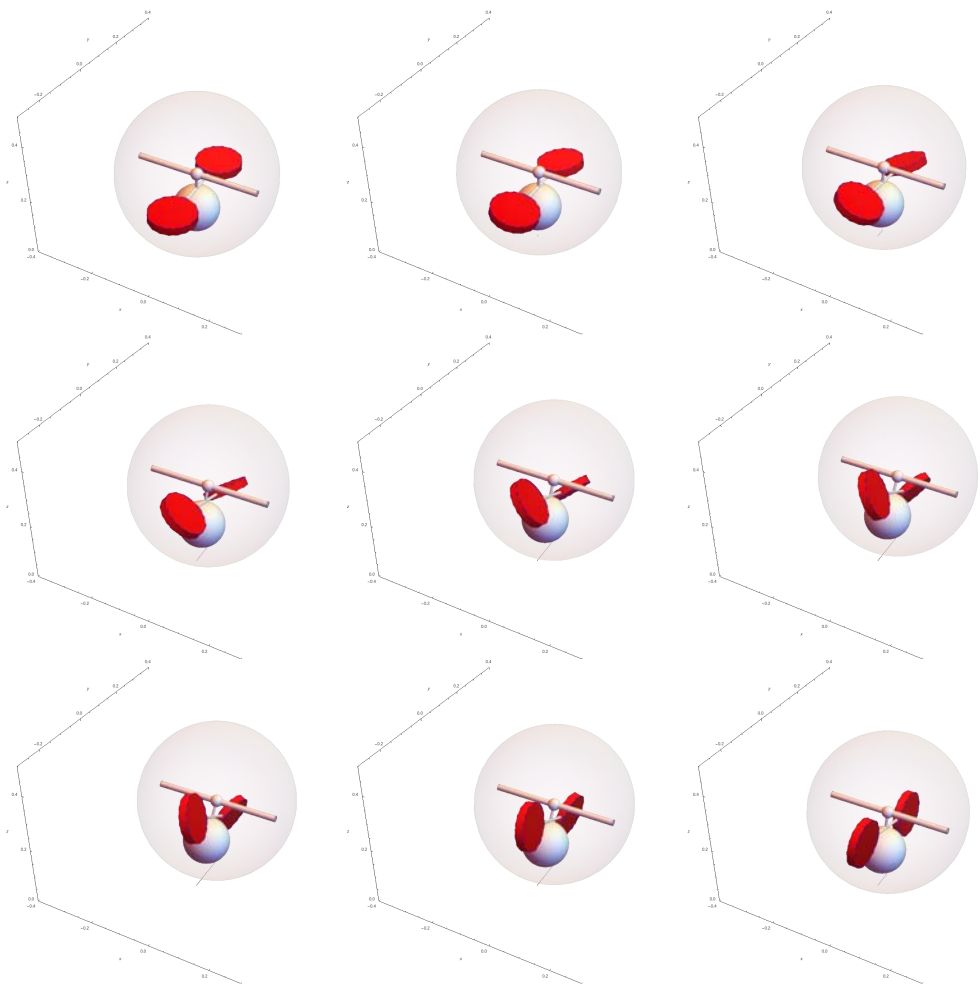


Figura 3.10 Animación del movimiento resultante, a intervalos de 0.2 s.

3.6 Uso del modelo para obtener parámetros de diseño

Varios de los aspectos de diseño estuvieron sujetos a restricciones de disponibilidad de componentes en el mercado, y a las necesidades de dimensionamiento consecuencia de tales restricciones. Sin embargo, un modelo más o menos fiel del comportamiento del sistema es útil, entre otras cosas, para determinar, previamente a la adquisición, los requerimientos de velocidad, par o potencia de los motores, y de esta manera poder obtener partes adecuadas.

En particular, es necesario dimensionar correctamente los motores que mueven el péndulo, el motor que mueve (simultáneamente) las cajas de los volantes, y los motores *brushless* que hacen girar a los volantes. Este dimensionamiento debe entenderse como una guía burda para las decisiones de adquisición, ya que no es realista esperar encontrar motores de las especificaciones exactas demandadas.

Para los motores del péndulo y el motor que mueve las cajas de los volantes, interesa estimar la capacidad de par generada por el motor para las condiciones de operación deseadas. Esto es, movimiento hacia adelante de la esfera a la velocidad especificada, y velocidad máxima de rotación de los motores *brushless*.

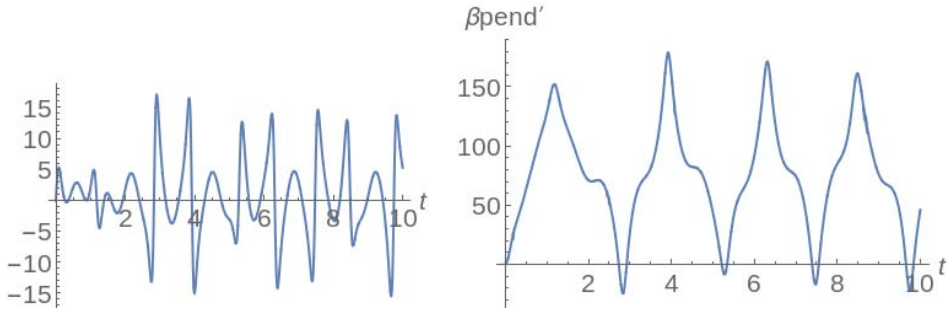
3.6.1 Dimensionamiento de los motores del péndulo

De manera arbitraria, se desea que el movimiento de la esfera en el plano permita un desplazamiento lineal a una velocidad máxima de 2 m/s, o 7.2 km/h.

El único cambio de importancia en el modelo del sistema se hace en las ecuaciones de entrada. En particular, para una velocidad en la coordenada y de 2 m/s, un movimiento nulo en la coordenada x, y con los volantes apagados, las ecuaciones de entrada son:

$$\begin{aligned}
&\text{Entrada 1:} \quad \dot{x} = 0 \\
&\text{Entrada 2:} \quad \dot{y} = 2 \\
&\text{Entrada 3:} \quad \dot{\varphi}_{vol_1} = 0 \\
&\text{Entrada 4:} \quad \dot{\gamma}_{vol_1} = 0 \\
&\text{Entrada 5:} \quad \dot{\varphi}_{vol_2} = 0 \\
&\text{Entrada 6:} \quad \dot{\gamma}_{vol_2} = 0
\end{aligned} \tag{3.47}$$

En el modelo simulado, el ángulo β_{pend} y la velocidad $\dot{\beta}_{pend}$ se comportan como:



(a) Par del péndulo en N·m, por 10 s (b) ω del péndulo en rpm, por 10 s

Figura 3.11

Interesa conocer la magnitud máxima del par a lo largo de la evolución del sistema, de forma más bien burda. Un valor pico de 15 N·m sugiere el máximo par del motor, a una velocidad de 150 rpm. Un motor con un par de 30 N·m a rotor bloqueado, y velocidad de 300 rpm sin carga, sería una buena elección. En el modelo real, dos motores contribuyen al movimiento del péndulo. No hay reducción entre el eje de un motor del péndulo y el engrane que acciona, pero los engranes en configuración cónica se "reparten" el trabajo de movimiento del péndulo, también sobre el ángulo α_{pend}. El efecto final es que el dimensionamiento del motor "ideal" de sólo un ángulo de avance β_{pend}, es también aplicable al de los dos motores reales.

3.6.2 Dimensionamiento de los motores de los volantes

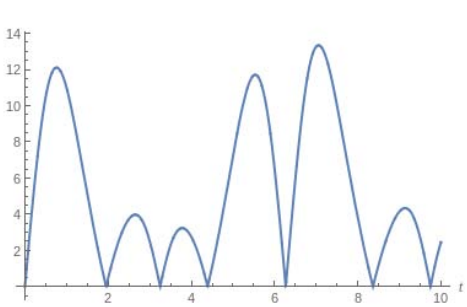
En este caso, interesa que el movimiento de la esfera en el plano sea tal que la esfera pueda "voltearse" sobre sí misma, únicamente rotando las cajas de los volantes. Pruebas de funcionamiento de motores *brushless* y los volantes sugieren una velocidad de rotación máxima de alrededor de 9000 rpm (cerca de 940 rad/s). Como los volantes se usan para que el robot esférico pueda pasar sobre obstáculos, interesa también que esta capacidad pueda sustentarse por un tiempo. Arbitrariamente, se escoge un tiempo deseado de 3 segundos, durante el cual pueda generarse continuamente el impulso para sobrepasar obstáculos.

La orientación inicial de los volantes es tal que su eje de rotación es paralelo al eje z del sistema inercial. Para anular los efectos indeseados de la rotación de giroscopios, ambos volantes están dispuestos de forma simétrica en torno al centro de la esfera, y giran a velocidades opuestas, de manera que las componentes indeseadas de par se cancelan, dejando como resultante un par que propulsa la esfera. Sin embargo, en este arreglo los volantes sólo pueden rotarse un cuarto de vuelta antes de que la dirección de dicha resultante cambie (en efecto, se invierta en signo), y el efecto se vuelva contraproducente. Esto puede resolverse si se permite recuperar, lentamente la orientación inicial de los volantes. Esto debe hacerse lentamente para aprovechar la fricción con el medio, y de tal forma evitar que la esfera se mueva, negando el avance realizado.

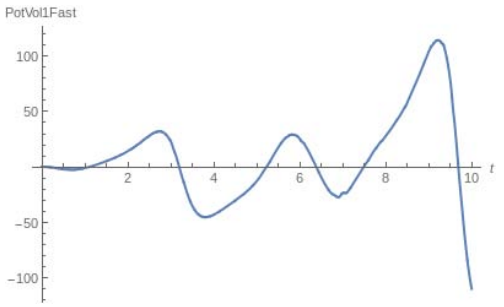
De nuevo, los únicos cambios importantes en el modelo del sistema se hacen en las ecuaciones de entrada. Se asume el máximo de velocidad rotacional observado en pruebas para los volantes, de cerca de 9000 rpm, o aproximadamente 940 rad/s. Se establece una velocidad de rotación de $\pi/6$ rad/s para las cajas de los volantes, de manera que el cambio de dirección de la resultante se alcanza, partiendo de la posición inicial, en 3 segundos, como se deseaba. Con los parámetros indicados, y a una velocidad de rotación de las cajas de los volantes ligeramente mayor en magnitud a $\pi/6$, la esfera se "vuelca". Las gráficas de este caso no se muestran, dado que este dimensionamiento es sólo una guía burda para adquisición.

$$\begin{aligned}
&\text{Entrada 1:} \quad \dot{\beta}_{pend} = 0 \\
&\text{Entrada 2:} \quad \dot{\alpha}_{pend} = 0 \\
&\text{Entrada 3:} \quad \dot{\varphi}_{vol_1} = \pi/6 \\
&\text{Entrada 4:} \quad \dot{\gamma}_{vol_1} = 940 \\
&\text{Entrada 5:} \quad \dot{\varphi}_{vol_2} = -\pi/6 \\
&\text{Entrada 6:} \quad \dot{\gamma}_{vol_2} = -940
\end{aligned} \tag{3.48}$$

En el prototipo real, sólo un motor mueve ambos CMG, usando un arreglo de engranes cónicos para asegurar que ambos roten en oposición. Por esta razón, se muestra la suma de las magnitudes de los pares de entrada en φ_{vol_i}. Además, se muestra la potencia necesaria para mantener a uno de los volantes girando a velocidad constante, usando el motor *brushless*.



(a) Suma de pares (absoluta), durante 10 s



(b) Potencia consumida por uno de los motores *brushless*

Figura 3.12

Interesa, en particular, la magnitud máxima del par a lo largo de la evolución del sistema, de forma más bien burda. Un valor pico de 12 N·m sugiere el máximo par del motor, a una velocidad cercana a cero, pues $\pi/6 \approx 1$ rad/s ≈ 10 rpm. Un motor de cerca de 12 N·m a rotor bloqueado sería una buena elección.

En el caso de los motores *bruhsless*, interesa la potencia necesaria para mantener a cada motor girando a velocidad constante aún con perturbaciones externas. Una potencia pico de aproximadamente 120 W es necesaria para mantener a uno de los motores girando a velocidad constante mientras se usan los volantes como CMG. Sin embargo, la potencia necesaria para alcanzar la velocidad de crucero de los volantes, partiendo del reposo, es en general considerablemente mayor.

Se puede asumir que, si la potencia proporcionada por el motor es constante hasta la velocidad de crucero, y la inercia del volante con respecto al eje z es de 0.005 kg·m^2, entonces la energía cinética del volante a velocidad de crucero es:

$$E_{cr} = 0.5 \cdot I_{zz} \cdot w^2 = 0.5 \cdot 0.005 \cdot 940^2 = 2209\,\text{J} \tag{3.49}$$

Para alcanzar esta energía cinética a partir del reposo, en un tiempo de 5 s, la potencia del motor brushless debe ser entonces:

$$\frac{2209\,\text{J}}{5\,\text{s}} \approx 440\,\text{W} \tag{3.50}$$

Esta es una potencia común entre motores brushless para aeromodelismo. Sin embargo, una vez alcanzada la velocidad de crucero se necesita menos potencia.

Capítulo 4

Diseño y construcción del prototipo

4.1 Diseño mecánico y manufactura

Después de definir la configuración mecánica para el robot, se comenzó el diseño a detalle del prototipo. El diseño mecánico se realizó con el software NX 7.5, y se dividió en tres módulos: esfera, péndulo y giroscopios de control de momento, aunque debido a las características del principio de funcionamiento escogido, estos módulos no pudieron ser diseñados de forma completamente independiente.

Debido a la necesidad de confinar todos los componentes mecánicos dentro de la esfera, el diseño a detalle fue un proceso iterativo. Las diferentes partes se fueron dibujando, y adecuando en forma paralela, para asegurar que no existieran problemas de interferencia, y que el tamaño del robot fuera el menor posible.

Para la selección de los motores y elementos mecánicos, se utilizaron algunos parámetros auxiliares, obtenidos de las simulaciones del modelo matemático del robot descrito en el capítulo anterior. Los parámetros utilizados y los elementos seleccionados se mencionan más adelante en la sección correspondiente.

4.1.1 Esfera

La esfera, que rodea al robot, está dividida en dos hemisferios cortados a lo largo de su ecuador, con la finalidad de poder separarse para montar la estructura

y los componentes internos del robot. A sus lados, tiene dos orificios, opuestos diametralmente, en los cuales se colocan dos placas que sostienen al eje principal del péndulo. Estas placas también sirven para cerrar la esfera, junto con otras placas más pequeñas, dispuestas alrededor de la unión de los hemisferios, a manera de broches, que a la vez sostienen en su lugar a un anillo metálico colocado a lo largo de la unión, para ayudar a la esfera a conservar su forma.

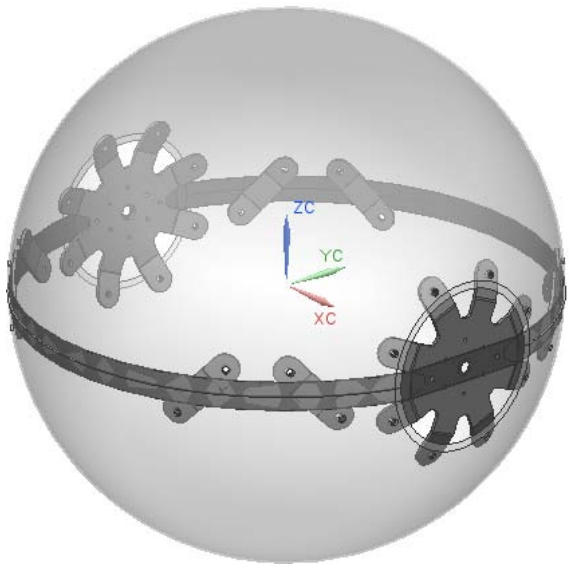


Figura 4.1 Esfera con placas de unión y anillo.

Manufactura

Para fabricar el prototipo se adquirió una esfera de policarbonato transparente, de 20 pulgadas de diámetro, para lámparas de alumbrado público. Ésta es una sola pieza con un orificio de 5.25 pulgadas de diámetro en su parte "inferior", que se usa como entrada de la lámpara. Para cortar la esfera en los dos hemisferios ya mencionados, ésta se montó en un torno, y se fijó en el portaherramientas un taladro Dremel con un disco de corte para plástico. Haciendo girar lentamente la esfera sobre el eje del torno, el taladro cortó sobre el ecuador de la esfera. Posteriormente, se hizo un segundo orificio lateral (opuesto al orificio original de la esfera, y del mismo diámetro), y se taladraron perforaciones para atornillar las placas de unión a la esfera.

Figura 4.2 Esfera de policarbonato transparente para alumbrado público.

Las placas de unión de la esfera, de lámina de acero calibre 12, se cortaron en una máquina de corte por chorro de agua, y posteriormente se hicieron algunos dobleces a las puntas de las placas, similares a estrellas, para adecuar su forma a la curvatura de la esfera.

El anillo metálico, que rodea la unión entre hemisferios de la esfera, se hizo con lámina flexible de acero templado calibre 22 de 1 pulgada de ancho. Se formó flexionando la lámina y atornillándola a las placas de unión laterales.

4.1.2 Péndulo

La configuración mecánica elegida tiene como principal sistema de locomoción, un péndulo de dos grados de libertad que requiere de dos actuadores para controlarse. Por medio de la manipulación del péndulo se puede mover la esfera.

Se eligió un sistema de engranes cónicos en configuración diferencial, para dar movimiento al péndulo sobre los dos grados de libertad requeridos. Dicho sistema fue utilizado en el prototipo de robot esférico desarrollado por G. Schroll [18]. El sistema está formado por dos engranes cónicos coaxiales dispuestos en contacto con un tercer engrane, este último fijo al eje diametral de la esfera. Los dos primeros engranes, conectados a motores, se pueden mover en relación con el tercer engrane. Este último engrane transmite fuerza y par a la esfera mediante el eje diametral (eje principal), el cual está acoplado a la esfera por medio de bridas.

Cuando los engranes conductores son actuados en dirección opuesta, el péndulo gira alrededor del eje principal, ocasionando que la esfera ruede. Cuando los engranes son actuados en la misma dirección, el péndulo se mueve sobre su otro eje de rotación, conformado por dos ejes secundarios, uno por cada engrane conductor. Esto ocasiona una inclinación de la esfera, al incrementar o disminuir el ángulo entre el eje principal y el péndulo.

Estos dos movimientos se pueden superponer, permitiendo a la esfera moverse en línea recta, o trazando arcos. Giros alrededor del eje z no son posibles sin el uso de los volantes.

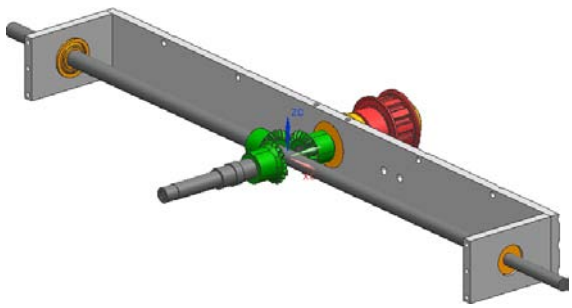


Figura 4.3 Arreglo diferencial de engranes.

Los ejes secundarios, sobre los cuales están montados los engranes conductores, se hacen girar con dos motores con reducción planetaria de varias etapas. Estos motores están montados en la parte inferior del péndulo, y mueven a los engranes a través de una transmisión de poleas y bandas dentadas con relación 1:1.

Del dimensionamiento para el motor del péndulo presentado en el capítulo anterior, se determinó que se requiere un par nominal de al menos 15 N·m, y una velocidad nominal mayor a 150 rpm. El motor seleccionado fue un motor AM9015 con reducción PG71, que proporciona un par a rotor bloqueado de 30 N·m, y una velocidad nominal de 216 rpm. Las especificaciones completas de este motor se pueden encontrar en el apéndice A.

La relación entre las velocidades del péndulo y la de los motores es la siguiente:

$$\begin{aligned} vel_{pend_1} &= \frac{vel_{mot_1} + vel_{mot_2}}{2} \\ vel_{pend_2} &= \frac{vel_{mot_1} - vel_{mot_2}}{2} \end{aligned} \tag{4.1}$$

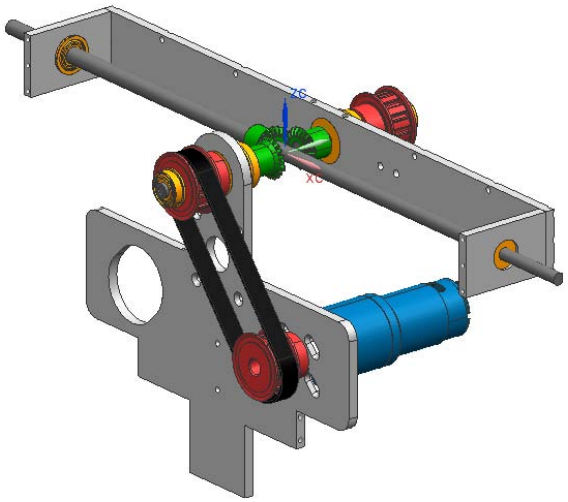


Figura 4.4 Transmisión por bandas dentadas de los motores al péndulo.

Cada uno de los ejes del péndulo está apoyado en sus extremos por dos rodamientos, montados en la estructura superior del péndulo, que alinea los ejes y mantiene la perpendicularidad y coplanaridad necesaria entre ellos.

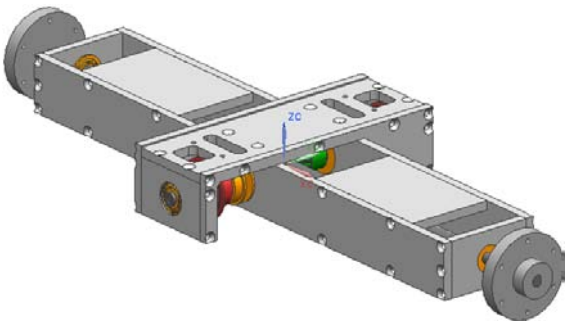


Figura 4.5 Estructura superior del péndulo.

La estructura inferior del péndulo está formada por dos placas paralelas que van montadas sobre los ejes secundarios, apoyadas también con rodamientos que permiten el movimiento del péndulo sobre su segundo eje de rotación. Sobre estas placas van montados los motores, que se colocaron en la parte inferior del péndulo con el objeto de que el centro de masa del péndulo esté a la máxima distancia del centro de la esfera, ya que entre más "abajo" se encuentre, mayor es el par que puede producir el péndulo para el movimiento de la esfera.

El péndulo cuenta con dos placas de plástico para el montaje de componentes y

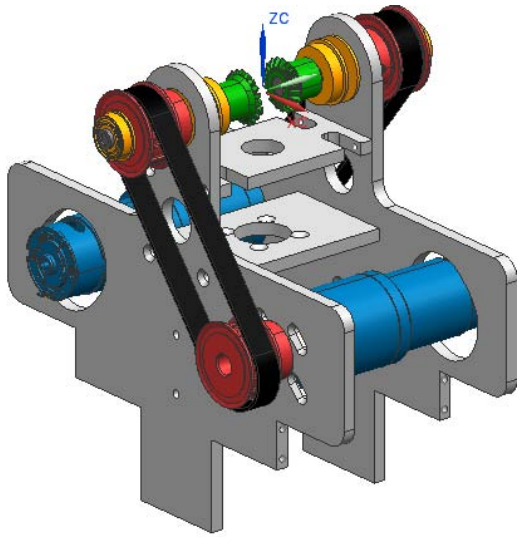


Figura 4.6 Estructura inferior del péndulo.

dispositivos. La primera se encuentra sobre la estructura superior del péndulo, y en ella van dispuestos los componentes electrónicos de robot. La segunda se encuentra en la parte inferior del péndulo, y sobre ésta van montadas las baterías de polímero de litio. Al igual que con los motores, se buscó que las baterías estuvieran a la máxima distancia del centro de la esfera.

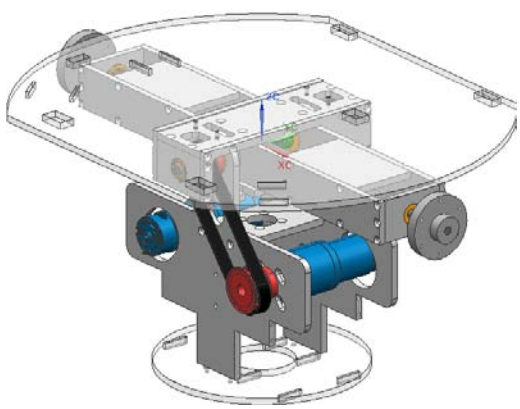


Figura 4.7 Péndulo con placas para montaje de componentes y dispositivos.

Manufactura

Para facilitar la manufactura, todas las piezas de la estructura del robot fueron diseñadas con placa de aluminio de 1/4 de pulgada. Las piezas se cortaron en una máquina de corte por chorro de agua, y después se maquinaron barrenos, cuerdas y avellanados.

De igual manera, todos los ejes del robot, tanto los de transmisión como los de soporte, fueron manufacturados en acero, y tienen escalonamientos para mantener en la posición deseada a los engranes, poleas y bridas que sostienen.

Las bridas, que sirven de acoplamiento entre los diferentes ejes del robot y las placas de la estructura, fueron manufacturadas en aluminio y tienen tornillos opresores para la sujeción a los ejes.

Las placas para el montaje de los dispositivos electrónicos fueron cortadas en máquina de corte por chorro de agua, y son de lámina de acrílico transparente de 6 mm de espesor.

Los engranes cónicos, las poleas y las bandas dentadas fueron adquiridas a través de una página de Internet. Detalles de las especificaciones de cada uno de estos elementos pueden encontrarse en el apéndice A.

4.1.3 Giroscopios de control de momento

Los CMG están formados por un volante de inercia que se hace girar mediante un motor *brushless* de relativamente alta potencia (calculada de al menos 440 W en la última sección del capítulo anterior), y una estructura de soporte que encierra al volante, a manera de jaula o caja, para que el volante pueda hacerlo de forma segura.

El volante de inercia es un cilindro sólido de acero, con un hueco en la parte superior que permite que el motor *brushless* quede acoplado dentro del volante. En la parte superior del volante sobresale la base del motor, para que sea sujetado, y puedan salir sus cables de conexión. En la parte inferior del volante se acopla un pequeño eje que se apoya sobre un rodamiento que soporta al volante, y le ayuda con la alineación y el giro, mitigando la carga sobre el rodamiento del motor.

Del dimensionamiento de los motores *brushless* realizado en el capítulo anterior,

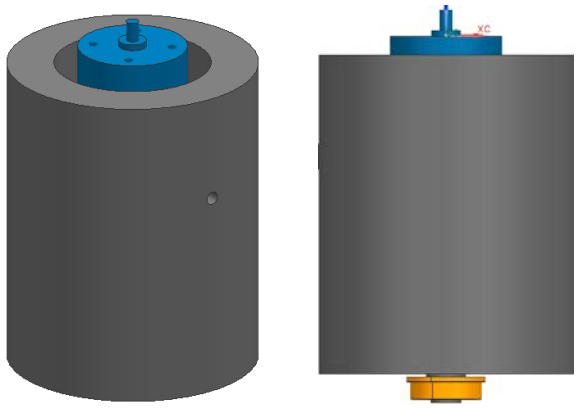


Figura 4.8 Volantes con motor *brushless* y rodamiento de apoyo

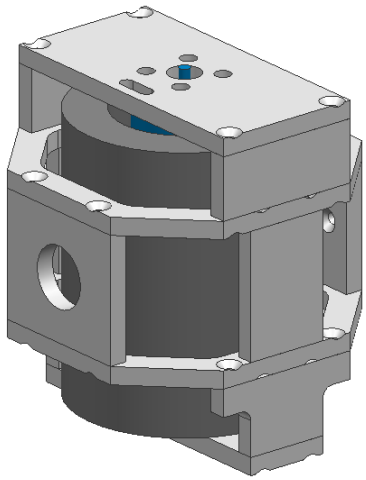


Figura 4.9 Soporte del volante de inercia

se determinó que se requiere un motor de aproximadamente 440 W para que los volantes alcancen una velocidad de 940 rad/s (8981 rpm). Los motores utilizados fueron los motores NTM Prop Drive 35-30, que entregan una potencia de 560 W a 15 V y pueden alcanzar una velocidad de hasta 21000 rpm a ese mismo voltaje. Especificaciones más completas de este motor se pueden encontrar en el apéndice A.

Para obtener el comportamiento característico de un CMG, es necesario hacer girar el soporte de los volantes de inercia sobre un eje perpendicular al eje de rotación del volante. Para ello, cada soporte está acoplado por uno de sus lados, mediante una brida, a un eje que transmite movimiento de un motor con reducción

a través de un juego de engranes cónicos, distinto del juego usado para la tracción. Los engranes tienen una reducción 1:2 y debido a su disposición, transmiten movimiento de rotación en sentidos opuestos a cada uno de los CMG, aspecto necesario para obtener el movimiento deseado de los mismos, cancelando pares indeseados. El motor con reducción está montado de forma vertical dentro de la estructura inferior del péndulo, y los ejes de transmisión están apoyados sobre rodamientos montados en diferentes placas de la estructura del péndulo.

El motor seleccionado fue un motor RS775 con reducción PG71 que proporciona un par a rotor bloqueado de 22.5 N·m, y tienen una velocidad nominal de 75 rpm. Más especificaciones de este motor se presentan en el apéndice A.

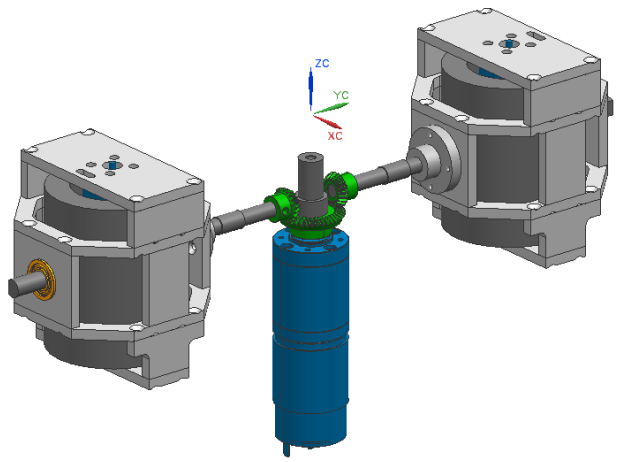


Figura 4.10 Los dos CMG con arreglo de engranes y motor.

Los soportes de los volantes están apoyados por el otro lado en placas unidas a la parte inferior del péndulo, de modo que toda la estructura de ambos CMG forma también parte del péndulo del robot.

Manufactura

Los volantes fueron manufacturados en tochos cilíndricos de acero, y se les hizo una perforación con forma especial para acoplarse al eje de los motores *brushless*. Para asegurar la sujeción del volante al eje del motor, se colocaron dos opresores que aprisionan al eje.

Como se mencionó en la sección del péndulo, las piezas de la estructura de los

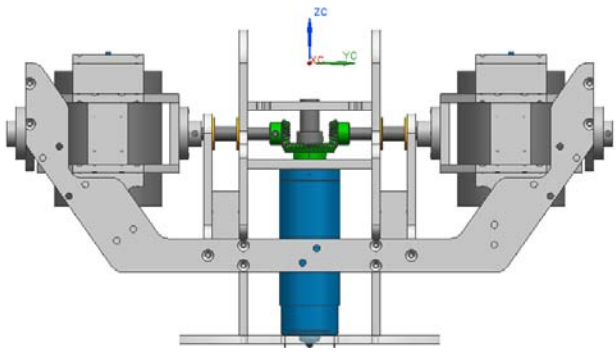


Figura 4.11 Los CMG montados en la estructura inferior del péndulo.

CMG fueron manufacturadas en placa de aluminio, los ejes de transmisión y de soporte en acero, y las bridas en aluminio.

Los engranes cónicos fueron adquiridos a través de una página de Internet, y sus especificaciones pueden ser encontradas en el apéndice A.

4.2 Componentes electrónicos

4.2.1 Sensado de los ángulos del péndulo y de la esfera

En la placa de los componentes electrónicos, se dispuso una IMU (*Inertial Measurement Unit*, con un acelerómetro, giroscopio y magnetómetro, cada uno de tres ejes) acoplada fijamente para obtener la orientación del ángulo del eje de la esfera, bajo el formato de *roll*, *pitch* y *yaw* (alabeo, cabeceo y guiño), que puede traducirse directamente a la aplicación ordenada de matrices de rotación sobre los ejes x, y, z del sistema acoplado al robot, respectivamente. De la misma forma, una IMU fija al péndulo obtiene los ángulos del péndulo, con respecto a una vertical que pasa por el centro geométrico de la esfera. Ambas IMU son leídas mediante el protocolo I^2C, con un BeagleBone Black, y procesadas con un filtro DCM (*Direction Cosine Matrix*) y observadores Kalman, para obtener una lectura filtrada, junto con las velocidades y aceleraciones de los ángulos de interés.

Las IMU se leen a una frecuencia de 50 Hz, y se hace un primer procesamiento con el algoritmo DCM, considerando un valor de *yaw* nulo (en general, no importa la orientación de la esfera con respecto a la línea norte-sur, y la cercanía de los motores a la IMU, de considerarse el *yaw*, introduce excesivo ruido). El

procesamiento para el filtrado Kalman toma los datos a una frecuencia de 10 Hz, descartando el resto de las muestras. Esto se hace por limitaciones en la capacidad de cómputo, y por la relativa ineficiencia del lenguaje Python. Versiones futuras en C++ podrían equiparar ambas frecuencias.

La IMU seleccionada fue una Sensor Stick versión 13 (SEN-10724) de SparkFun, y el algoritmo de procesamiento de los datos de los sensores fue el sugerido por el fabricante, con modificaciones menores para su uso en un BeagleBone Black.

4.2.2 Sensado de las velocidades de los motores

Los motores se leen en velocidad con codificadores magnéticos de efecto Hall, de posición absoluta de 10 bits, usando una biblioteca en C++ creada para el BeagleBone Black. No pueden usarse para medir posición, porque no hay un ángulo "cero" bien definido para los ejes de los motores en la configuración de engranes cónicos de transmisión usada (los motores podrían girar un ángulo arbitrario dejando al péndulo en una posición de reposo esencialmente indistinguible de la inicial); además, el juego de las bandas introduce un efecto de histéresis considerable, limitando severamente la utilidad de los codificadores para obtener ángulos de posición. Pero una vez que los motores se mueven en una dirección, superando la histéresis, las velocidades y aceleraciones pueden obtenerse con un filtro Kalman sobre la lectura de los codificadores.

Como la lectura de los codificadores se hace cada cierto intervalo de tiempo, y estos devuelven datos dentro de una revolución, debe distinguirse la "dirección" en que se movió el codificador desde la última lectura, pues un cambio de ángulo entre lecturas podría ser resultado de un giro en ambas direcciones. El criterio ha sido que el intervalo de lectura sea lo bastante pequeño para que, entre dos lecturas, y a la máxima velocidad posible de los motores, el eje del motor no puede girar más de media vuelta. Esto garantiza que siempre puede recuperarse la dirección de la rotación que produjo el cambio, y permite extender el ángulo por sobre los límites en una u otra dirección (0 y 2π).

4.2.3 Procesamiento y filtrado

Como se necesita de una potencia de cálculo relativamente alta para leer las IMU, filtrar sus salidas, obtener las derivadas de sus ángulos, calcular las salidas controlador a los motores, generar y registrar datos, manejar el intercambio de paquetes por la conexión Wi-Fi, procesar las salidas del *joystick* de control e implementar salvaguardas para cuidar de los motores contra sobrecalentamiento, una plataforma con microcontrolador no es suficiente para el sistema, al menos no sin optimizaciones fuera del alcance de un proyecto de esta categoría.

Pruebas realizadas con un procesador ARM Stellaris de 32 bits y 80 MHz demuestran que, solamente el procesamiento y filtrado de las dos IMU que se usan el sistema, demanda más que la capacidad de cómputo que puede ofrecer el microcontrolador. Sin embargo, una plataforma con microcontrolador tiene algunas ventajas sobre una plataforma con sistema operativo, como manejo de módulos de salida de hardware, algunas garantías en el tiempo de recepción y manejos de eventos e interrupciones, formación de señales de salida de perfil de tiempos preciso (con error inferior al microsegundo) y por sobre todo, la certeza de que el bucle de control se ejecutará eventualmente y hará al procesamiento pertinente, si el código está exento de bucles potencialmente infinitos.

En una plataforma con sistema operativo, estas garantías no existen, pues es posible que programas del usuario o procesos internos del *kernel* del sistema "secuestren" el procesador, y no puedan procesarse a tiempo entradas o emitirse salidas. O cuando menos, no sin el uso de sistemas operativos en "tiempo real", cuyo uso escapa al alcance del proyecto. Si algún motor, por cualquier motivo, quedase activado con alguna salida de magnitud considerable, y el sistema operativo se "trabase", no sería posible recuperar el motor por un tiempo, con efectos posiblemente catastróficos. Sin embargo, se necesita una plataforma como ésta por la capacidad de cálculo que ofrece.

Por ende, se opta por la división en una unidad con alta capacidad de procesamiento y sistema operativo, y otra unidad con microcontrolador. La primera lee los sensores y calcula los controladores, la segunda recibe los comandos de salida a los motores, y genera la señal apropiada. El flujo de datos hacia el microcontrolador es periódico, y una interrupción superior a un segundo en este flujo ocasiona el apagado inmediato de los motores.

Para la unidad con sistema operativo, se usa una tarjeta de desarrollo BeagleBone Black (BBB) revisión C, con algunos de sus pines conectados a las IMU y a los codificadores de los motores. Para la unidad con microcontrolador, se usó una Stellaris Launchpad 120XL, con algunos de sus pines conectados a los controladores de los motores.

4.2.4 Suministro de energía

La energía del robot la proporcionan dos baterías de polímero de litio, de carcasa rígida, 4 celdas, con un voltaje máximo de 16.8 V y mínimo de 12.8 V, y una capacidad individual de 5 A·h. Se usan dos unidades de baterías para distribuirlas simétricamente, y minimizar la necesidad de balanceo posterior. Una carcasa rígida evita la deformación de las baterías, pero introduce el riesgo de que pueden acumular presión, e incluso estallar si intentan usarse más allá de su vida útil.

Estas baterías son ligeras para su densidad energética, y son capaces de brindar grandes corrientes de salida, en el orden de los 80 A cada una. Sin embargo, su procedimiento de carga necesita de un cargador relativamente sofisticado. En uno de los extremos del eje de la esfera, conectores para las baterías permiten la entrada de los 7 pines necesarios para la carga, que dura un tiempo aproximado de tres horas desde el voltaje mínimo. Si las baterías se descargan muy por debajo de este mínimo, podrían no recuperarse, y cesar su vida útil, por lo que tendrían que desecharse. Un voltímetro digital dispuesto permanentemente en la placa de componentes del robot muestra el voltaje de las baterías, para prevenir este escenario.

Las baterías se conectan a un riel de alimentación de acero que, a través de un interruptor general de alta corriente, proporciona energía a otro riel al que se conectan todos los componentes electrónicos.

Como el voltaje mínimo de las baterías debe mantenerse superior a 12 V, los componentes electrónicos, que necesitan de 5 V, se alimentan a través de una tarjeta de reguladores lineales con disipadores. Como el ruido de los motores puede tener efectos considerables sobre el nivel de voltaje de entrada a los reguladores y afectar la integridad de los procesadores, antes y después de la fase de regulación se disponen filtros RC de primer orden, con resistencia de 1 Ω y capacitancia de 2000 µF.

4.2.5 Etapa de potencia de los motores

Para el manejo de la potencia requerida por los motores, se usan puentes H Talon SR, alimentados desde una línea de distribución general. Cada uno de estos puentes H es capaz de manejar una corriente máxima de 60 A. La señal de entrada a estos controladores es un tren de pulsos para servomotores de modelismo, esto es, una señal a 50 Hz, con ancho de pulso entre 1000 y 2000 µs, con el motor parado a un ancho de pulso de 1500 µs, y máxima velocidad en una u otra dirección en los extremos. La precision de salida de los controladores es del orden de 10 bits, de acuerdo con el fabricante.

Para el control de los motores *brushless*, dos ESC (*Electronic Speed Controller*) de 40 A con disipación se encargan de realizar la operación de inversión y control de la velocidad de rotación de los motores. La forma de la señal de entrada es también un tren de pulsos para servomotores de modelismo, pero con salida de velocidad cero para un ancho de pulso de 1000 µs, y velocidad máxima para 2000 µs.

4.2.6 Comunicación

Como el sistema debe usarse sin algún medio de conexión cableada, la comunicación con el exterior se hace con un *router* inalámbrico miniatura localizado dentro del robot, que establece un punto de acceso al que otros dispositivos Wi-Fi con las credenciales apropiadas pueden conectarse. Se establece una dirección IP fija para el BeagleBone Black, y todos los dispositivos aprobados deben registrarse a la tabla de direcciones reservadas del *router*. La comunicación con el BeagleBone Black se puede hacer por medio de puertos, o a través de SSH (*Secure Shell*) desde una terminal remota en tierra.

El robot tiene también un adaptador inalámbrico de largo alcance, que le permitirle conectarse de manera independiente a la red establecida por algún otro punto de acceso, para actualizar el sistema o instalar programas. Es posible prescindir del *router* y establecer un punto de acceso con el adaptador inalámbrico, pero incompatibilidades con la versión del *kernel* de Linux para sistemas ARM al momento de hacer las pruebas forzaron el uso del *router*.

4.3 Arquitectura de procesamiento

Como se dispone de una tarjeta de desarrollo BeagleBone Black con sistema operativo Linux, se decidió por usar la plataforma para robótica ROS (*Robot Operating System*), para el manejo de la comunicación.

En general, en sistemas complejos como un robot, se preferiría tener alguna forma de comunicación entre todos los algoritmos o programas que se necesitan para el funcionamiento del robot. Una forma de hacerlo es mediante la creación de objetos en una clase, comunicando diferentes algoritmos a través de algunas variables globales. Esto es, sin embargo, inconveniente para propósitos de desarrollo en equipos, o para pruebas de cambios menores que necesiten evaluarse con rapidez.

Una solución a este problema consiste en crear diferentes programas o nodos, uno para cada tarea especifica del robot, y comunicar estos a través de mensajes que quizá puedan procesarse en otros nodos, a manera de un grafo sin ciclos. Ésta es la solución implementada por ROS, que es en última instancia una arquitectura de comunicaciones donde varios programas o nodos de propósito específico se comunican a través de mensajes o temas, que pueden entenderse como variables globales cuya actualización fuerza el procesamiento en los nodos que consumen tales variables.

Una computadora en la tierra, que también usa ROS, procesa las salidas del *joystick*, y manda el mensaje de movimiento deseado a través de la conexión Wi-Fi al robot. En el BeagleBone Black, varios nodos escritos en Python y C++ leen las IMU y codificadores, filtran los datos de los sensores, calculan las salidas, y forman la cadena de datos de comunicación con el microcontrolador. Al microcontrolador se envía una cadena con protocolo propio y verificación de errores, que codifica las salidas de PWM de los motores. Si por algún motivo transcurre más de un segundo sin que esta cadena se reciba en el microcontrolador, las salidas se hacen cero en la siguiente iteración del bucle principal del microcontrolador.

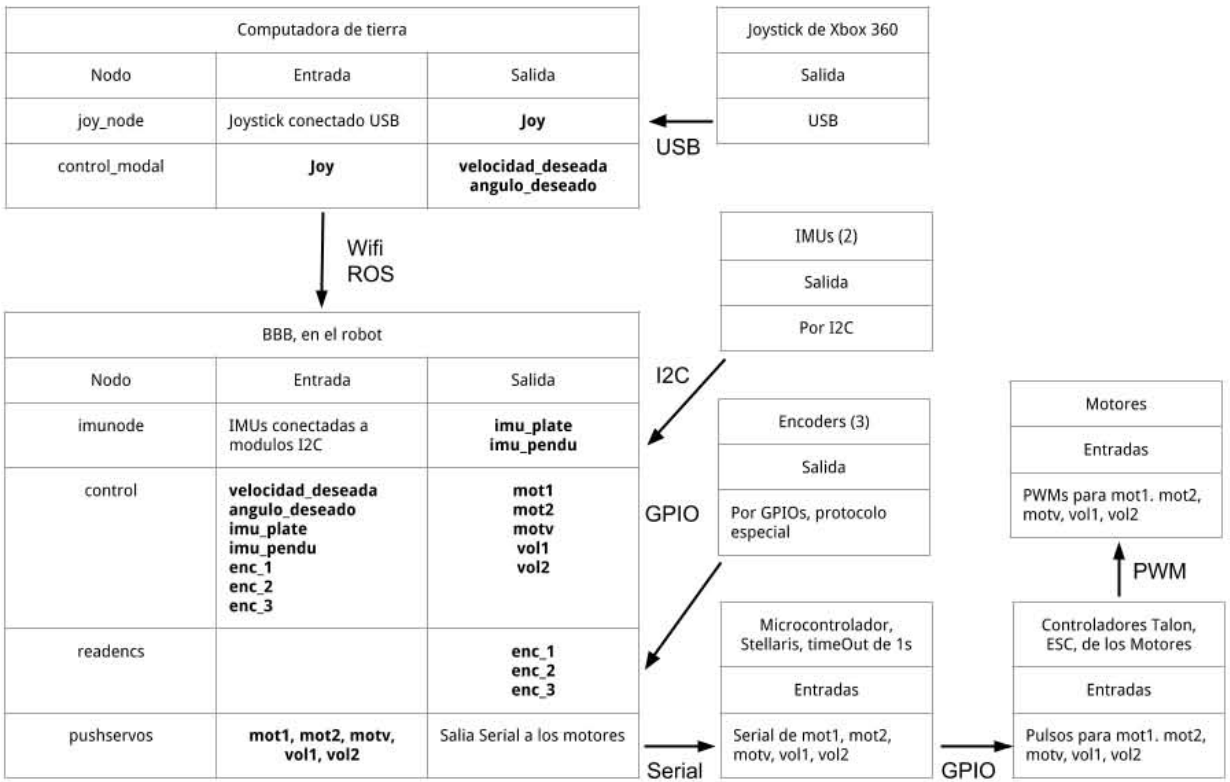


Figura 4.12 Arquitectura de cómputo.

Capítulo 5

Pruebas de control

Para propósitos de control, es de interés el ángulo del eje diametral de la esfera con respecto a alguna normal al terreno. Dado que la esfera avanza por medio del giro del péndulo alrededor del eje diametral, los cambios en su ángulo permiten que la esfera se mueva en arcos. La estabilización de este ángulo (mantenerlo igual a cero) ayuda a que la esfera avance en una línea recta. O bien, se puede virar la esfera hacia la izquierda o derecha mientras avanza, regulando el ángulo a algún valor distinto de cero.

Si bien la velocidad de avance de la esfera puede controlarse a través de la velocidad de rotación del péndulo, un controlador lineal de esta variable (la velocidad de avance) de orden mayor a uno necesitaría considerar tanto la aceleración como el *jerk* de la esfera, y el ruido del *jerk* puede ser tan grande que cualquier porción derivativa afecta excesivamente el rendimiento del controlador. Como no importan pequeños errores en la velocidad de avance, pero sí importa minimizar los errores en el viraje de la esfera, se puede considerar un control puramente proporcional, o, a lo más, de primer orden para el avance, y ajustarlo durante las pruebas. Pero es necesario un esquema más sofisticado para virar la esfera o hacer que se mueva sobre una línea recta.

En la implementación se distinguen dos modos de control, un modo "manual", que da salida a los motores sin considerar dinámicas o realimentación, y un modo "automático", que puede escogerse de entre dos posibles modos de control para el ángulo del eje. Estos modos son un controlador por realimentación de estados, y otro controlador PID con algunos términos no lineales adicionales. En el modo

automático, hay sólo un controlador proporcional de la velocidad de la esfera, con un término adicional proporcional al valor deseado.

5.1 Derivación de un controlador de estados

El modelo matemático del sistema puede usarse para derivar el controlador. Si bien es posible implementar controladores PID ajustados empíricamente para algunos de los grados de libertad del sistema, generalmente esto involucra una gran cantidad de pruebas de ensayo y error para ajustar adecuadamente sus parámetros. Un controlador diseñado con base en el modelo del sistema brinda una pauta base a partir de la cual pueden hacerse ajustes más elaborados. Sin embargo, la complejidad del modelo generado para el sistema limita severamente la posibilidad de generar un controlador lineal sobre muchos de los estados, por la dificultad de encontrar expresiones cerradas para las segundas derivadas de las variables de interés.

Interesa controlar el ángulo entre el péndulo y el eje diametral, el ángulo del eje diametral con respecto a una normal al terreno, y la velocidad de rotación del péndulo a medida que gira alrededor del eje. El controlador del ángulo debe mitigar las pequeñas variaciones en el ángulo del eje diametral, con el efecto de desestabilizar la esfera o desviarla de la trayectoria deseada. Al moverse, la esfera puede describir arcos de circunferencia si su eje diametral tiene algún ángulo distinto de cero en relación al plano. El centro de giro es el punto en el cual la línea que contiene al eje diametral hace contacto con el plano. Si se asume, por ejemplo, un radio de r_e para la esfera, la relación entre el ángulo α que el eje diametral forma con el plano y el radio de giro r_g esta dada por:

$$r_g = r_e \cdot cot(\alpha) \tag{5.1}$$

Para un desplazamiento de $1m$ en distancia recorrida (longitud de arco), partiendo desde el origen y avanzando en sentido horario sobre un circulo de radio r_g con centro $(r_g, 0)$, el desplazamiento final sobre la coordenada x esta dado por la relación:

$$x = -\sqrt{{r_g}^2 - r_g \cdot sen\left(\frac{1}{r_g}\right)} + r_g \tag{5.2}$$

Tabulando el desplazamiento final sobre la coordenada x a varios valores del ángulo del eje diametral α (en grados), para un radio de la esfera de $r_e = 0.2495m$.

Tabla 5.1 Desplazamiento en x a un recorrido de 1 m

α	$r_g[m]$	$x[m]$
1°	14.2938	0.0349
2°	7.1447	0.0698
3°	4.7607	0.1046
5°	2.8518	0.1735
10°	1.4149	0.3388
15°	0.9311	0.4873

Se desea limitar el ángulo α a valores inferiores a tres grados, para que la desviación de la trayectoria rectilínea no sobrepase en un 10 por ciento el avance hacia adelante. Desviaciones mayores además podrían tener el efecto de desestabilizar el robot.

El control de la velocidad se implementa como uno basado en el error de tipo proporcional, y se ajusta en las pruebas. Esto es aceptable, porque si bien se puede conocer la orientación con respecto al sistema acoplado a tierra de los ángulos del péndulo, y se puede calcular el ángulo entre la línea del péndulo y el eje diametral, no es posible conocer directamente los ángulos del cuerpo de la esfera, porque ésta podría rotar ilimitadamente para una posición de reposo del péndulo, y el cuerpo de la esfera no tiene sensores acoplados fijamente a éste. Por ello, implementar cualquier forma de control basada en el error para la velocidad de la esfera requeriría de algún método de observación de sus ángulos, introduciendo dinámicas complejas. Por ello, se opta por controlar solamente el ángulo de viraje o inclinación de la esfera de manera más sofisticada, para intentar mantenerla en una trayectoria recta, si así lo desea el operador, o trazar arcos.

Por limitaciones en la disponibilidad de partes, el control del sistema completo se escribió y probó sin involucrar entradas a los volantes, aunque sí se considera su masa, ya que su operación puede considerarse independiente del funcionamiento normal del péndulo, en condiciones ideales. Se usa el modelo de motor con entrada

de voltaje, como aproximación de los puentes H con salida de ancho de pulso modulado (PWM).

Para simplificar el manejo del sistema, se considera un escenario ideal, en el que el péndulo no es actuado en el ángulo que permite el avance de la esfera (β_{pend}), pero sí tiene una entrada en el ángulo que permite el viraje (α_{pend}). Esto restringe el movimiento de la esfera a un plano que incluye al origen y al eje diametral. Bajo tal simplificación, el centro de la esfera sólo puede avanzar sobre la coordenada y, cuando la esfera cambia su ángulo ϕ_{esf}. Se asume que un controlador capaz de controlar el ángulo del eje diametral con la esfera parada es adecuado para la esfera en movimiento de avance. Por ello, es necesario reescribir la condición de no deslizamiento, como una equivalencia entre el desplazamiento en y y el arco de la esfera dado por el cambio en ϕ_{esf}.

Dado que se trata de un escenario restringido, el proceso de derivación del controlador asume que algunas de las coordenadas generalizadas son siempre cero. Esto se hace para eliminarles de las ecuaciones del modelo completo, y facilitar el tratamiento en el software de cálculo simbólico. En particular, se eliminan ocho de las coordenadas generalizadas, una de las entradas de voltaje y uno de los multiplicadores de Lagrange. Esto es, deben realizarse las siguientes sustituciones:

$$
\begin{aligned}
x(t) = 0, \quad & \dot{x}(t) = 0 \\
\psi_{esf}(t) = 0, \quad & \dot{\psi}_{esf}(t) = 0 \\
\theta_{esf}(t) = 0, \quad & \dot{\theta}_{esf}(t) = 0 \\
\alpha_{pend}(t) = 0, \quad & \dot{\alpha}_{pend}(t) = 0 \\
\varphi_{vol_1}(t) = 0, \quad & \dot{\varphi}_{vol_1}(t) = 0 \\
\gamma_{vol_1}(t) = 0, \quad & \dot{\gamma}_{vol_1}(t) = 0 \\
\varphi_{vol_2}(t) = 0, \quad & \dot{\varphi}_{vol_2}(t) = 0 \\
\gamma_{vol_2}(t) = 0, \quad & \dot{\gamma}_{vol_2}(t) = 0 \\
V_{\alpha_{pend}}(t) = 0 & \\
\lambda_1(t) = 0 &
\end{aligned}
\tag{5.3}
$$

Las únicas coordenadas generalizadas que interesan son y, β_{pend} y ϕ_{esf}. Los parámetros usados para la simulación, que pueden verse en la tabla 3.6, son utilizados también para la derivación del controlador, pues el sistema, físicamente, no ha cambiado.

Existe una dependencia lineal entre y y ϕ_{esf} para el sistema restringido, pues se eliminan dos de los grados de libertad de rotación de la esfera en la condición de no deslizamiento. Es conveniente reducir el sistema a las variables ϕ_{esf} y β_{pend}, eliminando λ_2 y reescribiendo y en términos de ϕ_{esf}. Esta reducción debe considerar las derivadas pertinentes, pues debe también eliminarse $\dot{y}$. Al final, se consideran solamente las ecuaciones E-L 2, E-L 5 y E-L 6 (de las ecuaciones 3.7), así como la ecuación Restricción 2 bajos las sustituciones del sistema simplificado (ecuaciones 3.37), junto con su primer derivada:

$$\begin{aligned}
\text{E-L 2:} \quad & \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{y}}\right) - \frac{\partial L}{\partial y} = 0 \\
\text{E-L 5:} \quad & \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\phi}_{esf}}\right) - \frac{\partial L}{\partial \phi_{esf}} = 0 \\
\text{E-L 6:} \quad & \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\beta}_{pend}}\right) - \frac{\partial L}{\partial \beta_{pend}} = k_{1mot}(V_{\beta_{pend}} - k_{2mot}\dot{\beta}_{pend}) \\
\text{Restricción 2:} \quad & \dot{y} + r_{esf}\dot{\phi}_{esf} = 0 \\
\mathbf{D}(\text{ Restricción 2}): \quad & \ddot{y} + r_{esf}\ddot{\phi}_{esf} = 0
\end{aligned} \tag{5.4}$$

Reiterando, deben eliminarse las variables y, $\dot{y}$ y λ_2, reduciendo el sistema a sólo dos ecuaciones con variables β_{pend}, ϕ_{esf} y sus derivadas, y con una entrada $V_{\beta_{pend}}$. Para propósitos de la linealización, interesa obtener expresiones cerradas para $\ddot{\beta}_{pend}$ y $\ddot{\phi}_{esf}$. Estas expresiones simbólicas para $\ddot{\beta}_{pend}$ y $\ddot{\phi}_{esf}$ puede obtenerse con la ayuda de un software de cálculo simbólico.

Para realizar la linealización deben obtenerse las derivadas parciales con respecto a las coordenadas $\dot{\phi}_{esf}$, ϕ_{esf}, $\dot{\beta}_{pend}$ y β_{pend} de las segundas derivadas de β_{pend} y ϕ_{esf}. Si un sistema lineal puede expresarse como $\dot{\mathbf{x}} = \mathbf{A}\cdot\mathbf{x} + \mathbf{B}\cdot\mathbf{u}$, para obtener la matriz $\mathbf{A}$ debe considerarse que la entrada es nula, por lo que se hace la sustitución $V_{\beta_{pend}} = 0$:

$$
\begin{aligned}
d_{11} &= \ddot{\phi}_{\dot{\phi}} = \left.\frac{\partial \ddot{\phi}_{esf}}{\partial \dot{\phi}_{esf}}\right|_{V_{\beta_{pend}}=0} & d_{21} &= \ddot{\beta}_{\dot{\phi}} = \left.\frac{\partial \ddot{\beta}_{pend}}{\partial \dot{\phi}_{esf}}\right|_{V_{\beta_{pend}}=0} \\
d_{12} &= \ddot{\phi}_{\phi} = \left.\frac{\partial \ddot{\phi}_{esf}}{\partial \phi_{esf}}\right|_{V_{\beta_{pend}}=0} & d_{22} &= \ddot{\beta}_{\phi} = \left.\frac{\partial \ddot{\beta}_{pend}}{\partial \phi_{esf}}\right|_{V_{\beta_{pend}}=0} \\
d_{13} &= \ddot{\phi}_{\dot{\beta}} = \left.\frac{\partial \ddot{\phi}_{esf}}{\partial \dot{\beta}_{pend}}\right|_{V_{\beta_{pend}}=0} & d_{23} &= \ddot{\beta}_{\dot{\beta}} = \left.\frac{\partial \ddot{\beta}_{pend}}{\partial \dot{\beta}_{pend}}\right|_{V_{\beta_{pend}}=0} \\
d_{14} &= \ddot{\phi}_{\beta} = \left.\frac{\partial \ddot{\phi}_{esf}}{\partial \beta_{pend}}\right|_{V_{\beta_{pend}}=0} & d_{24} &= \ddot{\beta}_{\beta} = \left.\frac{\partial \ddot{\beta}_{pend}}{\partial \beta_{pend}}\right|_{V_{\beta_{pend}}=0}
\end{aligned}
\tag{5.5}
$$

Y, para hallar la matriz $\mathbf{B}$, sin la sustitución $V_{\beta_{pend}} = 0$, se obtienen las derivadas parciales con respecto a la entrada $V_{\beta_{pend}}$.

$$
\begin{aligned}
d_{31} &= \ddot{\phi}_{V_\alpha} = \frac{\partial \ddot{\phi}_{esf}}{\partial V_{\alpha_{pend}}} & d_{33} &= \ddot{\beta}_{V_\alpha} = \frac{\partial \ddot{\beta}_{pend}}{\partial V_{\alpha_{pend}}} \\
d_{32} &= \ddot{\phi}_{V_\beta} = \frac{\partial \ddot{\phi}_{esf}}{\partial V_{\beta_{pend}}} & d_{34} &= \ddot{\beta}_{V_\alpha} = \frac{\partial \ddot{\beta}_{pend}}{\partial V_{\beta_{pend}}}
\end{aligned}
\tag{5.6}
$$

Por simplicidad, se considera una linealización de segundo orden para un vector de estado compuesto de los estados:

$$
\begin{bmatrix} \dot{\phi}_{esf} \\ \phi_{esf} \end{bmatrix} \tag{5.7}
$$

Aunque podría haberse considerado un controlador de cuarto orden, la complejidad del modelo generado excede la capacidad de manejo del software de cálculo simbólico en el equipo de cómputo disponible. Otra razón para considerar solamente un controlador de segundo orden es mayormente práctica. Interesa estabilizar el ángulo ϕ_{esf}, a través de la entrada $V_{\beta_{pend}}$. Como se considera una representación matricial y lineal del sistema, un tipo de controlador natural es de realimentación de estados, en el cual deben asignarse polos directamente a la ecuación característica del sistema en lazo cerrado. El caso de segundo orden es relativamente trivial y permite obtener una expresión sencilla para el controlador, pero el ca-

so de cuarto orden genera expresiones de complejidad intratable. Es conveniente que la expresiones escalares finales generadas sean simples, pues deben sustituirse y probarse en la simulación, más aún, aunque el software matemático es capaz de manejar expresiones de gran tamaño, estas expresiones también deben implementarse en el software del controlador para que se calculen a cada iteración del controlador.

La restricción a una linealización de segundo orden implica que, aunque se considera un punto de operación igual al punto de velocidad y ángulo nulos para ϕ_{esf}, la matriz $\mathbf{A}$ obtenida seguirá dependiendo continuamente de los valores de β_{pend} y $\dot{\beta}_{pend}$, por lo que habrá que sustituir continuamente tales valores, obtenidos con los sensores del robot, y utilizarlos para calcular continuamente la matriz $\mathbf{A}$.

El desarrollo matemático es más fácil si se realiza primero de forma simplificada, sustituyendo al final los símbolos pertinentes por las expresiones generadas. Se puede suponer entonces que existe una matriz $\mathbf{A}$ para el sistema que tiene la forma

$$\mathbf{A} = \begin{bmatrix} a & b \\ 1 & 0 \end{bmatrix} \tag{5.8}$$

Donde la fila inferior tiene un uno y un cero dado que uno de los estados es la derivada con respecto al tiempo del otro. La matriz $\mathbf{B}$ puede asumirse de la forma

$$\mathbf{B} = \begin{bmatrix} c \\ 0 \end{bmatrix} \tag{5.9}$$

Y la matriz $\mathbf{K}$, que recoge los parámetros del controlador, tiene la forma

$$\mathbf{K} = \begin{bmatrix} k_1 & k_2 \end{bmatrix} \tag{5.10}$$

La matriz $\mathbf{A}_k$ del sistema en lazo cerrado es

$$\mathbf{A}_k = \mathbf{A} - \mathbf{B} \cdot \mathbf{K} \tag{5.11}$$

La ecuación característica del sistema en lazo cerrado es algún escalamiento proporcional del determinante de la matriz $s\mathbf{I} - \mathbf{A}_k$. Sin embargo, sólo interesan las raíces:

$$\begin{vmatrix} -a + c \cdot k_1 + s & -b + c \cdot k_2 \\ -1 & s \end{vmatrix} = 0 \tag{5.12}$$

Para propósitos de control, se propone un tiempo de asentamiento de dos segundos, con un criterio de tres constantes de tiempo para considerar que se alcanza el estado final. De esta forma, las soluciones para k_1 y k_2 son:

$$\begin{aligned} k_1 &= \frac{3 + a}{c} \\ k_2 &= \frac{9 + 4b}{4c} \end{aligned} \tag{5.13}$$

Como es un controlador por realimentación de estados, es necesario considerar algún factor de preescala para la entrada. Esto es:

$$k_0 = \mathbf{C} \cdot (-\mathbf{A} + \mathbf{B} \cdot \mathbf{K})^{-1} \cdot \mathbf{B} \tag{5.14}$$

La salida deseada del sistema es solamente el ángulo ϕ_{esf}. La salida de un sistema lineal puede expresarse como $\mathbf{y} = \mathbf{C} \cdot \mathbf{x} + \mathbf{D} \cdot \mathbf{u}$. En este sistema, la matriz $\mathbf{C}$ es de la forma:

$$\mathbf{C} = \begin{bmatrix} 0 & 1 \end{bmatrix} \tag{5.15}$$

Hallando k_0 para los valores encontrados de k_1 y k_2:

$$k_0 = \frac{c}{-b + \frac{9+4b}{4}} \tag{5.16}$$

La expresión para la entrada es de la forma:

$$u = k_0 \cdot \phi_{deseada} - \mathbf{K} \cdot \begin{bmatrix} \dot{\phi}_{esf} \\ \phi_{esf} \end{bmatrix} \tag{5.17}$$

Como el punto de operación alrededor del cual se linealiza permanece constante, el sistema linealizado tiene estados transformados, correspondientes a las diferencias

en cada instante con el punto de linealización. Como en este caso el punto de linealización corresponde a una condición de ángulo y velocidad nulas, los estados siguen siendo iguales a las coordenadas generalizadas. Sin embargo, los cambios que sí suceden en β_{pend}, ϕ_{esf} y sus derivadas tienen que sustituirse en la expresión de la matriz **A**.

Dos adiciones a este esquema de control se han encontrado útiles a través de las simulaciones del modelo y pruebas del robot. La primera es una $\phi_{deseada}$ igual al valor medido de $-\phi_{esf}$. Esto tiene el resultado de hacer el control un poco más fuerte al intentar llevar a ϕ_{esf} a cero. La segunda es que la expresión del controlador se beneficia de incluir una contribución del ángulo del péndulo β_{pend}, pues las oscilaciones del péndulo tienden a manifestarse como oscilaciones indeseadas de la esfera. Si bien esto habría sido evitable considerando una linealización de orden mayor, esto introdujo complicaciones adicionales que no son fáciles de tratar. Contrarrestando de alguna forma las oscilaciones de β_{pend}, el tiempo de asentamiento del sistema no lineal no se ve mayormente afectado, pero la trayectoria que el sistema sigue para estabilizar o regular ϕ_{esf} es más suave.

Se considera el controlador entonces de la forma:

$$u = k_0 \cdot (-\phi_{esf}) - \begin{bmatrix} k_1 & k_2 \end{bmatrix} \cdot \begin{bmatrix} \dot{\phi}_{esf} \\ \phi_{esf} \end{bmatrix} - \beta_{pend} \tag{5.18}$$

bajo las sustituciones:

$$\begin{aligned} k_0 &= \frac{c}{-b + \frac{9+4b}{4}} \\ k_1 &= \frac{3+a}{c} \\ k_2 &= \frac{9+4b}{4c} \end{aligned} \tag{5.19}$$

y éstas bajo las sustituciones de las expresiones simbólicas, quizá de considerable longitud:

$$\begin{aligned} a = d_{11} &= \ddot{\phi}_{\dot{\phi}} \\ b = d_{12} &= \ddot{\phi}_{\phi} \\ c = d_{32} &= \ddot{\phi}_{V_\beta} \end{aligned} \tag{5.20}$$

que se evalúan constantemente dadas las lecturas de los sensores para cada iteración del bucle de control.

La expresión resultante para u, en el sistema con los parámetros sustituidos, implementada en código de Python dentro de una clase, y haciendo $\beta_{pend} = beta$, $\dot{\beta}_{pend} = betap$, $\phi_{esf} = fi$ y $\dot{\phi}_{esf} = fip$, es una expresión también de considerable longitud. Se muestra como ejemplo una sección:

```
"self.u = -(((0.46137805817 + 0.036060142242*cos(self.fi) - \
0.0137267461187*cos(2*(self.beta + self.fi)))*(36*self.fip*(3. + \
(1.00000000000*self.fip*sin(self.fi) + ... \
... + 0.0137267461187*cos(2*(self.beta + \
self.fi)))**2))))/(-42.02908812 + 14.912185859*cos(self.beta + \
self.fi)))"
```

Esta expresión no tiene dinámicas internas propias, depende a lo mucho de las derivadas de estados que pueden medirse u observarse, y sólo necesita la sustitución de los valores de tales lecturas, por lo que no se realiza una operación de discretización adicional para implementar el controlador. Como éste se expresa en términos del voltaje de salida al motor, debe mapearse al valor de la salida PWM del puente H a las entradas correspondientes.

5.1.1 Simulación del controlador

Partiendo de una una condición inicial no nula $\phi_{esf_0} = 0.1$ y ángulos del péndulo nulos, el controlador logra estabilizar el sistema con la trayectoria para ϕ_{esf} como se muestra en la figura 5.1.

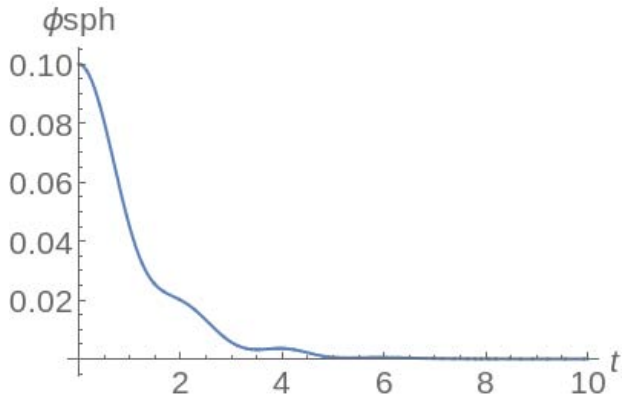


Figura 5.1 Comportamiento de ϕ_{esf} con control, por 10 s.

Como comparación, el sistema sin control, si se mantienen los ángulos del péndulo nulos (la fricción estática de los motores es lo bastante grande como para considerar esto razonable a pequeñas desviaciones del punto de operación) se comporta como se muestra en la figura 5.2.

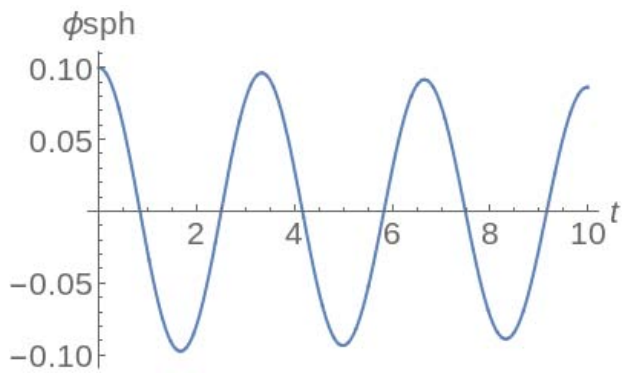


Figura 5.2 Comportamiento de ϕ_{esf} sin control, por 10 s.

A falta de fricciones, la simulación seguirá oscilando eternamente. Esto es lo que se pretende evitar con el controlador, porque si la esfera está moviéndose en línea

recta a través de terreno plano, estas oscilaciones la desviarán de su trayectoria, o podrían desestabilizarla luego de un tiempo y hacer que la esfera se vuelque, aunque puede recuperarse con facilidad, por su forma esférica.

En el sistema real, se realizaron pruebas para el sistema sin ninguna forma de control, con controladores PID ajustados por ensayo y error, y con el controlador obtenido anteriormente para el ángulo de inclinación del péndulo.

La velocidad de avance de la esfera usa solamente un control proporcional, en parte porque, aunque sería posible observar la velocidad de rotación de la esfera, cualquier modelo de al menos segundo orden para el control de ésta requeriría un término de *jerk*, además de la aceleración del ángulo, y el ruido resultante del cálculo de éste, aún filtrado, es lo bastante grande como para inutilizar su uso. Como no importan pequeños errores en la velocidad de avance, la contribución integral se considera también superflua.

5.2 Controlador PID

El controlador del ángulo del eje de la esfera se puede ajustar en tiempo real a través de una interfaz gráfica que permite variar las constantes. La expresión general de este esquema de control es

$$\%PWM = k_p e + k_i \int e + k_d \dot{e} + k_a \ddot{e} + k_s \sin(e) + k_m y_d \tag{5.21}$$

Donde el error es e y el valor deseado es y_d. Las diferentes constantes k se pueden ajustar en tiempo real. Dependiendo de si lo que se desea es un control de posición (y como a error cero la salida del controlador debe ser cero, entonces $k_m = 0$) o un control de velocidad (donde $k_s = 0$), puede ser conveniente el despreciar la contribución de la segunda derivada del error, por el ruido que pueda presentarse a pesar del procesamiento y filtrado. La magnitud de la segunda derivada del ruido eléctrico o proveniente de los sensores podría ser tan grande que sobrepase las contribuciones de los demás términos del controlador, inutilizándolo. La constante k_s puede usarse para mitigar el efecto no lineal del péndulo, y k_m se incluye porque para el control de velocidad de un motor, se necesita de una salida distinta de cero a error cero para velocidades deseadas no nulas.

El término integral tiene algunas restricciones adicionales en la implementación. En particular, se define un umbral para el valor absoluto del error, también ajustable por el usuario, de manera que para valores del error superiores al umbral exista una k_{i1} y para valores dentro del umbral, otra k_{i2}. Esto se hace porque, para desviaciones muy grandes con respecto al cero para el ángulo del eje de la esfera, la recuperación de la posición de avance en línea recta necesita de un control muy diferente. Esto es, no es conveniente que el controlador que trata de recuperar el robot desde una "caida" intente regular el ángulo para pequeñas desviaciones del cero. En la recuperación, se desea que se dé un "golpe" para que el robot vuelva a una posición cercana a la inicial.

La salida generada puede transmitirse a los motores a través de los puente H. Si ambos motores tienen salida con el mismo signo, el robot presenta un movimiento de avance; con signos opuestos, modifica el ángulo de su eje diametral para trazar arcos, por la disposición de la transmisión con engranes cónicos. Las salidas de ambos tipos de movimiento pueden sumarse para que el robot los efectúe en paralelo. Por ejemplo, si existe un controlador de la velocidad de avance del robot con salida s_{vel}, y un controlador del ángulo con salida s_{ang}, a los motores se les daría salida:

$$\begin{aligned} m_1 &= s_{vel} + s_{ang} \\ m_2 &= s_{vel} - s_{ang} \end{aligned} \tag{5.22}$$

5.3 Modos

En el funcionamiento controlado del robot se puede distinguir un gradiente de operación, de un estado de baja velocidad, hasta uno de alta velocidad. En el modo de baja velocidad, para el funcionamiento del robot es más importante el control del ángulo del eje que la velocidad de avance, en tanto que a mayor velocidad, se necesita una menor diferencia en las salidas de los motores para lograr el mismo viraje del ángulo, y así lograr que el robot se mueva hacia los lados. Esto sucede porque a bajas velocidad, los efectos de las fricciones, inercias, zonas muertas e histéresis son mayores, relativamente, a los experimentados a altas velocidades.

En el robot, este principio se aplica agregando, a bajas velocidad, un valor mínimo de salida para superar la zona muerta (el 5 % del ancho de pulso de salida por restricciones del controlador de los motores), pero multiplicando esta salida mínima por una exponencial negativa en función de la velocidad de cada motor, que se lee con los codificadores. De esta manera, esta contribución agregada para sobrepasar la zona muerta y el efecto de la inercia en el estado de reposo disminuye rápidamente una vez que los motores y la esfera se encuentran ya en movimiento.

5.4 Filtros

Se usa un filtro DCM para cada IMU, porque es el recomendado por el fabricante (SparkFun), para procesar los datos de los sensores y generar la información de los ángulos de *roll* y *pitch*. Estos datos luego son procesados con filtros Kalman que sirven también de observadores para obtener las velocidades y aceleraciones del *roll* y *pitch*.

Los datos de los codificadores de los ejes de los engranes cónicos centrales también se procesan de manera similar, el ángulo extendido del eje de los motores (que puede ser menor a 0 o mayor a 2Π) pasa a una etapa de filtrado Kalman para observar la velocidad y aceleración del giro de cada motor, siendo la velocidad de avance la suma de las dos velocidades de los motores, y la velocidad de cambio del ángulo, la diferencia. Como hay juego entre los engranes y como consecuencia histéresis, estos sensores no son útiles para medir, al menos directamente, la posición angular, porque el error introducido en la lectura sería demasiado grande para considerarla útil. Sin embargo, una vez que un engrane está en movimiento continuo en una dirección, se habrán superado ya los efectos del juego y de la histéresis, por lo que la medición sería de utilidad para, mediante la lectura a intervalos regulares del ángulo de salida de los codificadores, obtener la velocidad de rotación de los engranes centrales.

En ambos casos, se usa una matriz **A** que representa el caso general de estimación de derivadas en ausencia del modelo, para una sola variable de entrada de la cual se desean observar sus primeras dos derivadas y que se supone de aceleración prácticamente constante. En este caso, la entrada es el ángulo leído, y las derivadas

deseadas son la velocidad y la aceleración:

$$\mathbf{A} = \begin{bmatrix} 1 & dt & dt^2/2 \\ 0 & 1 & dt \\ 0 & 0 & 1 \end{bmatrix} \tag{5.23}$$

Los valores de esta matriz se obtienen de considerar las ecuaciones de movimiento uniformemente acelerado:

$$\begin{aligned} s &= s_0 + v \cdot t + \frac{a \cdot t^2}{2} \\ v &= v_0 + a \cdot t \end{aligned} \tag{5.24}$$

Y matrices $\mathbf{Q}$ y $\mathbf{R}$ diagonales, con valores en la diagonal de 10.

$$\mathbf{Q} = \mathbf{R} = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 10 \end{bmatrix} \tag{5.25}$$

Y una matriz $\mathbf{H}$ diagonal, con valores en la diagonal de 1.

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.26}$$

5.5 Pruebas de funcionamiento

Es conveniente describir algunos detalles generales de la realización del robot y sus sistemas electrónicos, en particular, en lo que concierne a las IMU y su procesamiento, y en la forma en que se han implementado los algoritmos de control.

5.5.1 Correcciones a las IMU

Además del procesamiento de las señales de salida de las IMU con filtros Kalman, se hace la corrección, previa al filtrado, de incrementar ligeramente la magnitud de las salidas de los giroscopios antes de aplicar el algoritmo DCM para la obtención de *roll*, *pitch* y *yaw*. Esto se vio útil para reducir el tiempo de levantamiento del

filtro, a costa de introducir un ligero sobrepaso, que permanece dentro de umbrales aceptables. El valor de *yaw* se hace nulo, ya que la cercanía de los motores dificulta la distinción del norte magnético verdadero.

El algoritmo DCM, que puede encontrarse implementado en otros repositorios y artículos, es relativamente sensible a cambios rápidos de los ángulos leídos, generando valores que salen del rango de un tipo de datos flotante, y arrastrando luego indefiniciones o infinitos durante todas las iteraciones siguientes del algoritmo. Esto se ha mediado estableciendo una etapa de revisión para todas las variables pertinentes que podrían provocar uno de tales errores, verificando que tales valores, y los resultados de operaciones con éstos, sean finitos. De no ser así, se descartan datos hasta que las condiciones de finitud se cumplan nuevamente. Este problema es transiente, y en general el descartar algunos de las tramas de datos de las IMU no afecta sensiblemente la operación del robot, en contra del fallo catastrófico del filtro, que imposibilita el uso del sistema en cualquier modo controlado.

En el modelo, y para el control del ángulo del eje de la esfera (despreciando como se dijo ya, el *yaw*), además, se considera el ángulo ϕ_{esf} para la esfera, y un ángulo β_{pend} para el péndulo, donde β_{pend} se expresa como uno de los ángulos de rotación entre la esfera y el péndulo (véase la figura 3.2). Pero en el sistema físico, se tiene una IMU en la placa de componente electrónicos y otra IMU acoplada al péndulo. Y como las IMU devuelven orientación con respecto a la tierra, aunque se puede conocer ϕ_{esf} directamente con la IMU fija a la placa, no se puede conocer β_{pend}, solamente la suma $\phi_{esf} + \beta_{pend}$ que lee la IMU del péndulo. Se puede reconstruir β_{pend} restando a la lectura de la IMU del péndulo, la lectura de la IMU de la placa. Lo mismo puede hacerse para hallar las derivadas, tomando los datos filtrados y procesados previamente.

5.5.2 Implementación de los algoritmos

Como se ha usado ROS como plataforma de desarrollo del software del proyecto, las opciones de lenguajes de programación se reducen a C++, Python y Lisp. Los algoritmos de procesamiento de las IMU con un filtro DCM se implementaron en C++, pero el resto de los programas de filtrado, con Kalman, y los sistemas de control, interfaz, comunicación y procesamiento se implementaron en Python, por

la relativa facilidad y rapidez de escritura y de prueba de cambios en el código, al no necesitar compilarse. El código se guarda bajo un sistema de versionado (Git) y en la "nube" (GitHub) para asegurar su integridad y permanencia ante posibles fallos en el sistema de archivos, que pueden ocurrir ante la exposición constante al ruido eléctrico introducido por los motores, o por fallas varias.

Todos los lazos de control y procesamiento se ejecutan a una frecuencia de 10 Hz, que fue cercana a la máxima posible antes de que la carga de procesamiento fuerce a una reducción de frecuencia, usando ROS como sistema de transmisión de mensajes. Es teóricamente posible implementar los bucles de control en algún lenguaje de menor nivel a una frecuencia considerablemente superior, pero se ha decidido que esto no es algo práctico ante la necesidad, en la fase de desarrollo, de cambios constantes en el código, y el tiempo de desarrollo adicional necesario.

5.5.3 Sobre los algoritmos de control

El algoritmo de control por realimentación de estados se deriva con base al modelo y no puede "ajustarse" por su naturaleza relativamente intrincada. Sin embargo, todos los algoritmos de control PID implementan una interfaz de ajuste en tiempo real de sus parámetros, y monitoreo de sus contribuciones parciales.

5.6 Resultados

5.6.1 Efectividad de los filtros

Si bien el efecto es relativamente modesto, es necesario porque en las derivadas superiores el ruido es tan grande que inutiliza el controlador. En la figura 5.3 se muestran diez segundos de lecturas del ángulo de *roll* de la IMU acoplada a la placa de componentes electrónicos, tal como se lee de la IMU, y bajo la aplicación del filtro Kalman. Este es el único ángulo útil que puede obtenerse de esta IMU en particular, puesto que, por disposición mecánica, los ángulos de *pitch* son iguales en la IMU de la placa y en la IMU del péndulo.

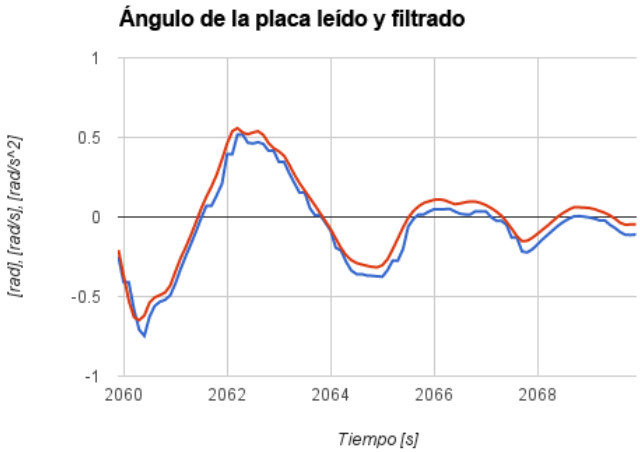


Figura 5.3 *Roll* de la placa, filtrado y sin filtrar, por 10 s.

Una aplicación del filtro Kalman para observar la primera derivada del ángulo obtiene la velocidad angular de tal ángulo. Como se había observado antes, los datos de los codificadores de los motores, aunque podrían ayudar en la tarea de obtener también estas velocidades angulares, se usan solamente para el control individual de la velocidad de cada motor. Los controladores de más alto nivel dependen solamente de los datos de las IMU. En la figura 5.4 se muestra el ángulo de *roll* filtrado de la figura 5.3 y la primera derivada observada.

Figura 5.4 *Roll* de la placa, filtrado y su primer derivada, por 10 s.

En la figura 5.5 se muestra, para el mismo caso que las figuras 5.3 y 5.4, la aceleración observada. La magnitud del ruido aparente es considerable, y las versiones del controlador que le usan mostraron un comportamiento claramente deficiente.

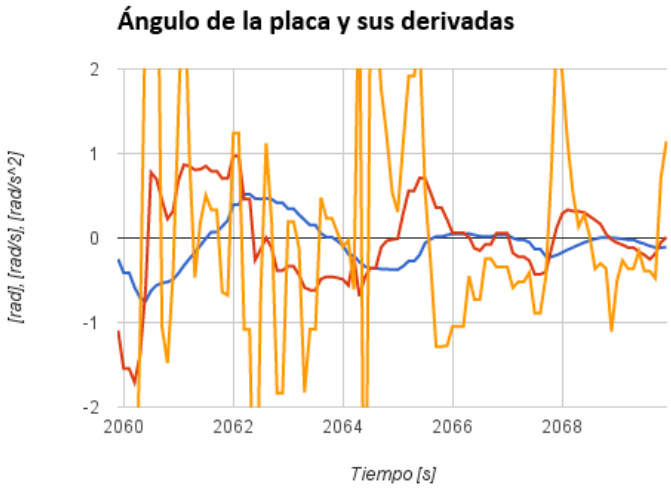


Figura 5.5 *Roll* de la placa, filtrado y sus primeras dos derivadas, por 10 s.

5.6.2 Control del ángulo del eje

El sistema usado (de lazo abierto) para tratar de seguir un movimiento rectilíneo en el modo de operación manual es propenso a virar en exceso hacia una u otra dirección, provocando eventualmente que la esfera "caiga" apoyándose en la zona cercana al extremo de uno de sus ejes. La magnitud de las desviaciones de un ángulo cero es considerablemente superior que en los demás escenarios. En la figura 5.6 se muestran cinco segundos de movimiento deseado como rectilíneo y las desviaciones leídas por la IMU del ángulo del eje diametral.

Cuando se usa el control PID ajustado de forma manual, el comportamiento del sistema es relativamente bueno, y mejora a velocidades más altas (pero no demasiado). En particular, el error permanente es casi nulo, por la contribución del término integral. Sin embargo, el sistema es aún propicio a que, en ocasiones, el viraje sea tan alto como para causar una desviación apreciable del movimiento rectilíneo, especialmente a bajas velocidades, donde el controlador no parece actuar de modo muy diferente del caso manual. Una diferencia notable con respecto

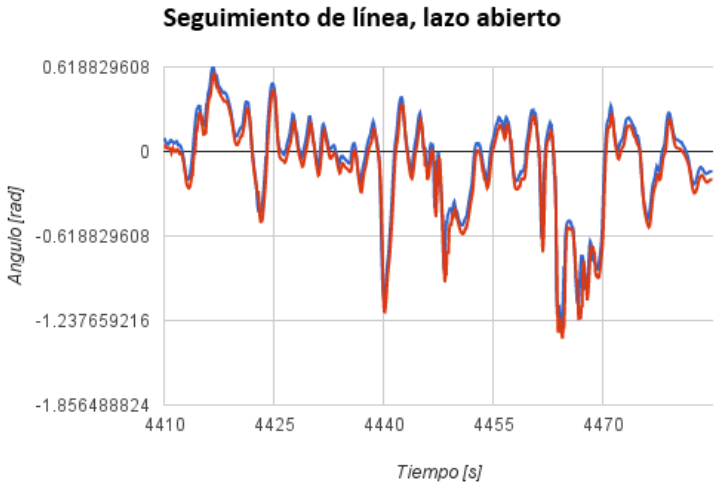


Figura 5.6 *Roll* del eje diametral, sin control, al moverse la esfera hacia "delante".

al caso en lazo abierto y con el control de realimentación de estados es que el comportamiento del ángulo de viraje de la esfera es apreciablemente más suave. En la figura 5.7 se muestran cerca de cinco segundos de operación con el controlador PID.

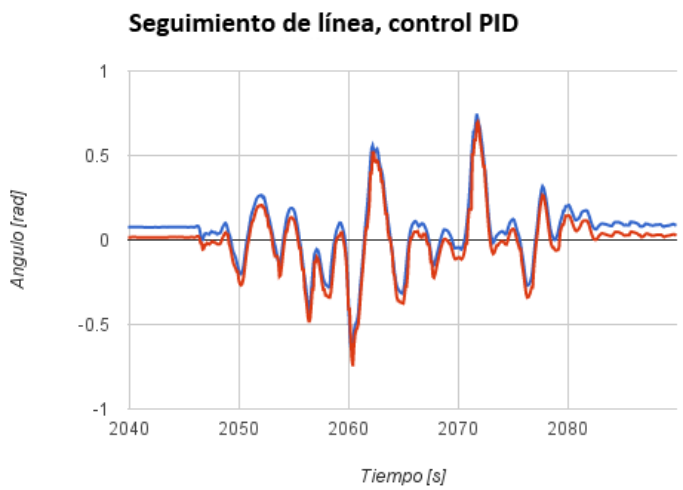


Figura 5.7 *Roll* del eje diametral, con control PID, al moverse la esfera hacia "delante".

El control en realimentación de estados parece superior al controlador PID a velocidades bajas. A velocidades más altas, el segundo estabiliza mejor el ángulo de viraje de la esfera. La magnitud máxima del error del controlador por realimentación de estados es inferior a la análoga en el caso PID. Como no se incluye un término integral, el error permanente no es nulo.

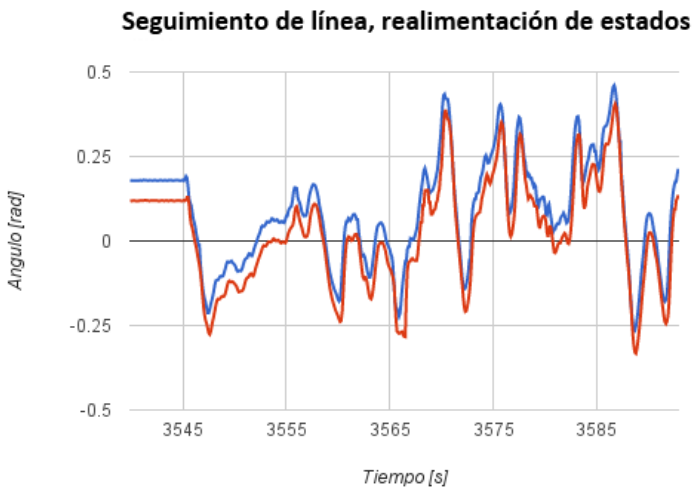


Figura 5.8 *Roll* del eje diametral, con control por realimentación de estados, al moverse la esfera hacia "delante".

Capítulo 6

Conclusiones y trabajo a futuro

6.1 Conclusiones

Los objetivos particulares presentados al inicio de este trabajo fueron:

- Proponer una configuración mecánica para el prototipo de robot esférico con base en el análisis de los sistemas de locomoción existentes.
- Obtener el modelo matemático de la configuración propuesta para generar simulaciones que permitan comprender el comportamiento del sistema y obtener algunos parámetros auxiliares para el diseño del prototipo.
- Realizar el diseño mecánico y electrónico a detalle, la manufactura y el ensamble del prototipo.
- Diseñar sistemas de control de movimiento del robot y realizar pruebas sobre el prototipo.

Puede concluirse, en relación con el objetivo, que sí es posible una configuración mecánica para el prototipo de robot esférico, como se probó con el diseño, construcción y prueba de prototipo presentados en el trabajo.

El modelado matemático, aunque complejo, funciona en el sentido de que las trayectorias observadas son realistas y reflejan, aunque sea de forma burda, el comportamiento del sistema real, y ha servido para brindar una pauta de dimensionamiento de algunos de los componentes del robot, y también para diseñar una ley de control que se ha probado con algo de éxito, ayudando a mantener

el desplazamiento rectilíneo del robot pero incapaz de mantener tal por más de algunos segundos de desplazamiento, por lo que se necesitan correcciones. Además, el método de modelado propuesto es procedural y perfectamente extensible a otros sistemas o robots de eslabones rígidos, incluso con lazos presentes, modelables por medio de ecuaciones de restricción adicionales. Un objetivo importante de este trabajo fue en algún momento aprender a modelar de forma lagrangiana sistemas mecánicos clásicos en tres dimensiones, y este objetivo personal de los autores se ha cumplido.

Se ha diseñado el sistema mecánico y electrónico a detalle, culminando en la manufactura y el ensamble del robot con pocos cambios de diseño. En el proceso, se ha aprendido una vasta cantidad de temas sobre manufactura, electrónica, control y programación, que en algún momento fueron necesarios para el robot.

Finalmente, con base en el modelo del robot, fue posible diseñar un sistema de control para una versión linealizada del robot, restringiendo algunos de sus grados de libertad, y este control, en conjunto con salidas PWM directamente a los motores, y controles PID para variables de interés, fueron probados generando datos de desempeño.

6.2 Trabajo a futuro

Entre otras cosas, varias de las limitaciones y carencias encontradas en el diseño del robot han sido:

- El diámetro de la esfera es considerable, y sistemas más versátiles podrían beneficiarse de una esfera de menor diámetro. La reducción del diámetro, sin embargo, requeriría de considerables cambios en el diseño de la mecánica del robot.
- Un sensor de la orientación externa de la esfera, acoplado a ésta y quizá comunicada por medio de un anillo de deslizamiento, permitiría el control del ángulo de la esfera en la dirección del avance, o bien, algún sensor de la diferencia absoluta de ángulos entre la placa y la esfera, al rotar el eje, podría servir indirectamente para el mismo propósito.
- Los volantes, por restricciones de tiempo e insumos, se colocaron en el robot pero no se probaron en conjunto.

- Gran parte del código puede someterse a optimizaciones, por ejemplo, usando C o C++ en lugar de Python.
- Un botón de paro de emergencia "físico" podría incrementar la seguridad del sistema ante fallas catastróficas en la comunicación.
- Una versión más moderna del *kernel* de Linux podría hacer prescindible el *router* en el cuerpo del robot.
- Debe diseñarse la PCB central para la conexión de los componentes que no se usaron en la prueba de control sin los volantes. De la misma forma, habría que extender el protocolo de comunicación. El código para el movimiento de los volantes está ya escrito.
- Un sistema electrónico de apagado y encendido, podría permitir que el sistema se apague por completo desde algún controlador externo, como con el control de videojuegos o un botón de encendido y apagado remoto.
- Podría ser conveniente probar esquemas de control mas sofisticados para mejorar la operación y eventualmente conceder autonomía al sistema, en conjunto con sistemas de visión o reconocimiento.

Apéndice A

Especificaciones

Detalles de hojas técnicas y de aplicación en el proyecto de los componentes y dispositivos utilizados.

A.1 Motores y componentes mecánicos

A.1.1 Pédulo

Para el movimiento de los dos grados de libertad del péndulo se utilizaron dos motores de corriente directa (CD) con reducción planetaria, una transmisión mediante poleas y bandas dentadas, y un juego de tres engranes cónicos.

Motores

Se utilizaron dos motores AM9015 con reducción planetaria PG71 de AndyMark, adquiridos a través de la página de Internet `www.andymark.com`.

Figura A.1 Motor AM9015 con reducción PG71.

Tabla A.1 Características del motor AM9015 con reducción PG71.

Peso	0.7575 kg
Voltaje nominal	12 V
Velocidad sin carga	216 rpm
Corriente sin carga	1.8 A
Corriente a rotor bloqueado	63.8 A
Par a rotor bloqueado	30.388 N·m
Reducción	71:1

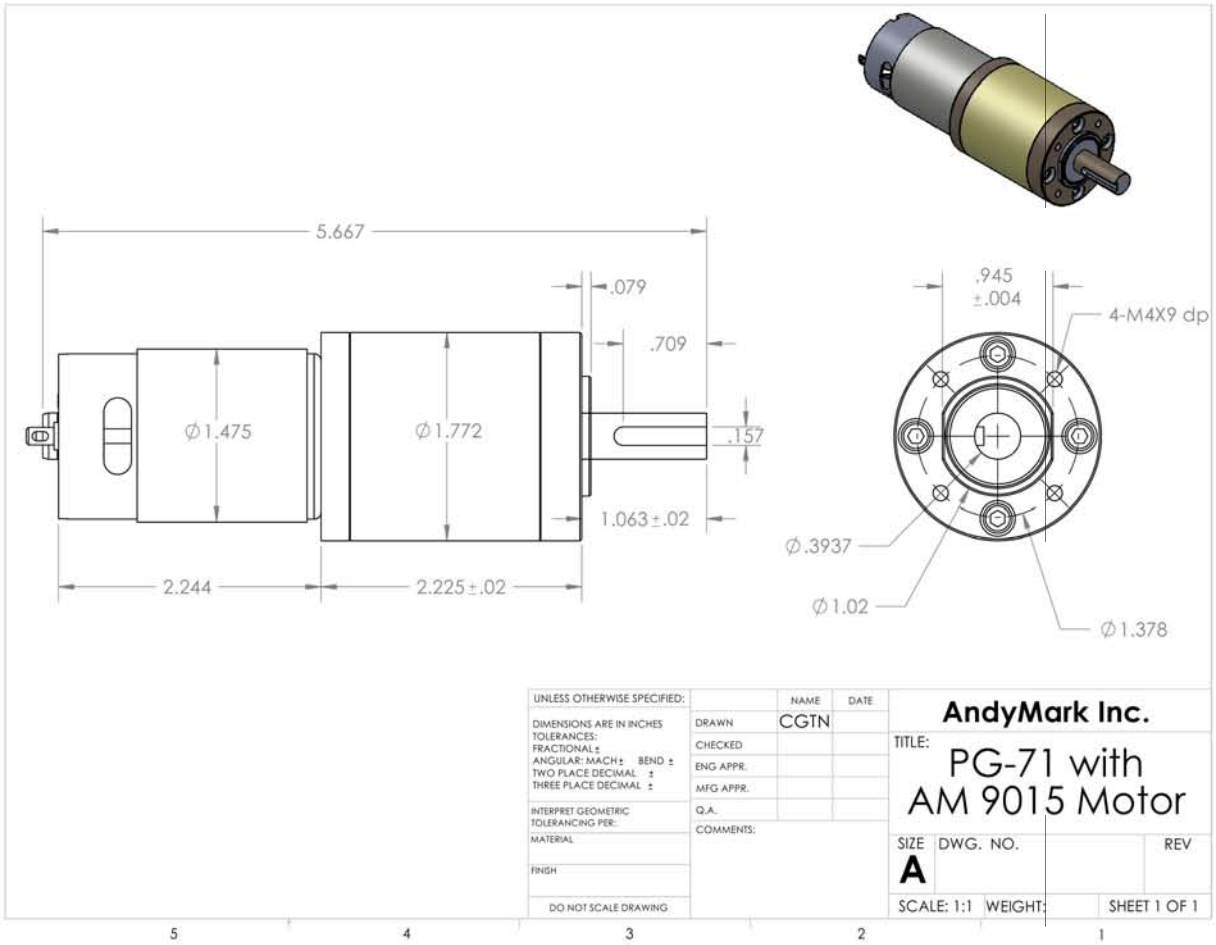


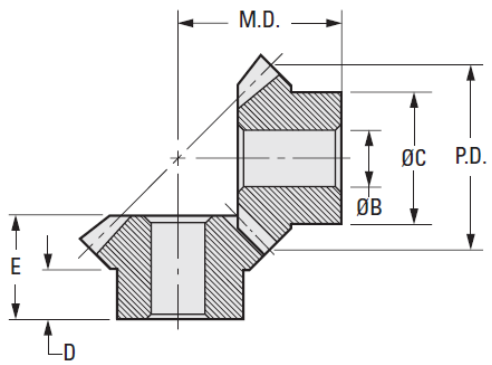
Figura A.2 Plano mecánico del motor AM9015 con reducción PG71.

Engranes

Se utilizaron tres engranes cónicos de acero al carbón, con número de catálogo A 1C 4-Y20020 de Stock Drive Products/Sterling Instruments (SDP/SI), adquiridos a través de la página de Internet `www.sdp-si.com`.

Figura A.3 Engranes cónicos A 1C 4-Y20020.

Tabla A.2 Características del engrane cónico A 1C 4-Y20020.



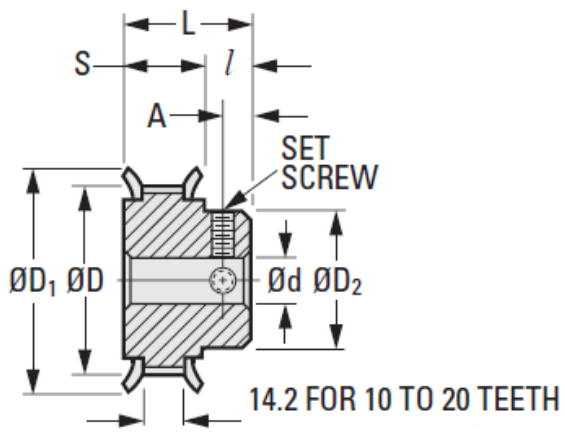
Ángulo de presión	20 °
Paso diametral	20 dientes/in
Número de dientes	20
Diámetro primitivo (P.D.)	1 in
Diámetro interior (B)	0.375 in
Ancho de cara	0.234 in
Longitud (E)	0.8125 in
Diámetro de maza (C)	0.75 in
Proyección de maza (D)	0.5 in
Distancia de montaje (M.D.)	1.125 in

Poleas

Se utilizaron cuatro poleas dentadas de aleación de aluminio con acabado anodizado claro, con número de catálogo A 6A 4M12DF12510 de SDP/SI.

Figura A.4 Polea A 6A 4M12DF12510.

Tabla A.3 Características de la polea A 6A 4M12DF12510.



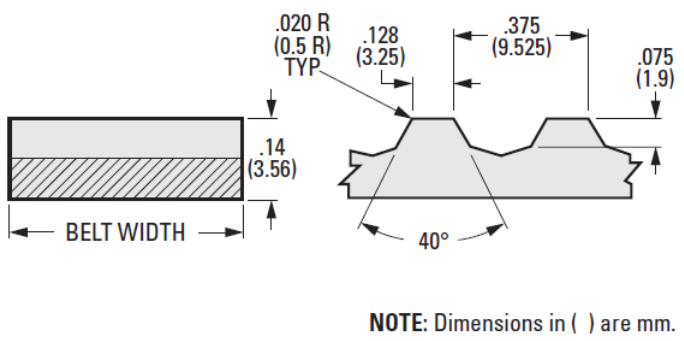
Número de dientes	12
Diámetro de paso	36.4 mm
Diámetro exterior (D)	35.6 mm
Diámetro de ceja (D_1)	42 mm
Diámetro interior (d)	10 mm
Ancho de polea (S)	18.3 mm
Longitud total (L)	30.2 mm
Diámetro de maza(D_2)	28.6 mm
Proyección de maza (l)	11.9 mm
Distancia al opresor (A)	5 mm
Tornillo opresor	M5

Bandas

Se utilizaron dos bandas dentadas de neopreno recubierto de nylon y con refuerzo de fibra de vidrio, con número de catálogo A 6R 4M036125 de SDP/SI.

Figura A.5 Banda A 6R 4M036125.

Tabla A.4 Características de la banda A 6R 4M036125.



Número de dientes	36
Paso	9.525 mm
Longitud total (línea primitiva)	342.9 mm
Ancho de la banda	12.5 mm
Fuerza de ruptura (por mm)	175 N/mm
Temperatura de operación	-34 a 85 °C

A.1.2 Giroscopios de control de momento

Para los giroscopios de control de momento se utilizaron un motor de corriente directa con reducción planetaria, dos motores trifásicos de corriente directa (motores *brushless*) y un juego de tres engranes cónicos, uno conductor y dos conducidos, con relación de transmisión de 1:2.

Motor de CD

Se utilizó un motor RS775 con reducción planetaria PG71 de AndyMark.

Figura A.6 Motor RS775 con reducción PG71.

Tabla A.5 Características del motor RS775 con reducción PG71.

Peso	0.8482 kg
Voltaje nominal	12 V
Velocidad sin carga	75 rpm
Corriente sin carga	0.6 A
Corriente a rotor bloqueado	22 A
Par a rotor bloqueado	22.514 N·m
Reducción	71:1

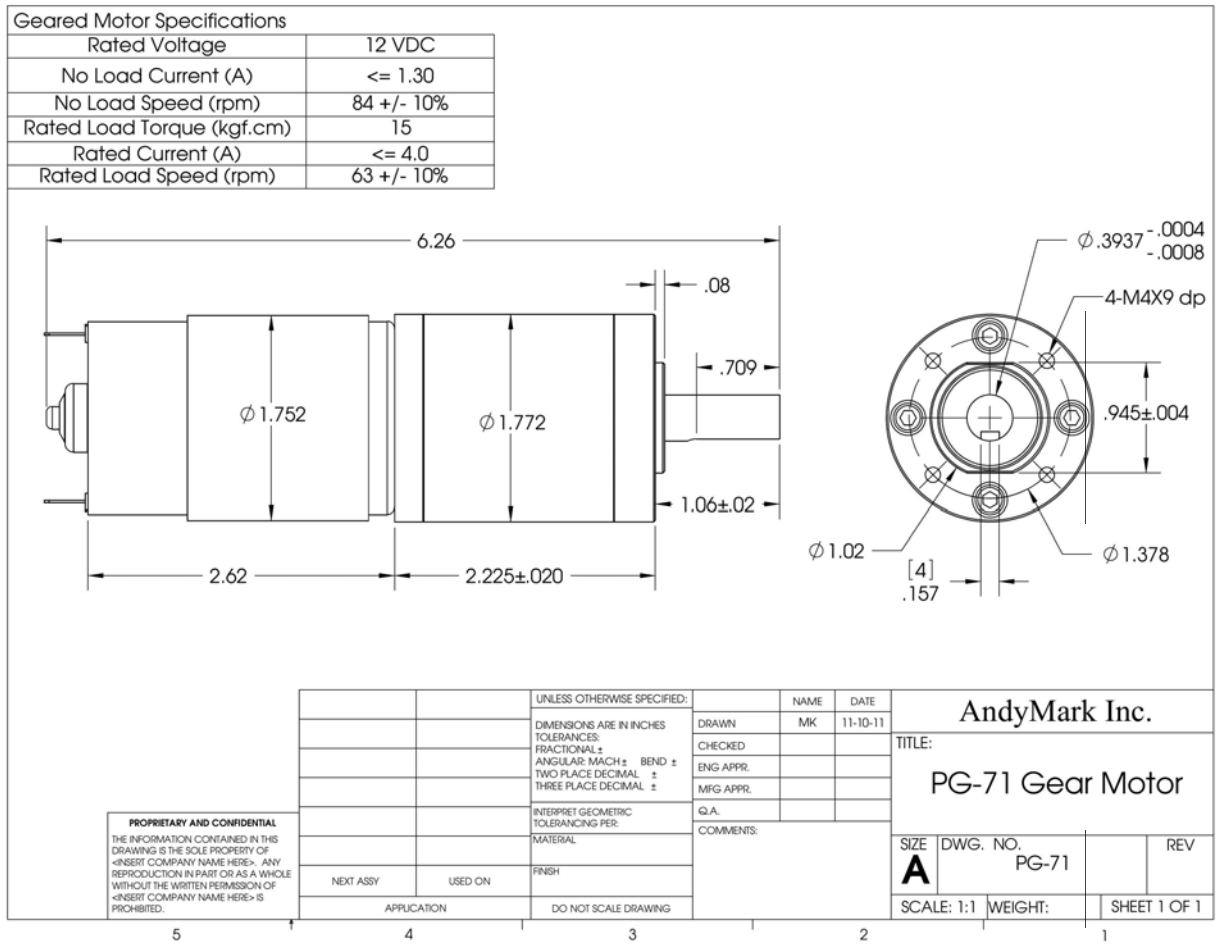


Figura A.7 Plano mecánico del motor RS775 con reducción PG71.

Motores *brushless*

Se utilizaron dos motores NTM Prop Drive Series 35-30A 1400kv, de HobbyKing, adquiridos a través de la página de Internet `www.hobbyking.com`.

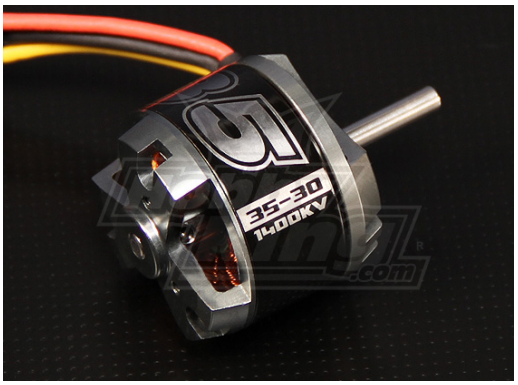
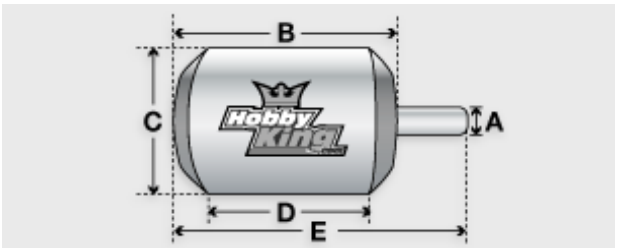


Figura A.8 Motor NTM Prop Drive Series 35-30A 1400kv.

Tabla A.6 Características del motor NTM Prop Drive Series 35-30A 1400kv.



Peso	88.3 g
Constante de velocidad (kv)	1400 rpm/V
Corriente máxima	37 A
Alimentación (batería LiPo)	3S ó 4S
Potencia máxima (a 15 V)	560 W
Diámetro eje (A)	4 mm
Longitud carcasa (B)	32 mm
Diámetro carcasa (C)	35 mm
Longitud estator (D)	16 mm
Longitud total (E)	53 mm

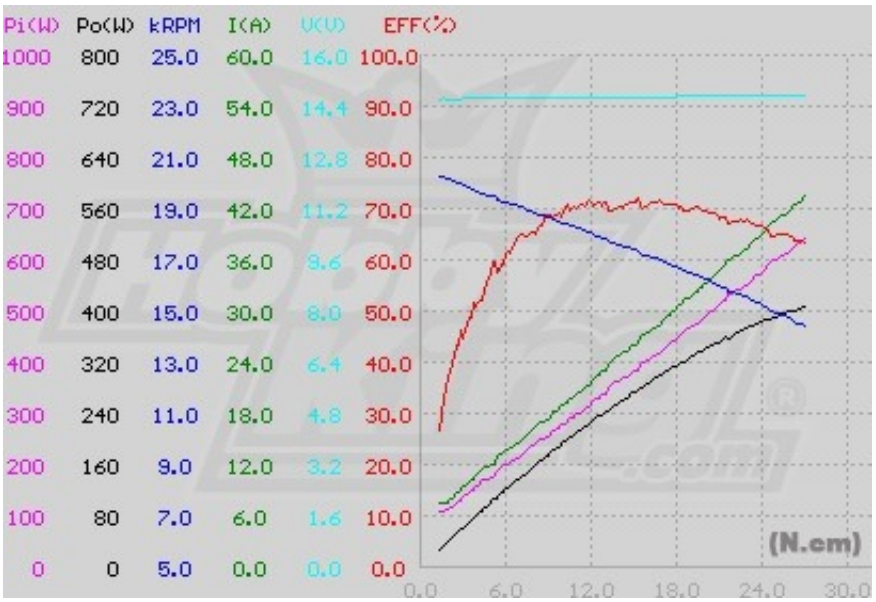


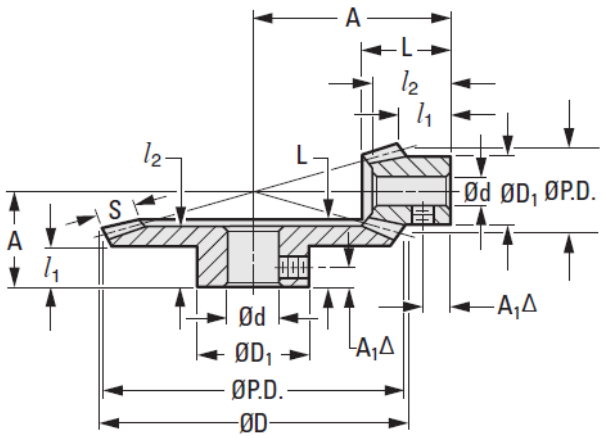
Figura A.9 Curvas características del motor NTM Prop Drive Series 35-30A 1400kv a 15 V.

Engranes

Se utilizó un juego de tres engranes cónicos de acero al carbón, formado por un engrane con número de catálogo A 1C33MYK10040S y dos engranes con número de catálogo A 1C33MYK10020S, de SDP/SI.

Figura A.10 Engranes A 1C33MYK10020S y A 1C33MYK10040S.

Tabla A.7 Características de los engranes A 1C33MYK10020S y A 1C33MYK10040S.



	A 1C33MYK10020S	A 1C33MYK10040S
Módulo	1 mm	1 mm
Número de dientes	20	40
Diámetro primitivo (P.D)	20 mm	40 mm
Diámetro exterior (D)	21.79 mm	40.89 mm
Diámetro interior (d)	8 mm	10 mm
Ancho de cara (S)	5.7 mm	5.7 mm
Longitud total (L)	15.03 mm	15.04 mm
Diámetro de maza (D_1)	16 mm	25 mm
Proyección de maza (l_1)	8.6 mm	8 mm
Distancia de montaje (A)	29.6 mm	21.8 mm
Longitud interior (l_2)	14 mm	13 mm
Distancia al opresor (A_1)	4 mm	4 mm
Tornillo opresor	M4	M5

A.2 Componentes electrónicos

A.2.1 Electrónica de potencia y energía

Puente H

Para la etapa de potencia y el control de los motores de corriente directa se utilizaron tres puentes H Talon SR, de Cross the Road Electronics.

Figura A.11 Puente H Talon SR.

Tabla A.8 Características del puente H Talon SR.

Voltaje de entrada	6 a 28 V
Corriente continua máxima	60 A
Corriente pico máxima (por 2 s)	100 A
Señal PWM de entrada	1 a 2 ms a 330 Hz
Resolución de entrada	10 bit
Resolución de salida	10 bit
Frecuencia de conmutación	15 kHz
Zona muerta	4 %

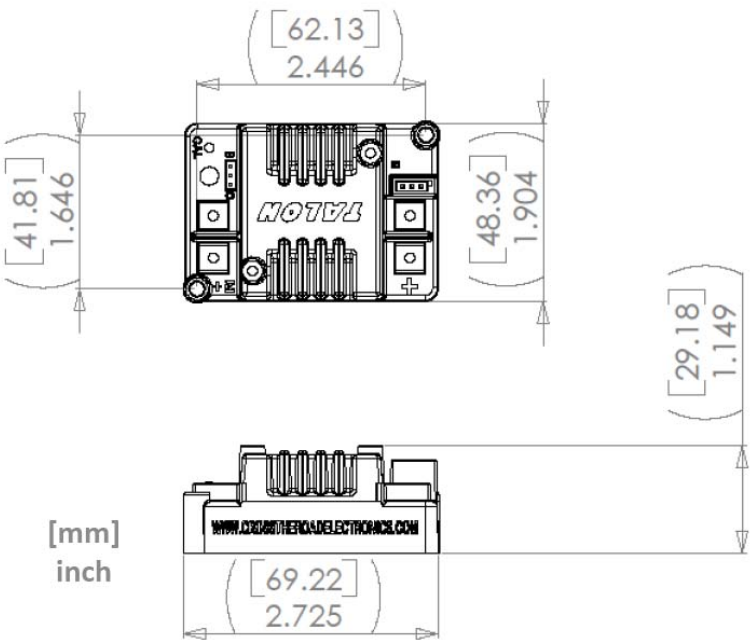


Figura A.12 Dimensiones del puente H Talon SR.

ESC

Para la etapa de potencia y el control de los motores *brushless* se utilizaron dos controladores de velocidad Dlux 55A SBEC de Turnigy.

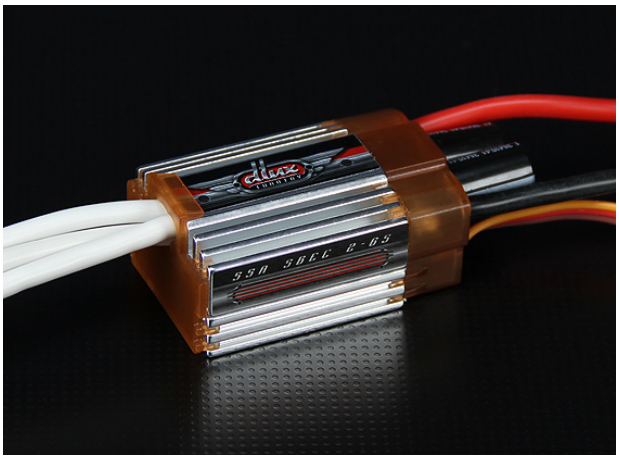


Figura A.13 ESC Dlux 55A SBEC.

Tabla A.9 Características del ESC Dlux 55A SBEC.

Alimentación (batería LiPo)	2S a 6S
Corriente continua máxima	55 A
Corriente pico máxima	67 A
Peso	81 g
Velocidad máxima	200,000 rpm
Voltaje de corte	2.5 a 3.5 V
Avance	Adelante/reversa

Baterías

Para la alimentación de los componentes del robot se utilizaron dos baterías de polímero de litio Hardcase Pack LiPo 4S 5000 mA·h de Turnigy, adquiridas a través de HobbyKing.

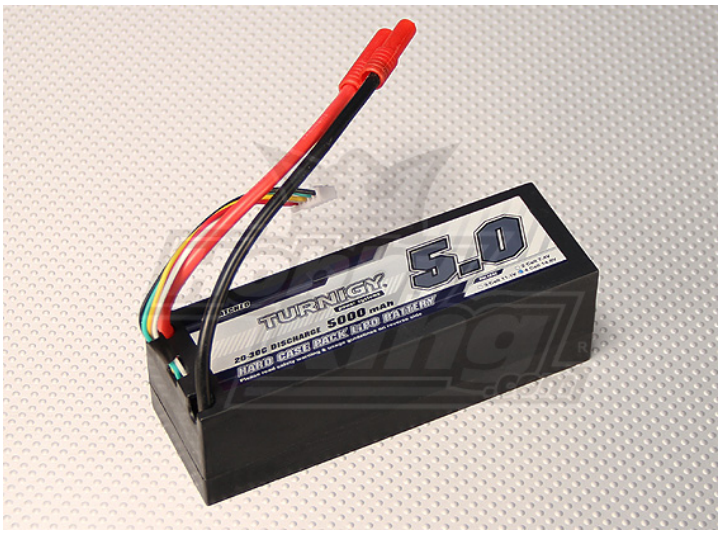


Figura A.14 Batería Hardcase Pack LiPo 4S 5000 mA·h.

Tabla A.10 Características de la batería Hardcase Pack LiPo 4S 5000 mA·h.

Capacidad (carga)	5000 mA·h
Configuración de celdas	4S1P
Constante de descarga	20C
Descarga máxima (20 s)	30C
Tasa de carga máxima	5C
Peso	528 g
Dimensiones	139 × 45 × 44 mm
Conector de carga	JST-XH
Conector de descarga	Conector bala de 4 mm

A.2.2 Sensores

Codificadores

Para la lectura de la posición angular cada uno de los motores de corriente directa se utilizaron tres codificadores de efecto Hall AS5043, fabricados por AMS. Estos codificadores miden la rotación de un imán de neodimio que debe encontrarse a una distancia máxima de 3 mm. El fabricante recomienda imanes de 6 mm de diámetro por 3 mm de altura.

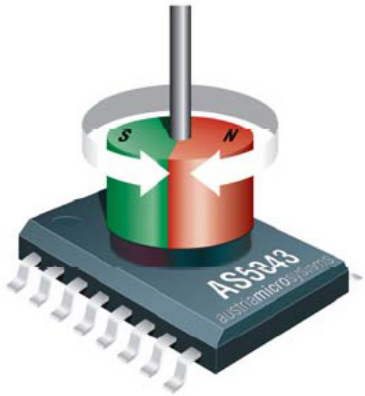


Figura A.15 Arreglo del codificador AS5043 e imán.

Tabla A.11 Características del codificador AS5043.

Resolución	10 bit
Salida	Digital(por interfaz) o analógica
Interfaz de salida	SSI
Velocidad máxima	30,000 rpm
Voltaje de alimentación	5 ó 3.3 V
Temperatura de operación	-40 a 125 °C
Encapsulado	SSOP-16
Densidad de flujo magnético (imán)	45 a 75 mT

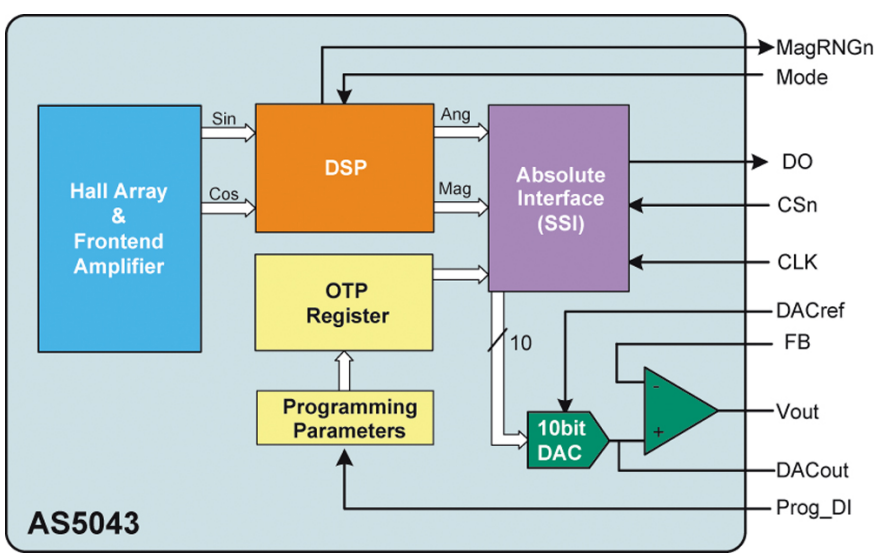


Figura A.16 Diagrama de bloques del codificador AS5043.

IMU

Para la medición de los ángulos del péndulo y de la esfera se utilizaron dos IMU 9DOF SensorStick, versión 13 (con número de parte SEN-10724) de SparkFun. Esta IMU está formada por un acelerómetro ADXL345, un giroscopio ITG-3200 y un magnetómetro HMC5883L, cada uno de tres grados de libertad.

Figura A.17 IMU 9DOF Sensor Stick, versión 13.

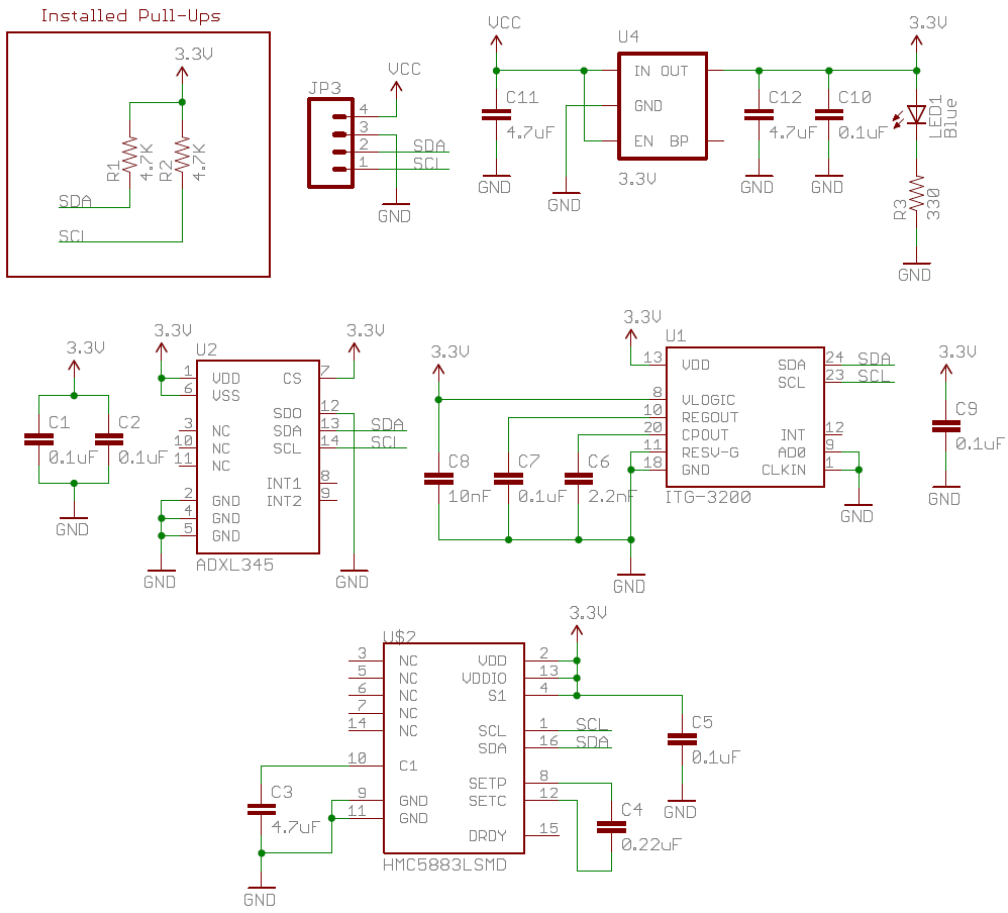


Figura A.18 Diagrama esquemático de la IMU 9DOF Sensor Stick, versión 13.

Tabla A.12 Características del acelerómetro ADXL345.

Resolución	10 a 13 bit
Rangos de medición	$\pm 2g$, $\pm 4g$, $\pm 8g$, $\pm 16g$
Voltaje de alimentación (V_s)	2 a 3.6 V
Corriente máxima de operación	145 µA
Voltaje de I/O	1.7 V a V_s
Interfaces digitales	SPI e I^2C
Temperatura de operación	-40 a 85 °C
Aceleración de choque máxima	$10{,}000g$
Encapsulado	LGA ($3 \times 5 \times 1$ mm)
Frecuencia de muestreo	6.25 a 3200 Hz
Peso	20 mg

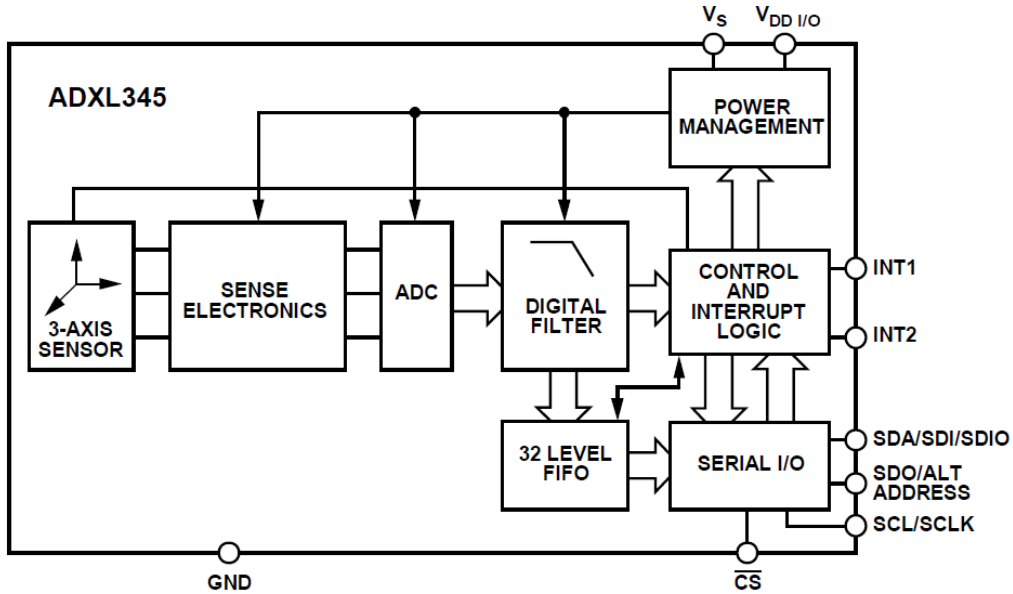


Figura A.19 Diagrama de bloques del acelerómetro ADXL345.

Tabla A.13 Características del giroscopio ITG3200.

Voltaje de alimentación	2.1 a 3.6 V
Corriente de operación	6.4 mA
Rango de medición	±2000 °/s
Resolución	16 bit
Interfaz digital	I^2C (400 kHz)
Frecuencia de muestreo	1 a 8 kHz
Temperatura de operación	-40 a 85 °C
Encapsulado	QFN (4 × 4 × 0.9 mm)

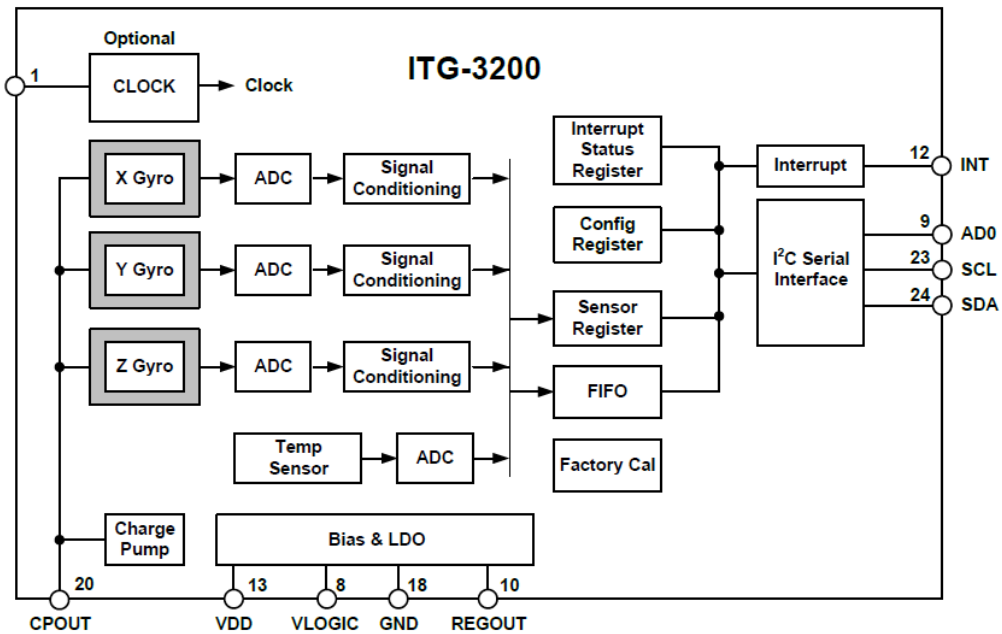


Figura A.20 Diagrama de bloques del giroscopio ITG3200.

Tabla A.14 Características del magnetómetro HMC5883L.

Voltaje de alimentación	2.16 a 3.6 V
Corriente de operación	100 µA
Densidad de flujo magnético	-8 a 8 G
Resolución	12 bit
Interfaz digital	I^2C
Temperatura de operación	-30 a 85 °C
Frecuencia de muestreo	0.75 a 160 Hz
Precisión	1 a 2 °
Encapsulado	LCC ($3 \times 3 \times 0.9$ mm)

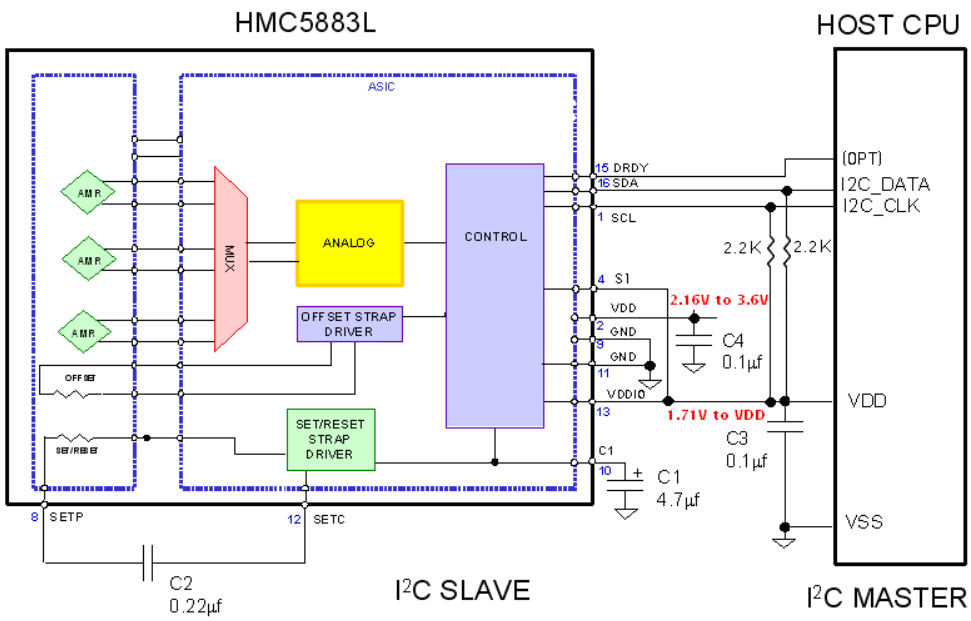


Figura A.21 Diagrama de bloques del magnetómetro HMC5883L.

A.2.3 Procesamiento y comunicación

Computadora

Para el procesamiento se utilizó la computadora BeagleBone Black, desarrollada por la fundación BeagleBoard.org.

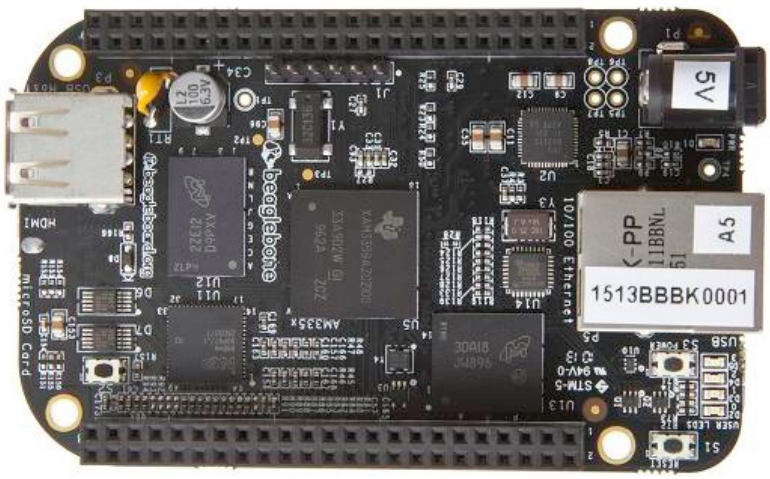


Figura A.22 Computadora BeagleBone Black.

Tabla A.15 Características de la computadora BeagleBone Black.

Procesador	Sitara ARM Cortex-A8 AM3359, 1 GHz
Memoria SDRAM	512 MB, DDR3L, 800 MHz
Memoria flash	4 GB, 8 bit, eMMC
Soporte para depuración	Puerto JTAG de 20 pines
Alimentación	5 V (por mini USB o conector Jack)
Puertos USB	USB 2.0 Host y USB 2.0 Client
Puerto serial	UART0, 3.3 V TTL (pin 6)
Puerto Ethernet	10/100, conector RJ45
Salida de vídeo	16b HDMI, 1280 × 1024, 24 Hz
Sistemas operativos	Ubuntu, Debian, Android
Pines	2 × 46
Dimensiones	3.4 × 2.1 in
Peso	39.68 g

Microcontrolador

En adición a la computadora, se utilizó para algunas tareas de procesamiento la tarjeta de desarrollo con microcontrolador Stellaris LM4F120 LaunchPad Evaluation Board, de Texas Instruments.

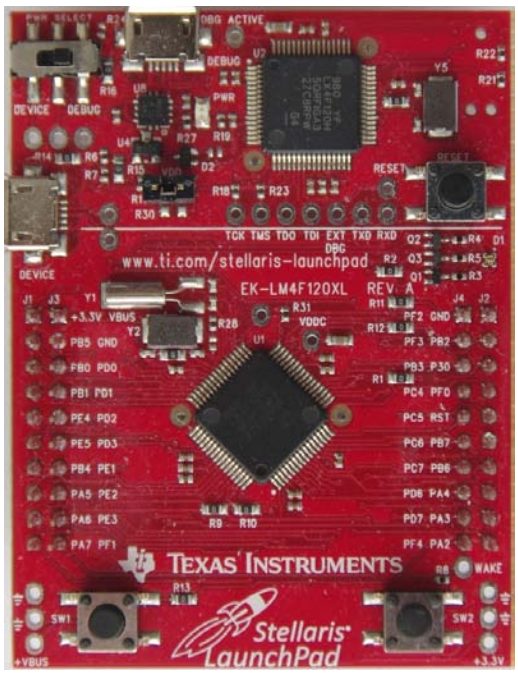


Figura A.23 Tarjeta de desarrollo Stellaris LMF4120.

Tabla A.16 Características de la tarjeta Stellaris LM4F120.

Microcontrolador	32 bit ARM cortex-M4F LM4F120H5QR
Memoria SRAM	32 kB
Memoria EEPROM	2 kB
Memoria flash	256 kB
Velocidad máxima	80 MHz
Pines GPIO	43
Módulos CCP	24
Canales ADC	12
Resolución ADC	12 bit
Puertos seriales	4 SSI/SPI, 4 I^2C, 8 UART
Alimentación	5 V, por conector micro USB

Router inalámbrico

Para la comunicación entre la computadora de tierra y el robot se utilizó un Nano Router Inalámbrico N de 150Mbps, modelo TL-WR702N de TP-LINK.

Figura A.24 *Router* inalámbrico TL-WR702N.

Tabla A.17 Características del *router* inalámbrico TL-WR702N.

Interfaz	Puertos WAN/LAN y USB, botón de *reset*
Antena	Integrada
Estándares	802.11n, 802.11g, 802.11b
Dimensiones	$57 \times 57 \times 18$ mm
Frecuencia	2.4 a 2.4835 GHz
Velocidad de señal	150 Mb/s (11n), 54 Mb/s (11g), 11 Mb/s (11b)
Modos inalámbricos	Access Point, Router, Client, Repeater, Bridge
Seguridad inalámbrica	WEP, WPA/WPA2, WPA-PSK/WPA2-PSK
Tipo WAN	IP dinámica, IP estática, PPPoE, PPTP/L2TP

Adaptador inalámbrico

En adición al *router* inalámbrico, se utilizó para la comunicación un Adaptador USB Inalámbrico de Alta Ganancia de 150Mbps, modelo TL-WN722N de TP-LINK.

Figura A.25 Adaptador inalámbrico TL-WR702N.

Tabla A.18 Características del adaptador inalámbrico TL-WR702N.

Interfaz	Puerto USB 2.0, botón QSS
Antena	Desmontable omnidireccional
Rendimiento de antena	4 dBi
Estándares	802.11n, 802.11g, 802.11b
Dimensiones	93.5 × 26 × 11 mm
Frecuencia	2.4 a 2.4835 GHz
Velocidad de señal	150 Mb/s (11n), 54 Mb/s (11g), 11 Mb/s (11b)
Modos inalámbricos	Ad-Hoc/Infraestructura
Seguridad inalámbrica	WEP, WPA-PSK/WPA2-PSK

A.3 Tarjetas impresas

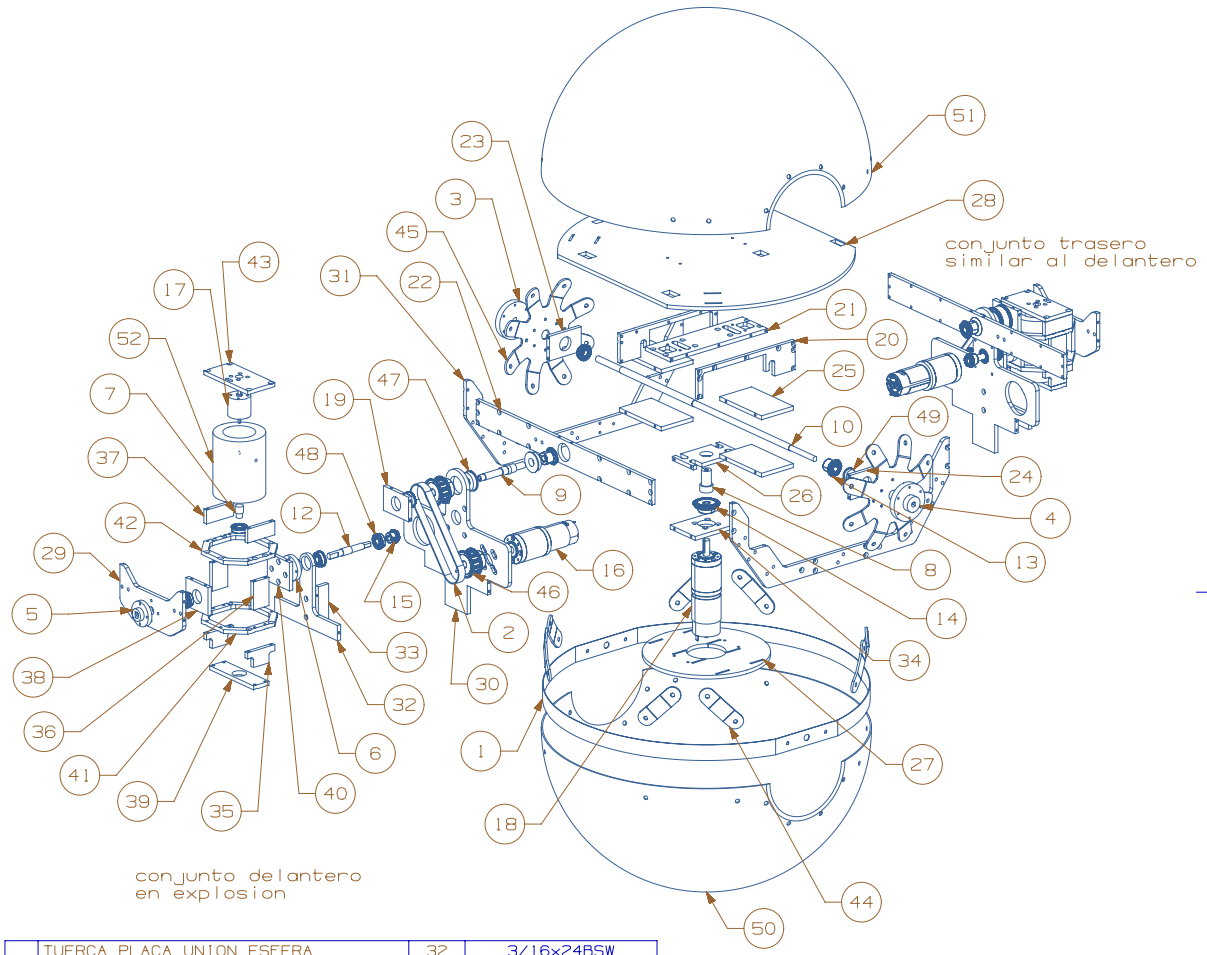
Se diseñaron para el proyecto y se usan dos tipos, un diseño de tarjeta para los codificadores magnéticos, y una tarjeta de reguladores.

Apéndice B

Planos mecánicos

B.1 Planos de ensamble

	TORNILLO_BRIDA_EJE_PRINCIPAL	12	M4x0.7x8mm c. cilindrica allen
	TORNILLO_ANILLO	4	3/16x24BSWx1/2in c. redonda
52	VOLANTE	2	MEC-043
51	SEMIESFERA_SUP	1	MEC-042
50	SEMIESFERA_INF	1	MEC-041
49	RODAMIENTO_8	1	8mm con ceja
48	RODAMIENTO_3_8	11	3/8in con ceja
47	RODAMIENTO_1_2	4	1/2in con ceja
46	POLEA	4	SDP/SI - A 6A 4M12DF12510
45	PLACA_UNION_ESFERA_LAT	2	MEC-039
44	PLACA_UNION_ESFERA	8	MEC-038
43	PLACA_SOPORTE_VOLANTE_SUP	2	MEC-035
42	PLACA_SOPORTE_VOLANTE_MED_SUP	2	MEC-034
41	PLACA_SOPORTE_VOLANTE_MED_INF	2	MEC-033
40	PLACA_SOPORTE_VOLANTE_INT	2	MEC-032
39	PLACA_SOPORTE_VOLANTE_INF	2	MEC-031
38	PLACA_SOPORTE_VOLANTE_EXT	2	MEC-030
37	PLACA_SOPORTE_VOLANTE_DEL_TRAS_SUP	4	MEC-029
36	PLACA_SOPORTE_VOLANTE_DEL_TRAS_MED	4	MEC-028
35	PLACA_SOPORTE_VOLANTE_DEL_TRAS_INF	4	MEC-027
34	PLACA_SOPORTE_MOTOR_VERT	1	MEC-026
33	PLACA_PENDULO_UNION	2	MEC-025
32	PLACA_PENDULO_MED	2	MEC-024
31	PLACA_PENDULO_LAT	2	MEC-023
30	PLACA_PENDULO_INT	2	MEC-022
29	PLACA_PENDULO_EXT	2	MEC-021
28	PLACA_MONTAJE_SUP	1	MEC-037
27	PLACA_MONTAJE_INF	1	MEC-036
26	PLACA_ENCODER_MOTOR_VERT	1	MEC-020
25	PLACA_EJE_PRINCIPAL_UNION	4	MEC-019
24	PLACA_EJE_PRINCIPAL_LAT_IZQ	1	MEC-018
23	PLACA_EJE_PRINCIPAL_LAT_DER	1	MEC-017
22	PLACA_EJE_PRINCIPAL_DEL_TRAS	2	MEC-016
21	PLACA_EJE_POLEA_SUP	1	MEC-015
20	PLACA_EJE_POLEA_LAT	2	MEC-014
19	PLACA_EJE_POLEA_DEL_TRAS	2	MEC-013
18	MOTOR_RS775_PG71	1	AndyMark - RS775 PG71
17	MOTOR_NTM35-30	2	NTM - Prop Drive 30-35
16	MOTOR_AM9015_PG71	2	AndyMark - AM9015 PG71
15	ENGRANE_VOLANTES_FOLLOWER	2	SDP/SI - A 1C33MYK10040S
14	ENGRANE_VOLANTES_DRIVER	1	SDP/SI - A 1C33MYK10020S
13	ENGRANE_PENDULO	3	SDP/SI - A 1C 4-Y20020
12	EJE_VOLANTE_INT	2	MEC-008
11	EJE_VOLANTE_EXT	2	MEC-007
10	EJE_PRINCIPAL	1	MEC-006
9	EJE_POLEA	2	MEC-005
8	EJE_EXTENSION_ENCODER	1	MEC-004
7	EJE_APOYO_VOLANTE	2	MEC-003
6	BRIDA_SOPORTE_VOLANTE_INT	2	MEC-012
5	BRIDA_SOPORTE_VOLANTE_EXT	2	MEC-011
4	BRIDA_EJE_PRINCIPAL_IZQ	1	MEC-010
3	BRIDA_EJE_PRINCIPAL_DER	1	MEC-009
2	BANDA	2	SDP/SI - A 6R 4M036125
1	ANILLO	1	MEC-040
NO	NOMBRE DE PARTE	CANT	PLANO/PARTE



	TUERCA_PLACA_UNION_ESFERA	32	3/16x24BSW
	TUERCA_ANILLO	4	10x24UNC
	TORNILLO_VOLANTE	4	M5x0.8x8mm opresor
	TORNILLO_PLACA_UNION_ESFERA	32	5/16x18UNCx1/2in c. redonda
	TORNILLO_PLACA_MONTAJE_SUP	4	M4x0.7x14mm c. cilindrica allen
	TORNILLO_PLACA_MONTAJE_INF	4	M4x0.7x10mm c. cilindrica allen
	TORNILLO_PLACA	174	M4x0.7x12mm c. avellanada allen
	TORNILLO_ENGRANE	3	M4x0.7x5mm opresor allen
	TORNILLO_ENGRANE_POLEA_BRIDA	17	M5x0.8x6mm opresor allen
NO	NOMBRE DE PARTE	CANT	PLANO/PARTE

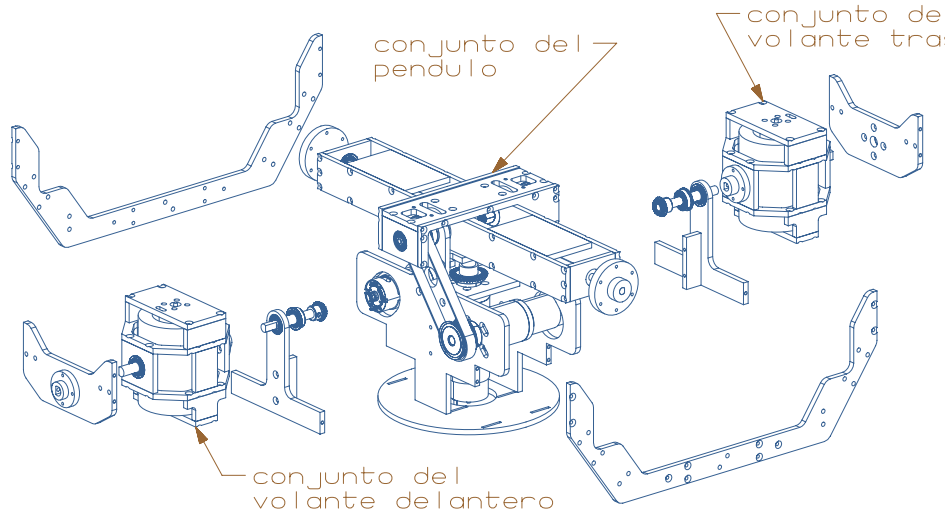
ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO
FACULTAD DE INGENIERIA

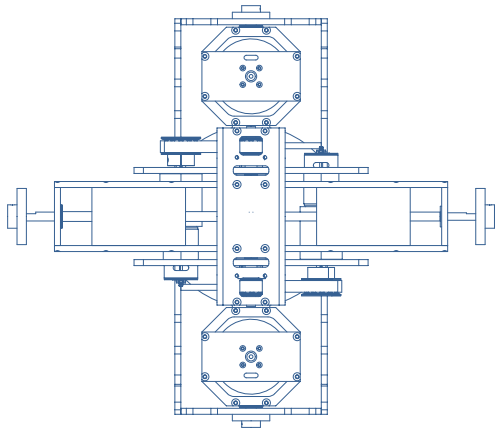
ROBOT ESFERICO

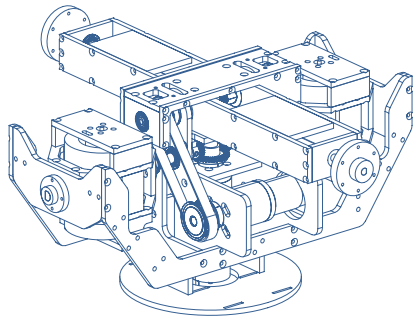
TITLE: EXPLOSION COMPLETA

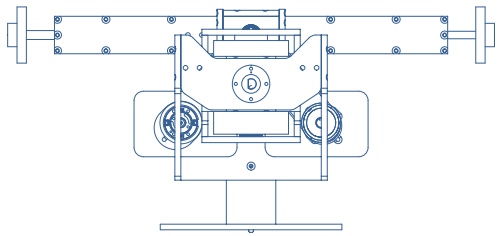
DRAWN BY	JLAS	SIZE	FILE: ensamble_completo	REV
CHECKED BY	-	A2		
APPROVED BY	-	SCALE 1:5	SHEET 1 OF 1	PLANO MEC-001

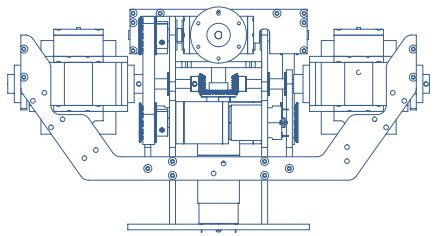


ensamble sin esfera ni
placa de montaje superior









ALL DIMENSIONS IN MM

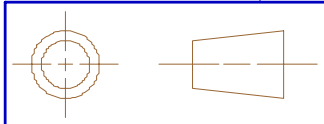
UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO
FACULTAD DE INGENIERIA

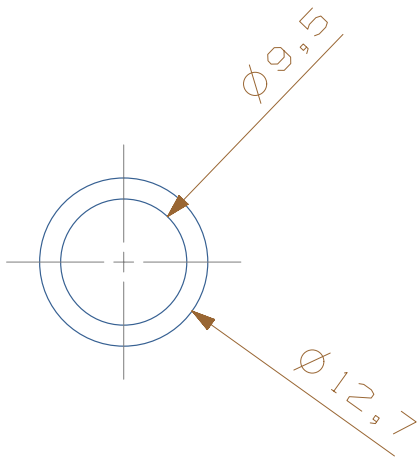
ROBOT ESFERICO

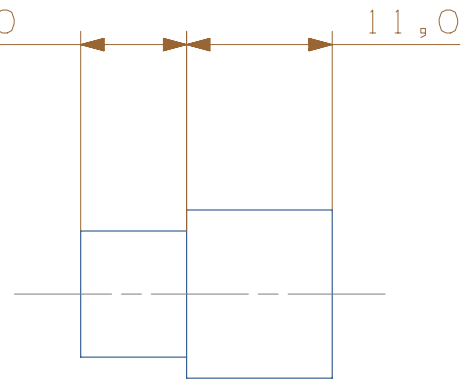
TITLE
ENSAMBLE SIN ESFERA
EXPLOSION POR CONJUNTOS

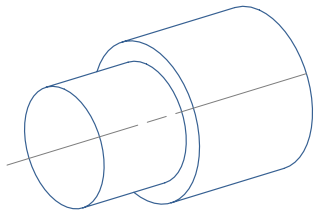
DRAWN BY	JLAS	SIZE	FILE NAME		REV
CHECKED BY	-	A3	ensamble_sin_esfera		
APPROVED BY	-	SCALE 1:5	SHEET 1 OF 1	PLANO MEC-002	

B.2 Planos de fabricación









ALL DIMENSIONS IN MM

DRAWN BY	JLAS
CHECKED BY	-
APPROVED BY	-

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

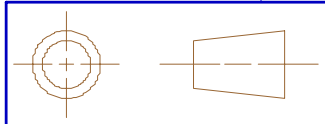
ROBOT ESFERICO

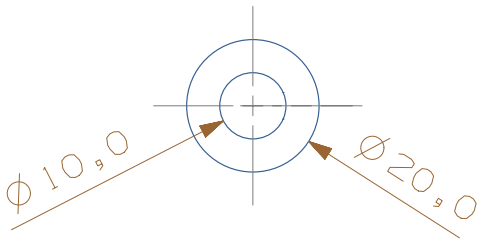
TITLE

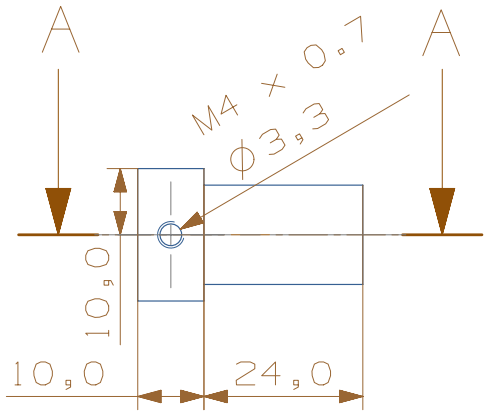
EJE APOYO VOLANTE

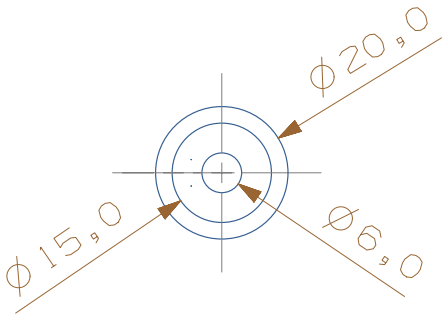
SIZE	FILE NAME	REV
A4	eje_apoyo_volante	☒

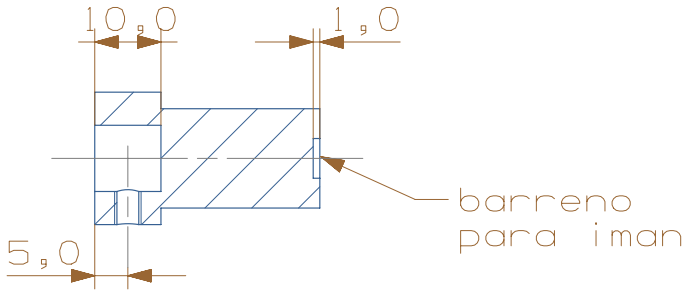
SCALE 2:1	SHEET 1 OF 1	PLANO MEC-003



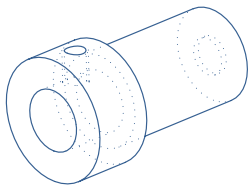








SECTION A-A



ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

ROBOT ESFERICO

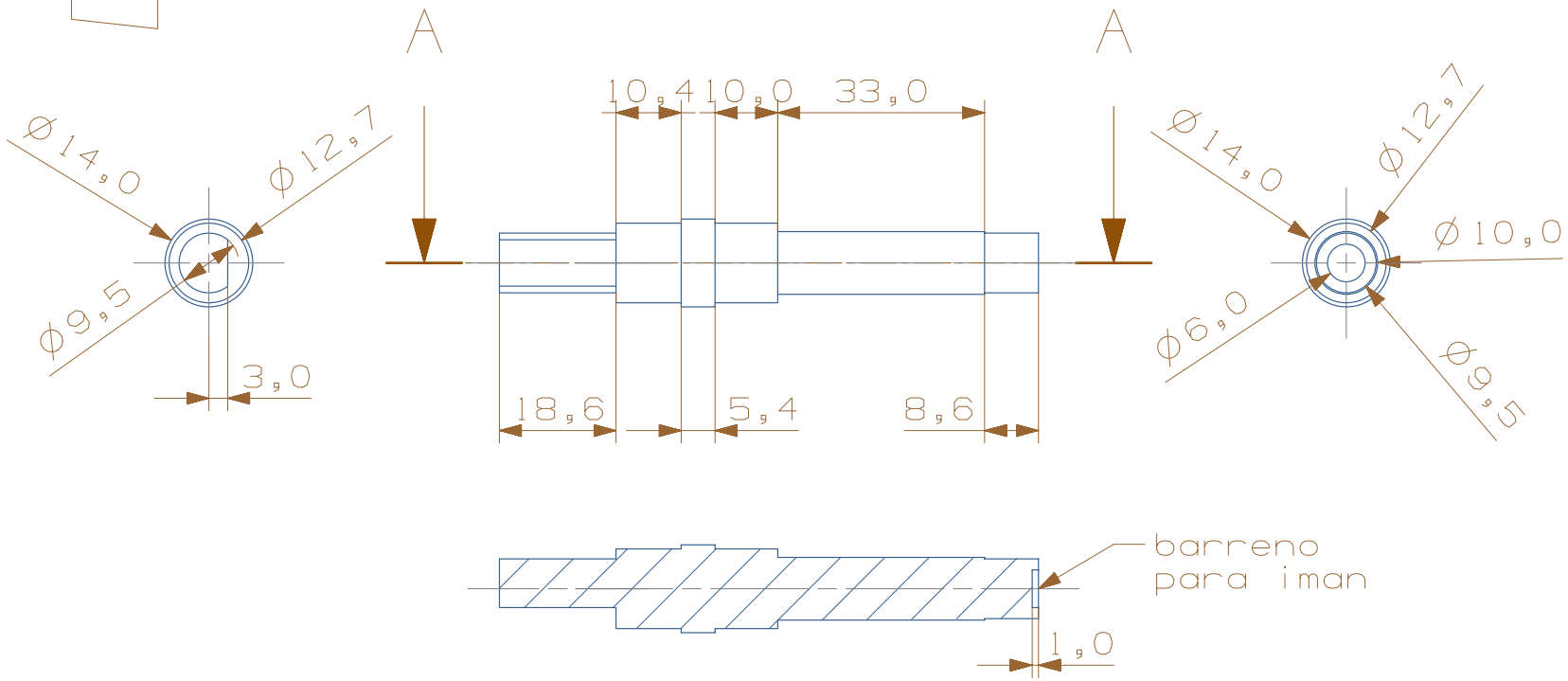
TITLE

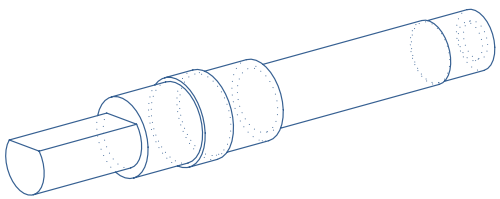
EJE EXTENSION ENCODER

DRAWN BY	JLAS
CHECKED BY	-
APPROVED BY	-

SIZE	FILE NAME	REV
A4	eje_extension_encoder	

SCALE 1:1	SHEET 1 OF 1	PLANO MEC-004





ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

ROBOT ESFERICO

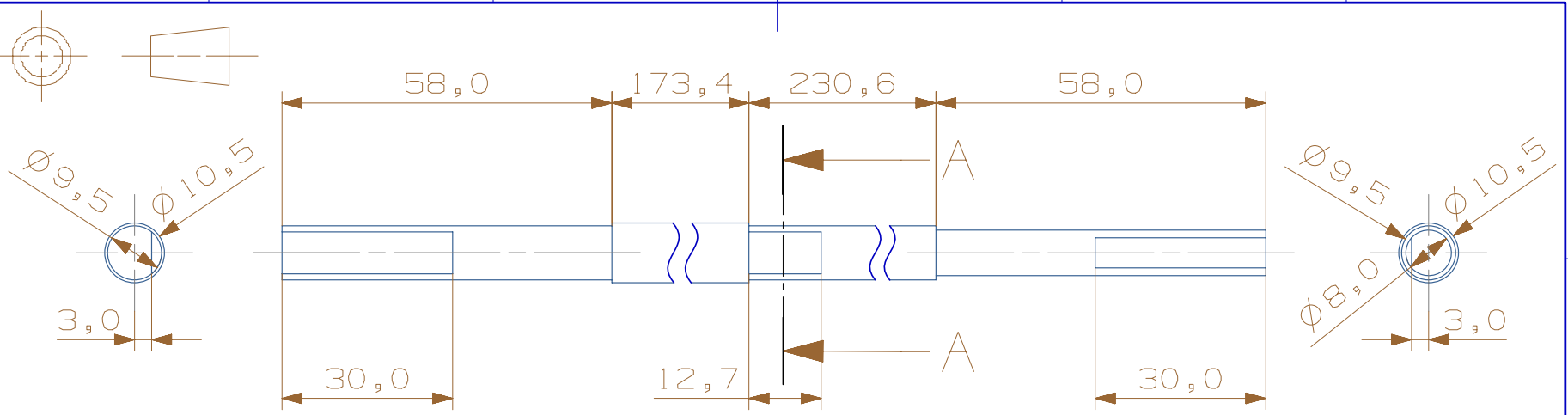
TITLE

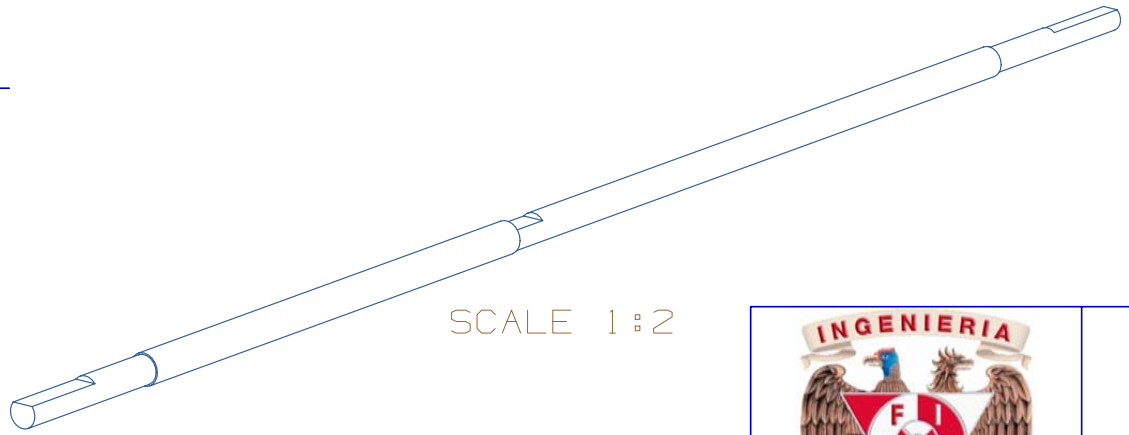
EJE POLEA

DRAWN BY	JLAS
CHECKED BY	-
APPROVED BY	-

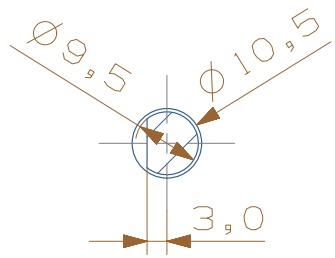
SIZE	FILE NAME	REV
A4	eje_polea	

SCALE 1:1	SHEET 1 OF 1	PLANO MEC-005





SCALE 1:2



SECTION A-A

ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

ROBOT ESFERICO

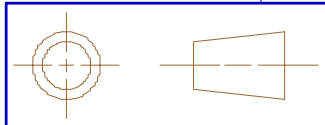
TITLE

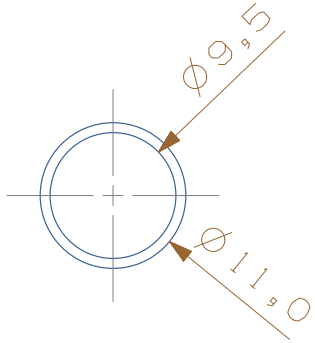
EJE PRINCIPAL

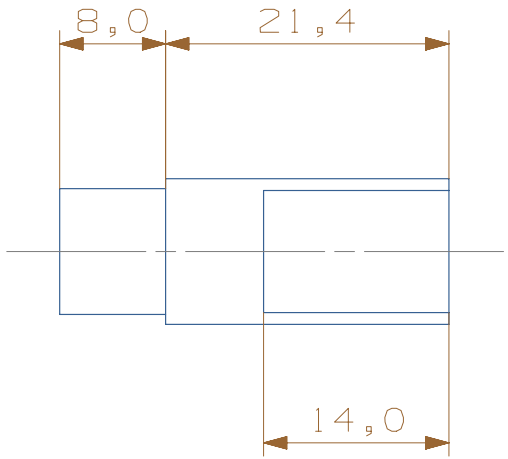
DRAWN BY	JLAS
CHECKED BY	-
APPROVED BY	-

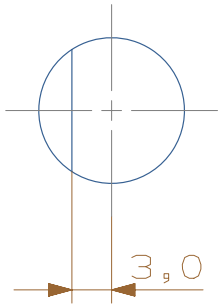
SIZE	FILE NAME	REV
A4	eje_principal	☒

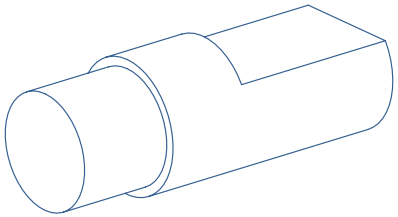
SCALE 1:1	SHEET 1 OF 1	PLANO MEC-006











ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

ROBOT ESFERICO

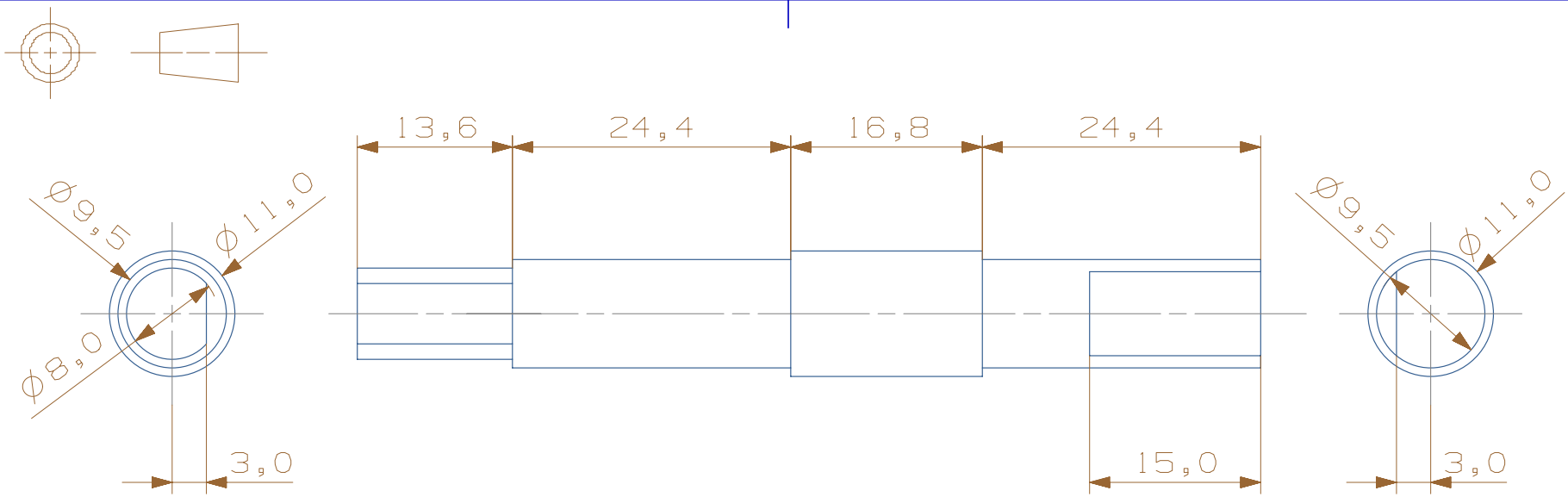
TITLE

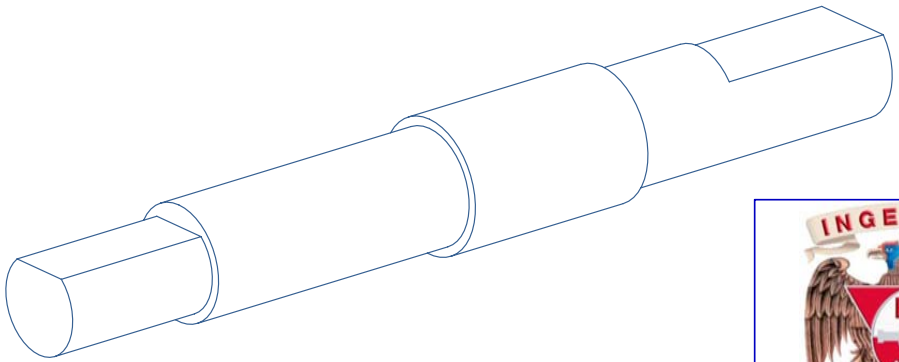
EJE VOLANTE EXTERIOR

DRAWN BY	JLAS
CHECKED BY	-
APPROVED BY	-

SIZE	FILE NAME	REV
A4	eje_volante_ext	☒

SCALE 2:1	SHEET 1 OF 1	PLANO MEC-007





ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

ROBOT ESFERICO

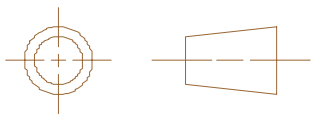
TITLE

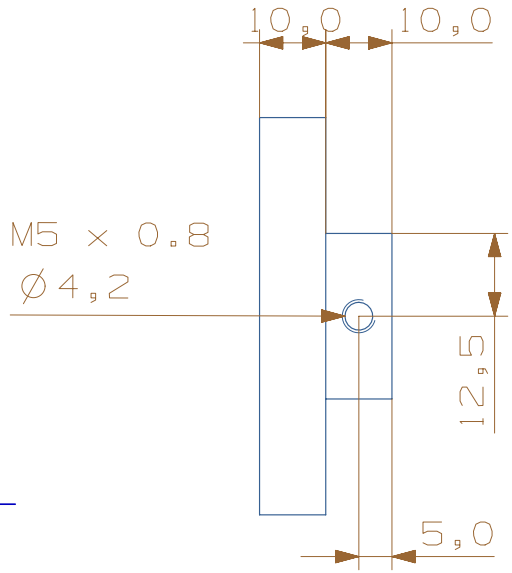
EJE VOLANTE INTERIOR

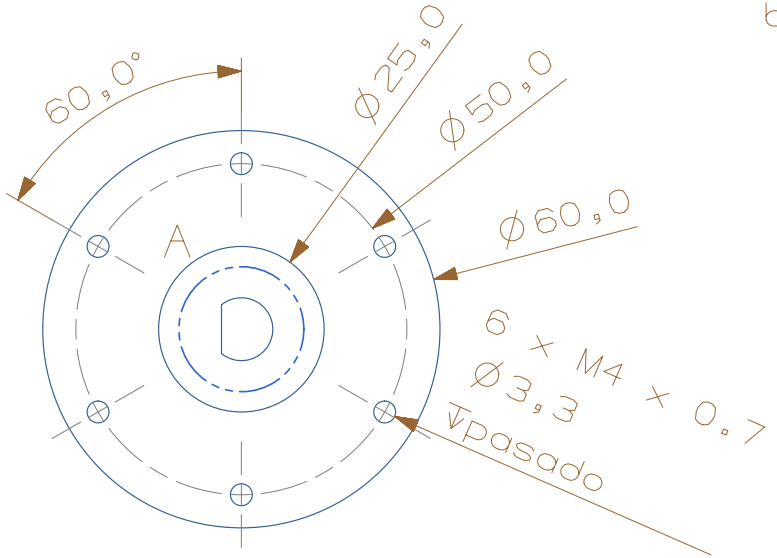
DRAWN BY	JLAS
CHECKED BY	-
APPROVED BY	-

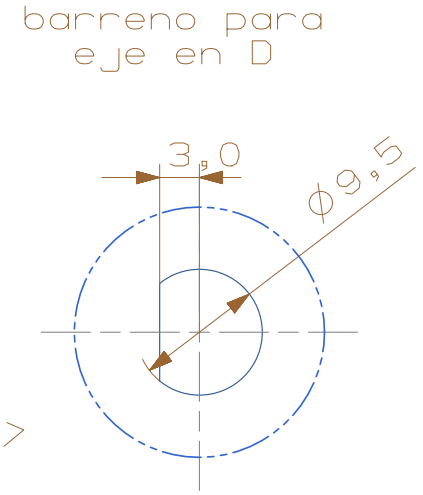
SIZE	FILE NAME	REV
A4	eje_volante_int	☒

SCALE 2:1	SHEET 1 OF 1	PLANO MEC-008

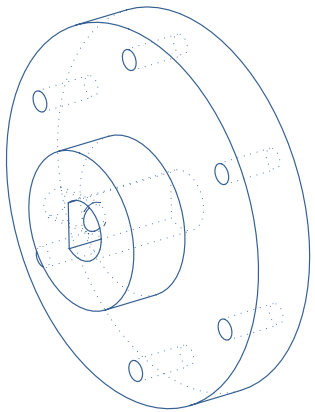








DETAIL A
SCALE 2:1



ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

ROBOT ESFERICO

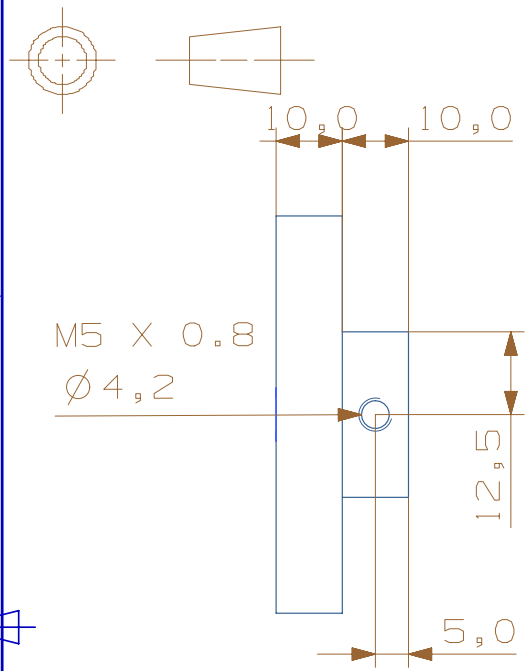
TITLE

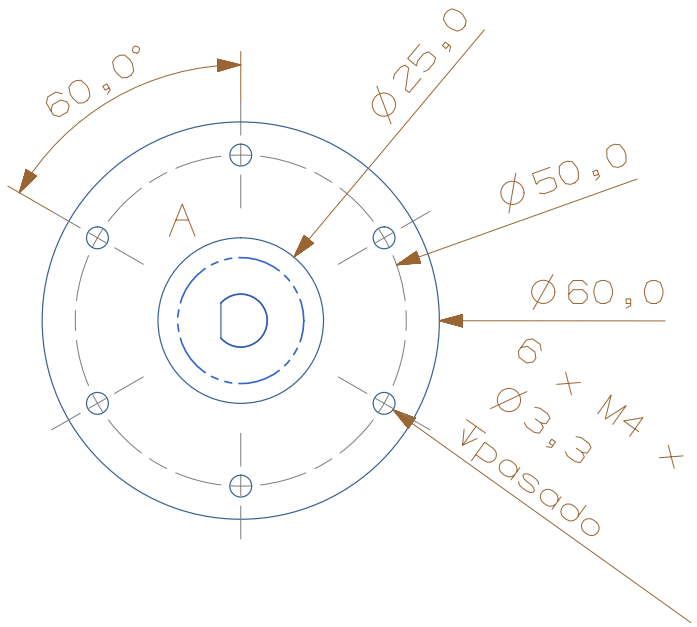
BRIDA EJE PRINCIPAL DERECHA

DRAWN BY	JLAS
CHECKED BY	-
APPROVED BY	-

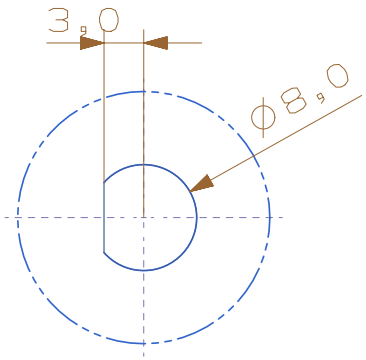
SIZE	FILE NAME	REV
A4	brida_eje_principal_der	☒

SCALE 1:1	SHEET 1 OF 1	PLANO MEC-009

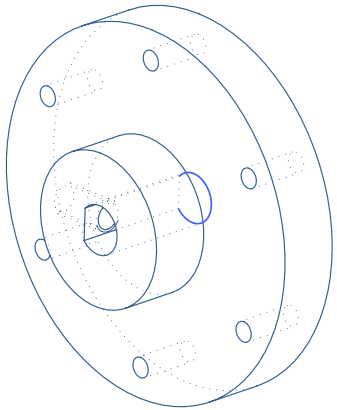




barreno para
eje en D



DETAIL A
SCALE 2:1



ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

ROBOT ESFERICO

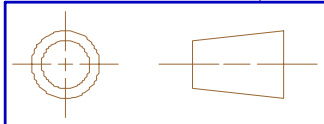
TITLE

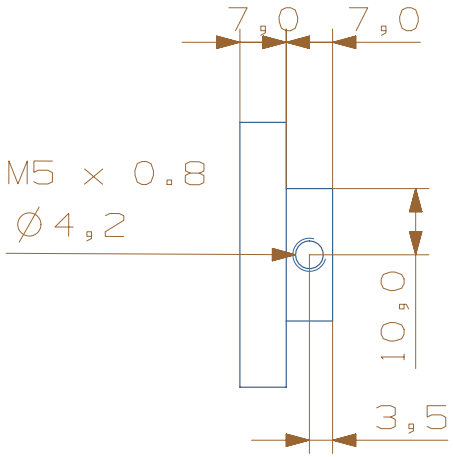
BRIDA EJE PRINCIPAL IZQUIERDA

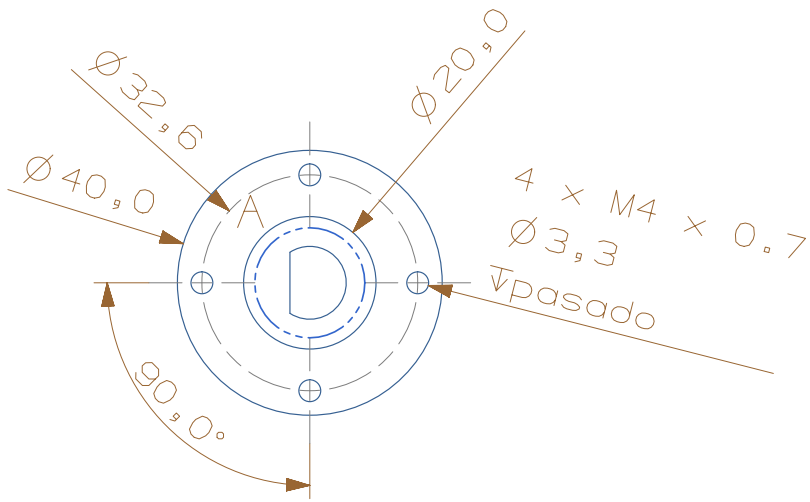
DRAWN BY	JLAS
CHECKED BY	-
APPROVED BY	-

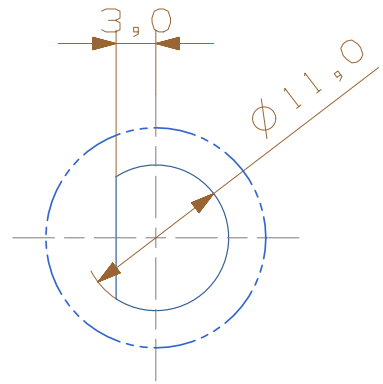
SIZE	FILE NAME	REV
A4	brida_eje_principal_izq	☒

SCALE 1:1	SHEET 1 OF 1	PLANO MEC-010

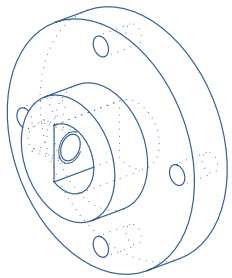








DETAIL A
SCALE 2:1



ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

ROBOT ESFERICO

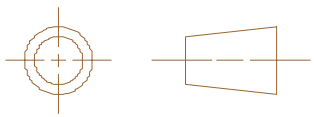
TITLE

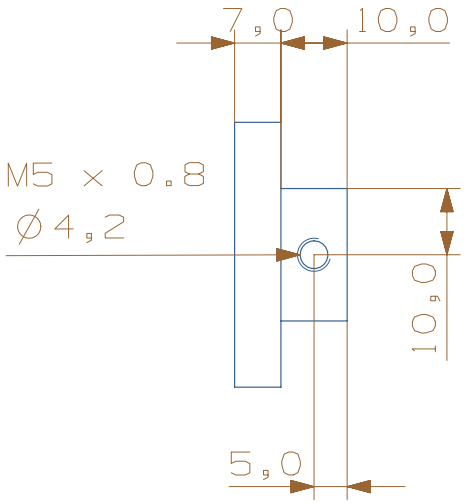
BRIDA SOPORTE VOLANTE EXTERIOR

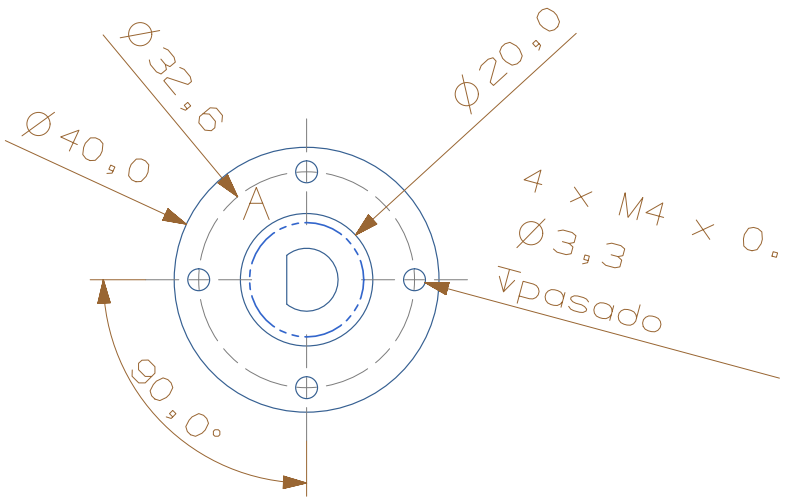
DRAWN BY	JLAS
CHECKED BY	-
APPROVED BY	-

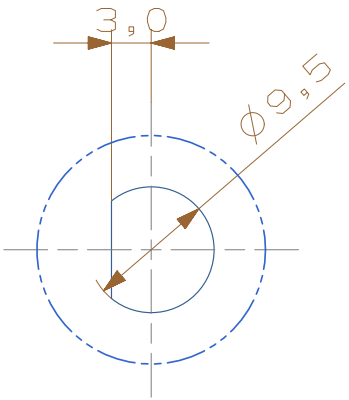
SIZE	FILE NAME	REV
A4	brida_soporte_volante_ext	

SCALE 1:1	SHEET 1 OF 1	PLANO MEC-011

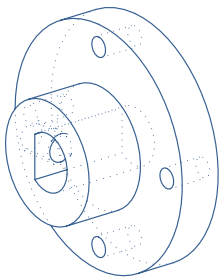








DETAIL A
SCALE 2:1



ALL DIMENSIONS IN MM

INGENIERIA

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

ROBOT ESFERICO

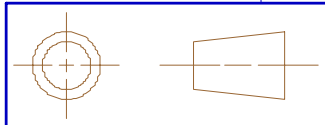
TITLE

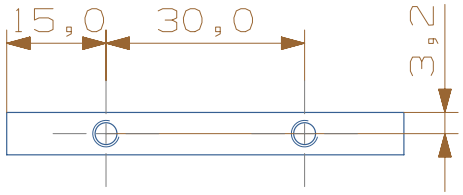
BRIDA SOPORTE VOLANTE INTERIOR

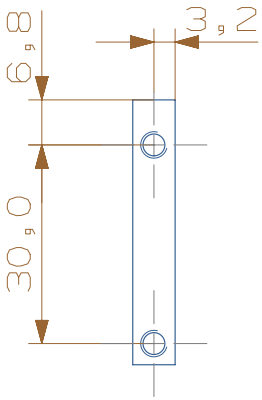
DRAWN BY	JLAS
CHECKED BY	-
APPROVED BY	-

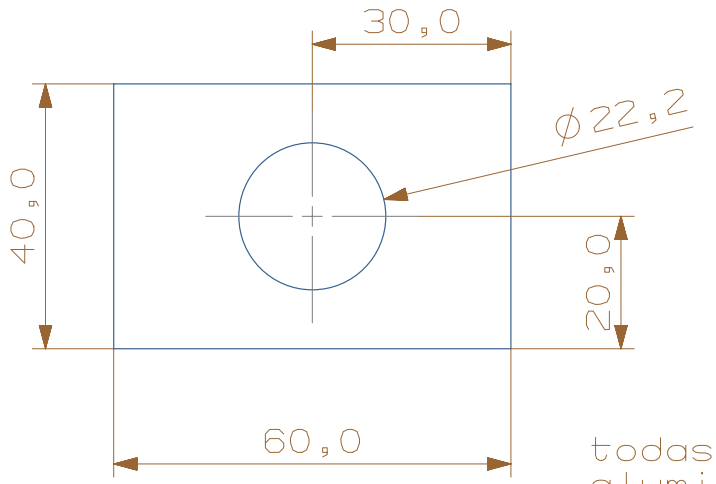
SIZE	FILE NAME	REV
A4	brida_soporte_volante_int	

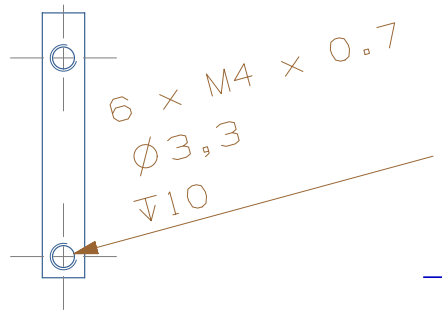
SCALE 1:1	SHEET 1 OF 1	PLANO MEC-012



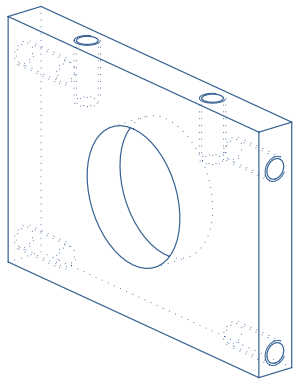








todas las placas son de aluminio de 6.4mm a menos que se indique otra cosa



ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

ROBOT ESFERICO

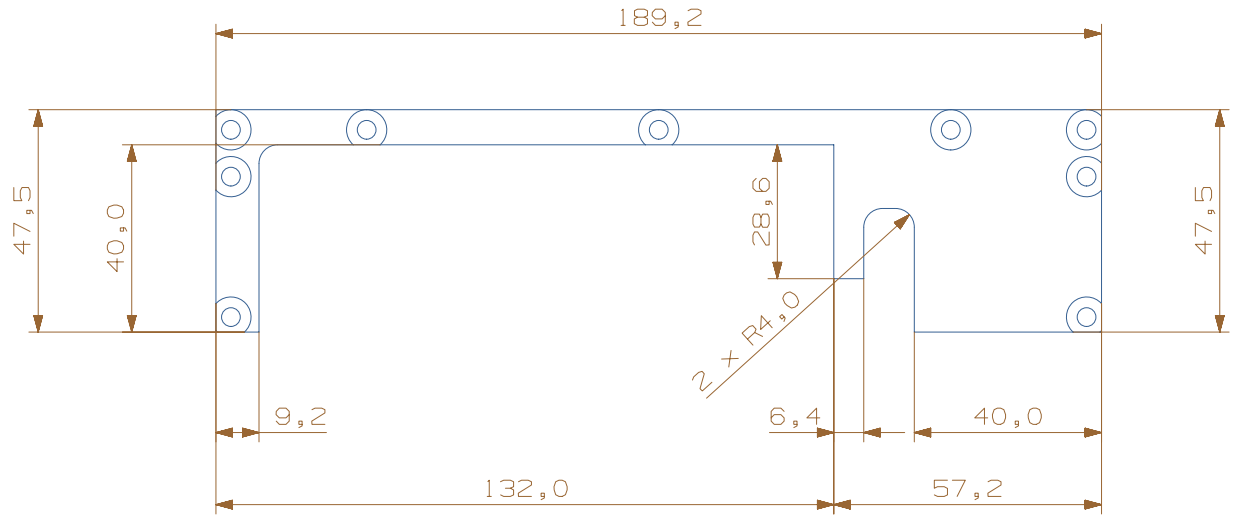
TITLE

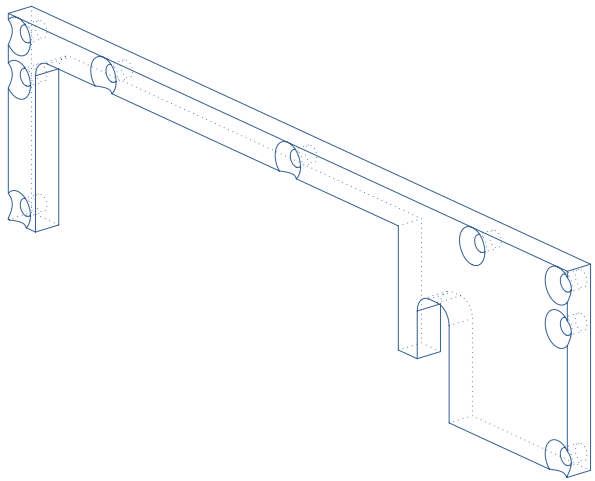
PLACA EJE POLEA DELANTERA/TRASERA

DRAWN BY	JLAS
CHECKED BY	-
APPROVED BY	-

SIZE	FILE NAME	REV
A4	placa_eje_polea_del_tras	

SCALE 1:1	SHEET 1 OF 1	PLANO MEC-013





ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

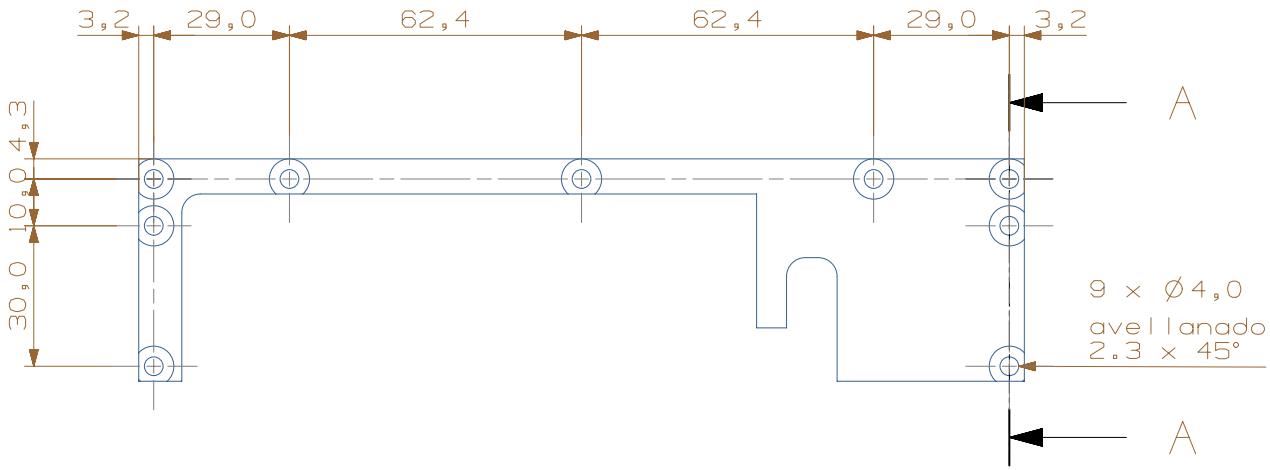
FACULTAD DE INGENIERIA

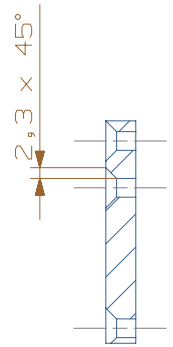
ROBOT ESFERICO

TITLE

PLACA EJE POLEA LATERAL

DRAWN BY	JLAS	SIZE	FILE NAME		REV
CHECKED BY	-	A3	placa_eje_polea_lat		
APPROVED BY	-	SCALE 1:1	SHEET 1 OF 2	PLANO MEC-014	





SECTION A-A

detalle del avellanado

ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

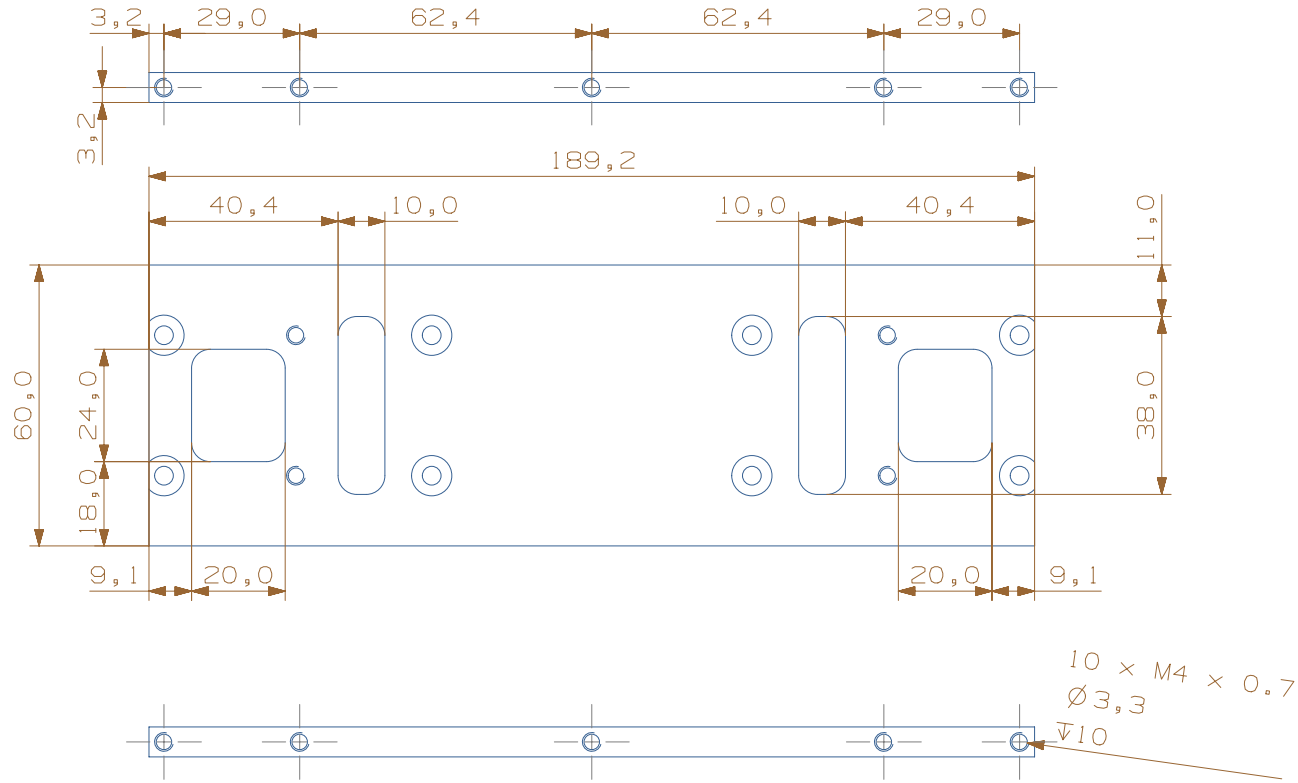
FACULTAD DE INGENIERIA

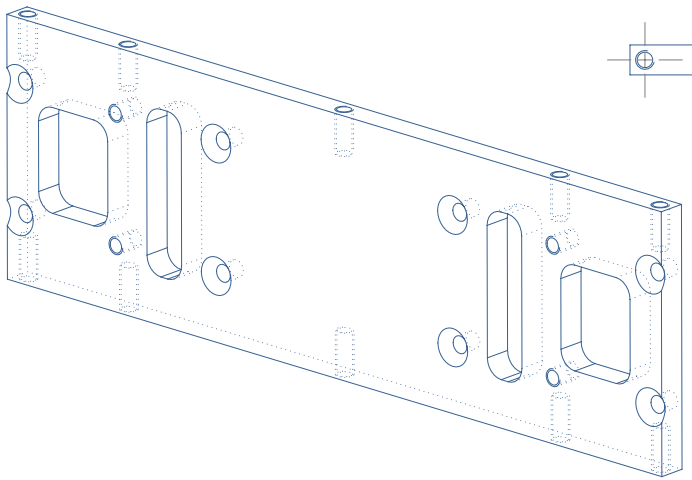
ROBOT ESFERICO

TITLE

PLACA EJE POLEA LATERAL

DRAWN BY	JLAS	SIZE	FILE NAME		REV
CHECKED BY	-	A3	placa_eje_polea_lat		
APPROVED BY	-	SCALE 1:1	SHEET 2 OF 2	PLANO MEC-014	





ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

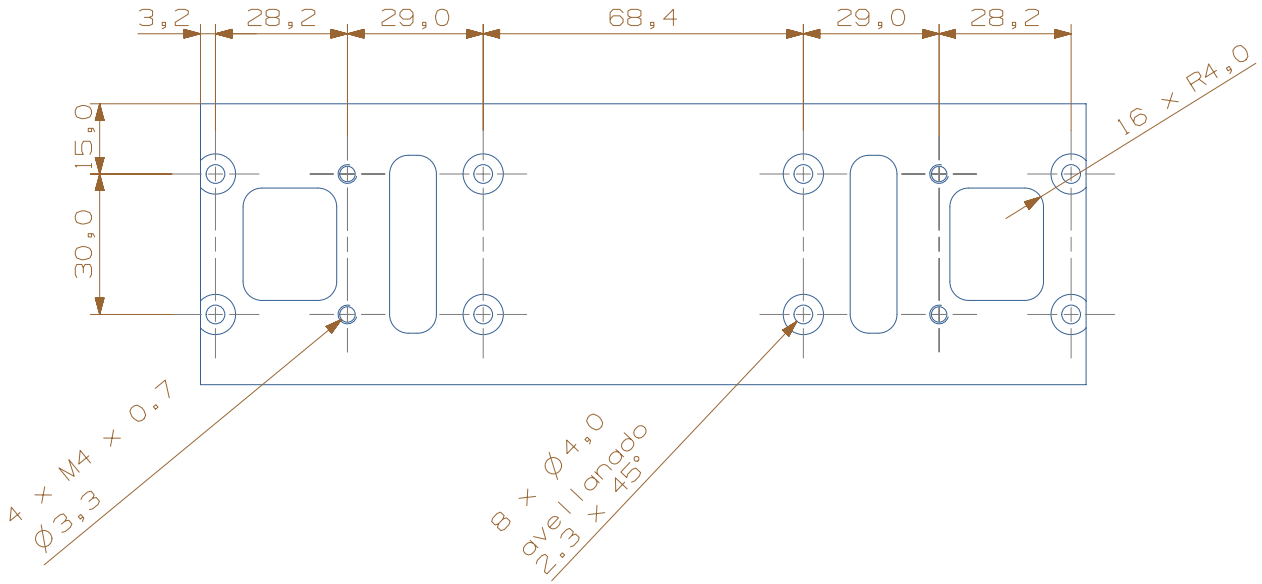
FACULTAD DE INGENIERIA

ROBOT ESFERICO

TITLE

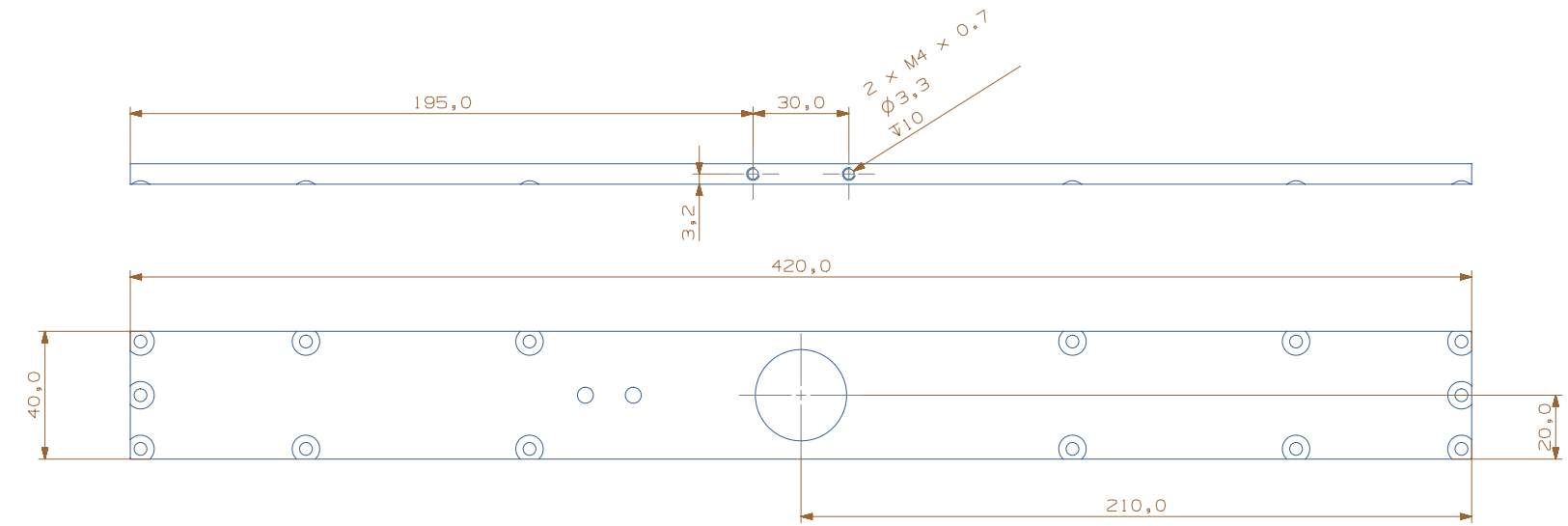
PLACA EJE POLEA SUPERIOR

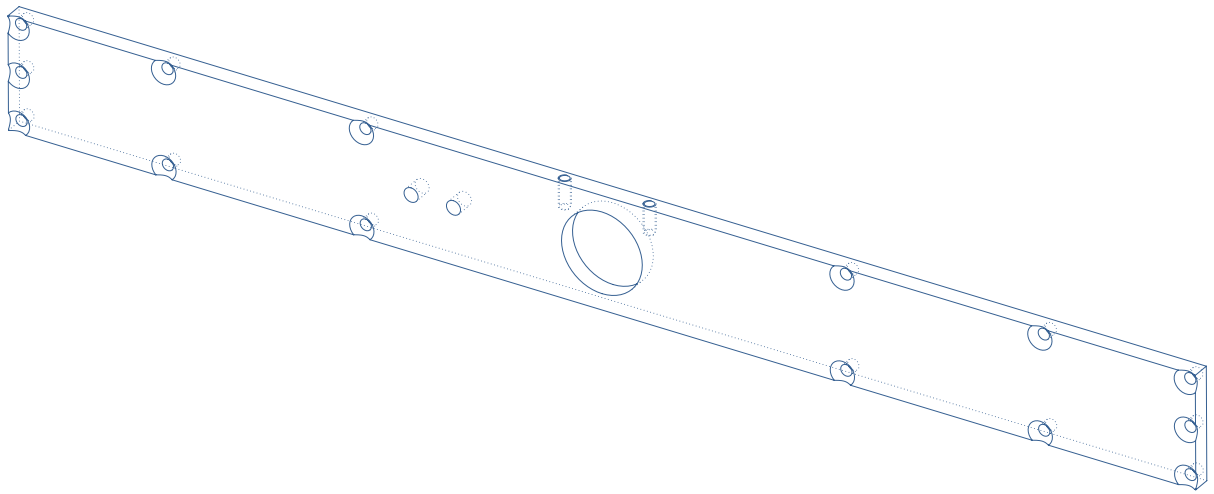
DRAWN BY	JLAS	SIZE	FILE NAME		REV
CHECKED BY	-	A3	placa_eje_polea_sup		
APPROVED BY	-	SCALE 1:1	SHEET 1 OF 2	PLANO MEC-015	



ALL DIMENSIONS IN MM

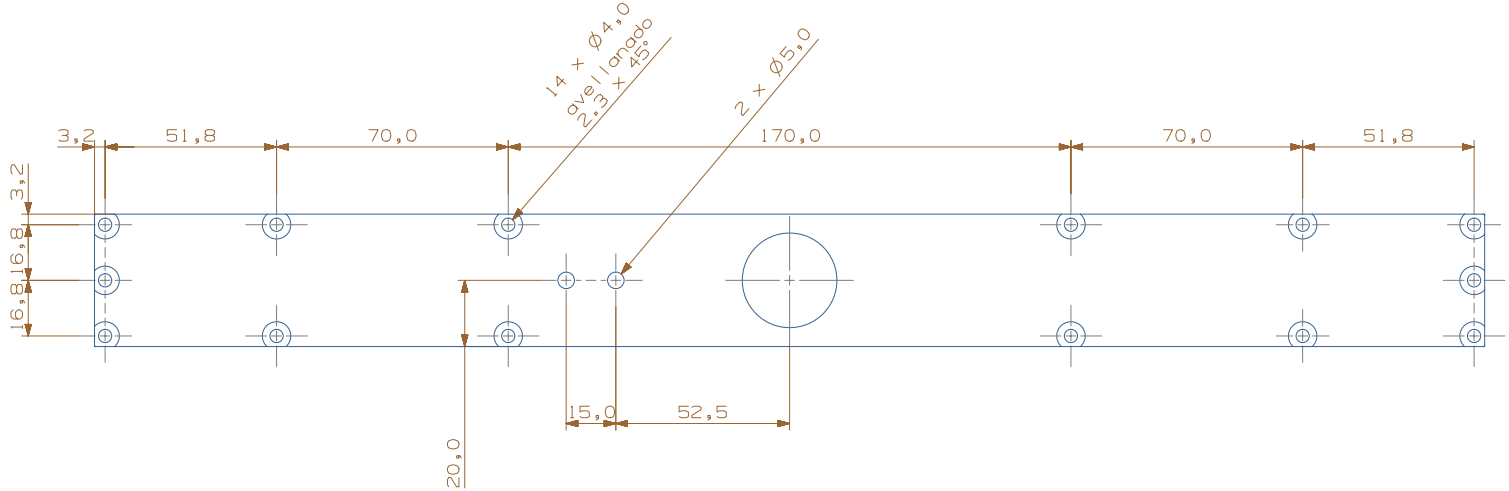
UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO	
FACULTAD DE INGENIERIA	
ROBOT ESFERICO	
TITLE: PLACA EJE POLEA SUPERIOR	
DRAWN BY: JLAS	SIZE: A3 / FILE NAME: placa_eje_polea_sup / REV:
CHECKED BY: -	
APPROVED BY: -	SCALE 1:1 / SHEET 2 OF 2 / PLANO MEC-015





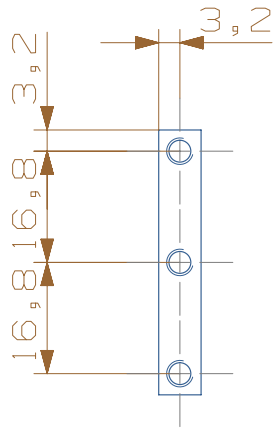
ALL DIMENSIONS IN MM

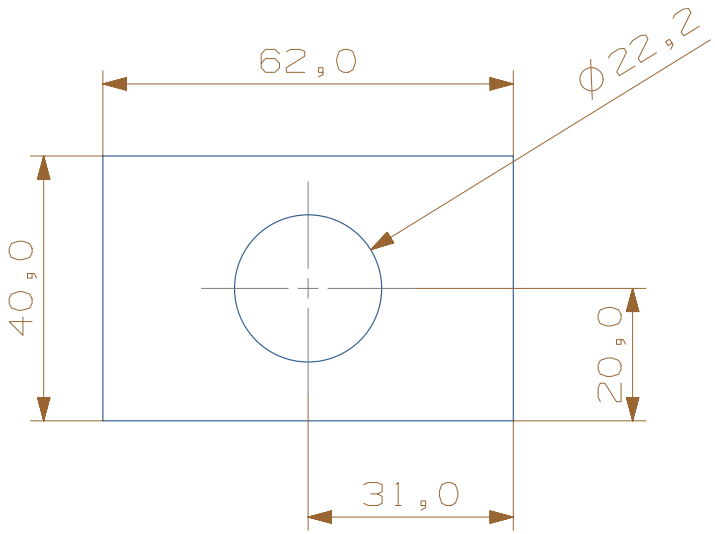
DRAWN BY	JLAS	UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO			
CHECKED BY	-	FACULTAD DE INGENIERIA			
APPROVED BY	-	ROBOT ESFERICO			
		TITLE: PLACA EJE PRINCIPAL DELANTERA/TRASERA			
		SIZE: A2	FILE NAME: placa_eje_principal_del_tras	REV: 0	
		SCALE 1:1	SHEET 1 OF 2	PLANO MEC-016	

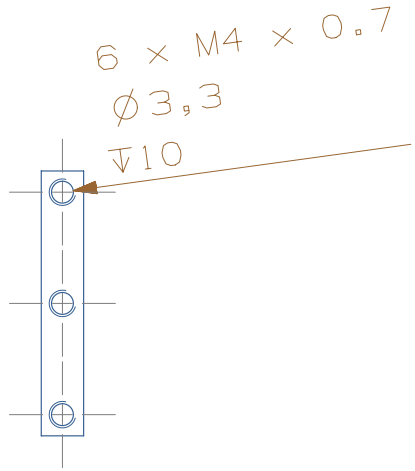


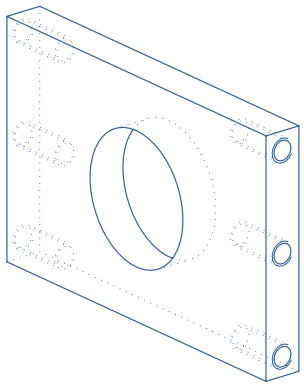
ALL DIMENSIONS IN MM

INGENIERIA		UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO			
		FACULTAD DE INGENIERIA			
		ROBOT ESFERICO			
		TITLE: PLACA EJE PRINCIPAL DELANTERA/TRASERA			
DRAWN BY	JLAS	SIZE	FILE NAME		REV
CHECKED BY	-	A2	placa_eje_principal_del_tras		
APPROVED BY	-	SCALE 1:1	SHEET 2 OF 2	PLANO MEC-016	









ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

ROBOT ESFERICO

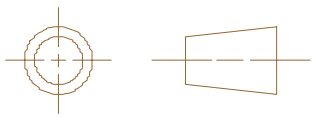
TITLE

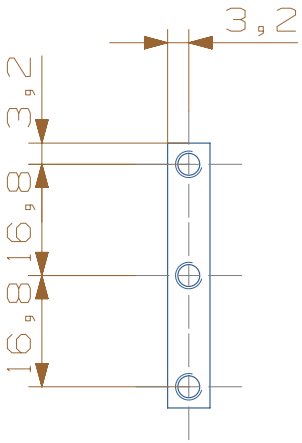
PLACA EJE PRINCIPAL LATERAL DERECHA

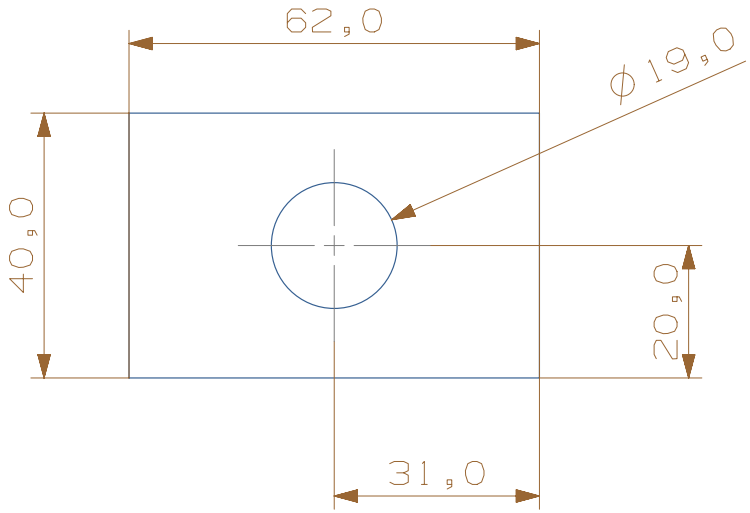
DRAWN BY	JLAS
CHECKED BY	-
APPROVED BY	-

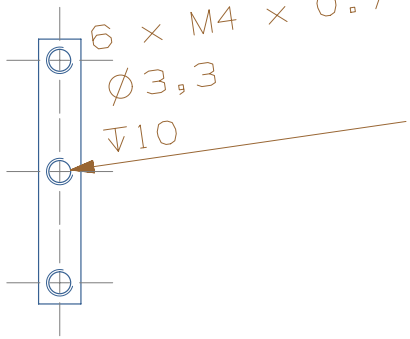
SIZE	FILE NAME	REV
A4	placa_eje_principal_lat_der	

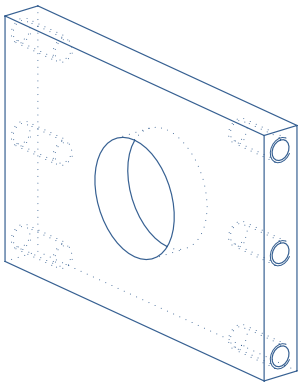
SCALE 1:1 | SHEET 1 OF 1 | PLANO MEC-017











ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

ROBOT ESFERICO

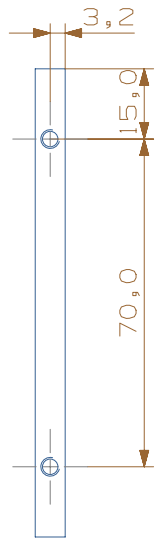
TITLE

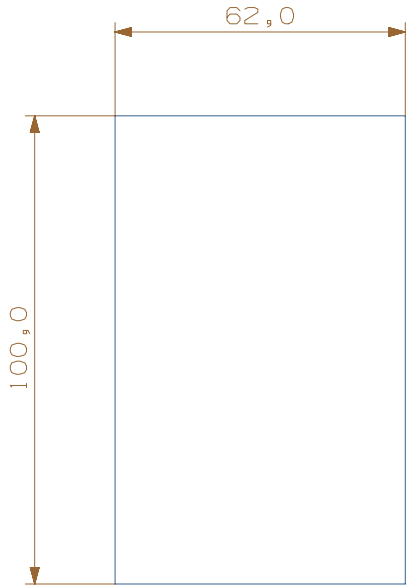
PLACA EJE PRINCIPAL LATERAL IZQUIERDA

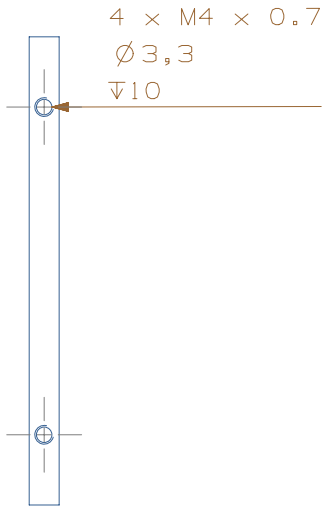
DRAWN BY	JLAS
CHECKED BY	-
APPROVED BY	-

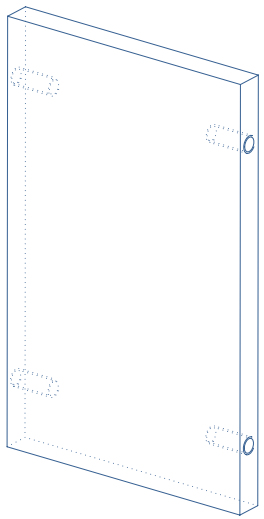
SIZE	FILE NAME	REV
A4	placa_eje_principal_lat_izq	☒

SCALE 1:1	SHEET 1 OF 1	PLANO MEC-018









ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

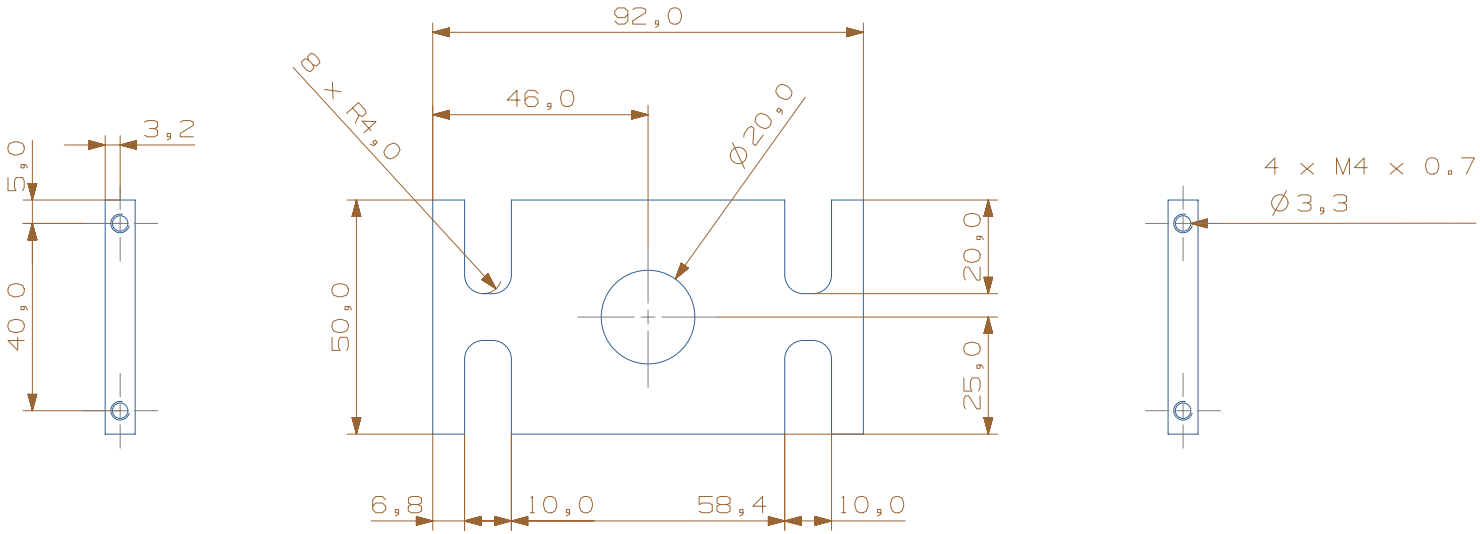
FACULTAD DE INGENIERIA

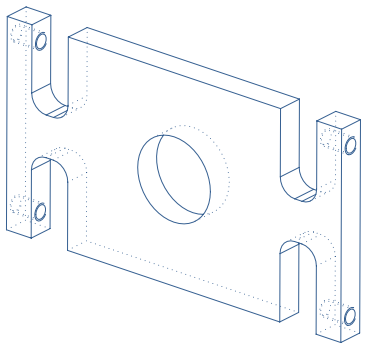
ROBOT ESFERICO

TITLE

PLACA EJE PRINCIPAL UNION

DRAWN BY	JLAS	SIZE	FILE NAME		REV
CHECKED BY	-	A3	placa_eje_principal_union		
APPROVED BY	-	SCALE 1:1	SHEET 1 OF 1	PLANO MEC-019	





ALL DIMENSIONS IN MM

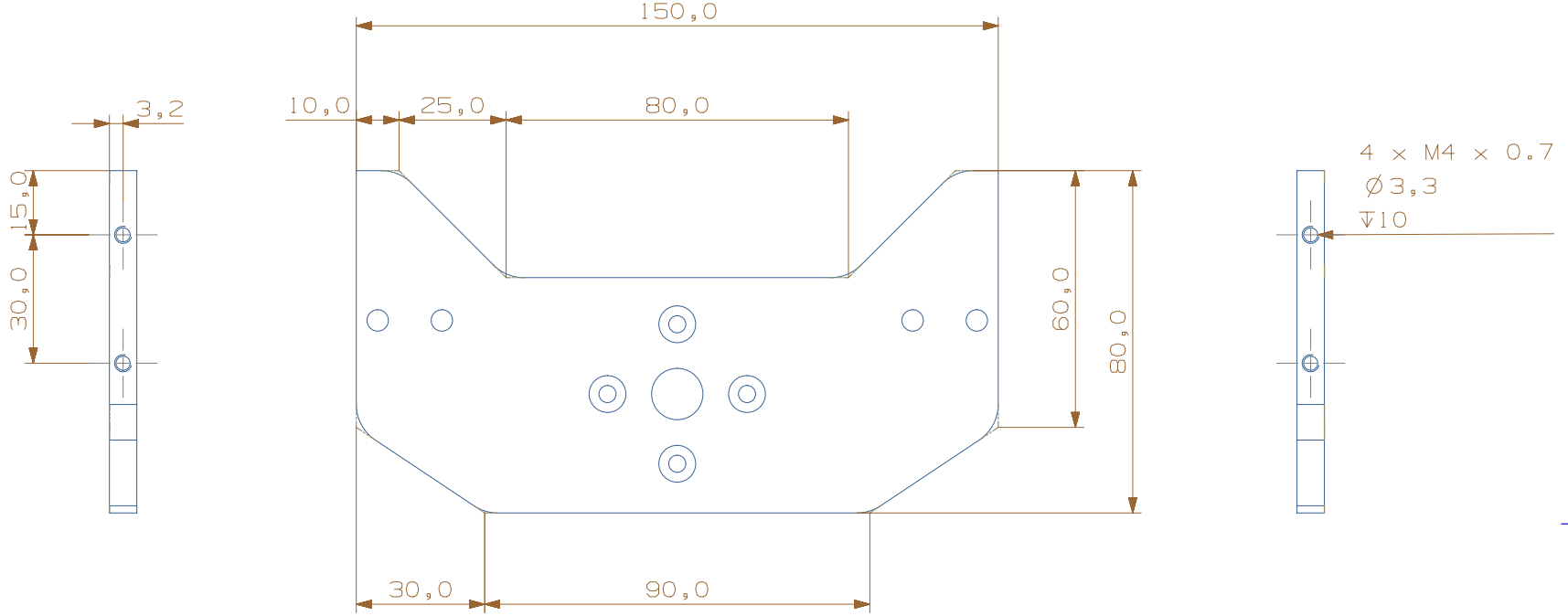
UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

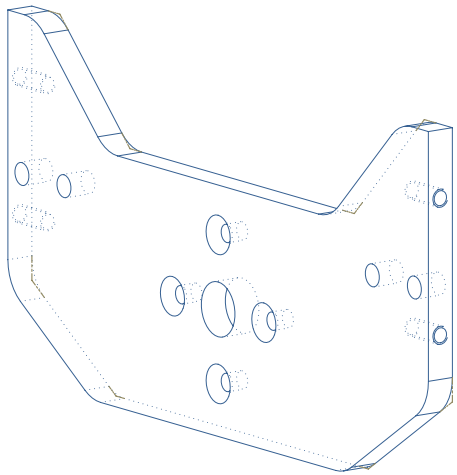
FACULTAD DE INGENIERIA

ROBOT ESFERICO

TITLE: PLACA ENCODER MOTOR VERTICAL

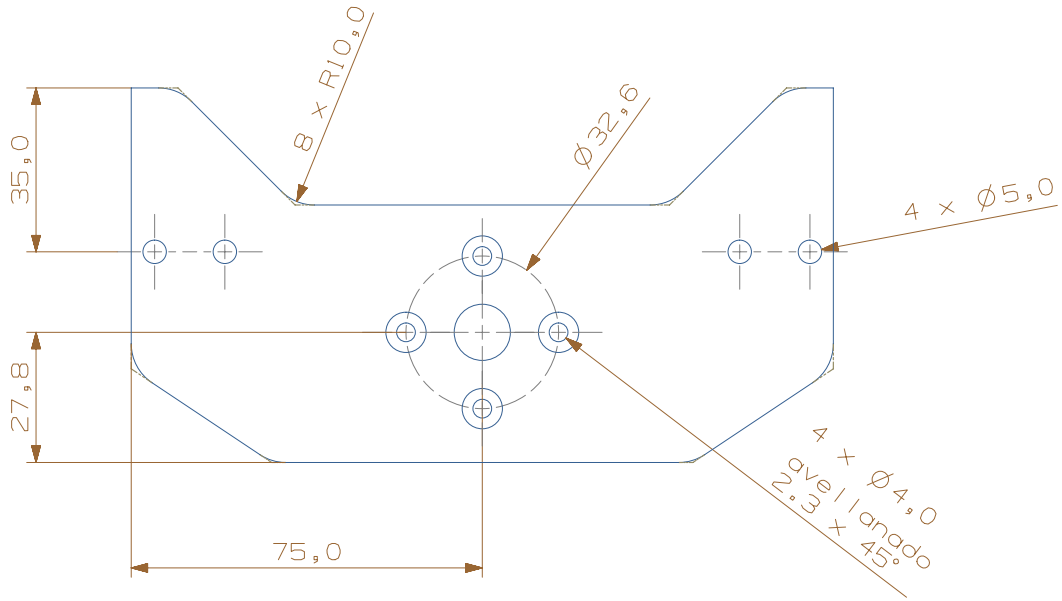
DRAWN BY	JLAS	SIZE	FILE NAME	REV	
CHECKED BY	-	A3	placa_encoder_motor_vert		
APPROVED BY	-	SCALE 1:1	SHEET 1 OF 1	PLANO MEC-020	





ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO	
FACULTAD DE INGENIERIA	
ROBOT ESFERICO	
TITLE: PLACA PENDULO EXTERIOR	
DRAWN BY: JLAS	SIZE: A3
CHECKED BY: -	FILE NAME: placa_pendulo_ext
APPROVED BY: -	REV: 0
SCALE 1:1	SHEET 1 OF 2
PLANO MEC-021	



ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

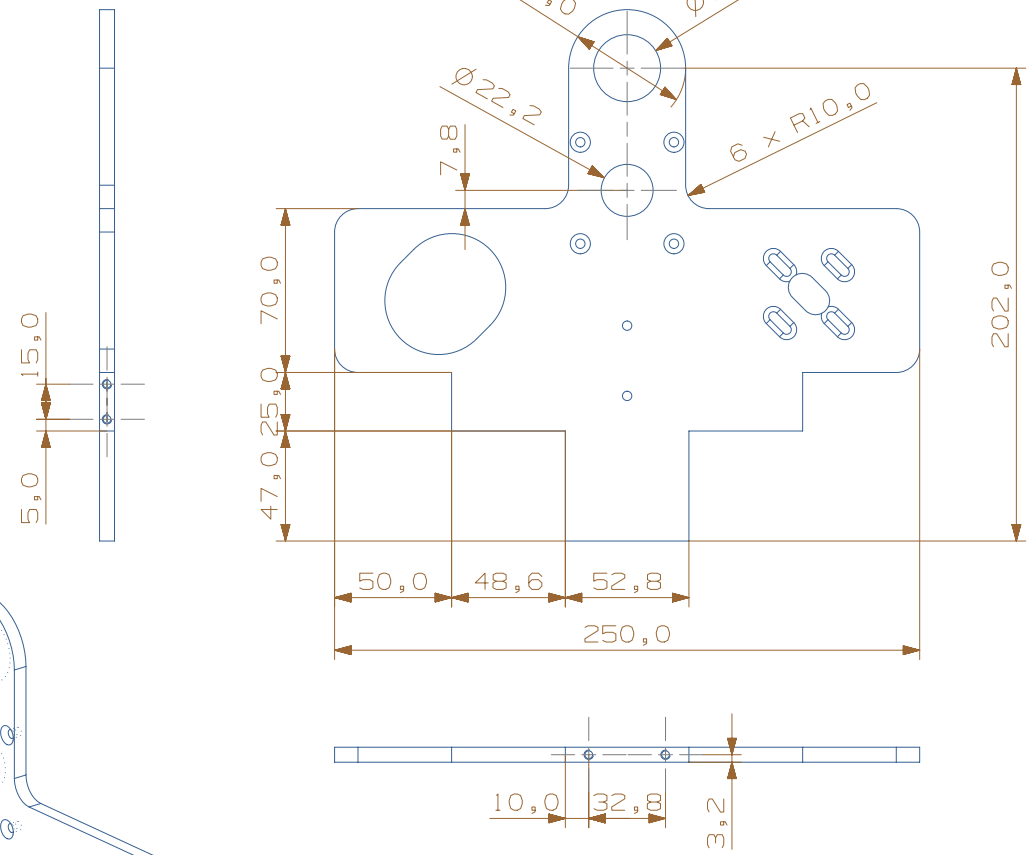
FACULTAD DE INGENIERIA

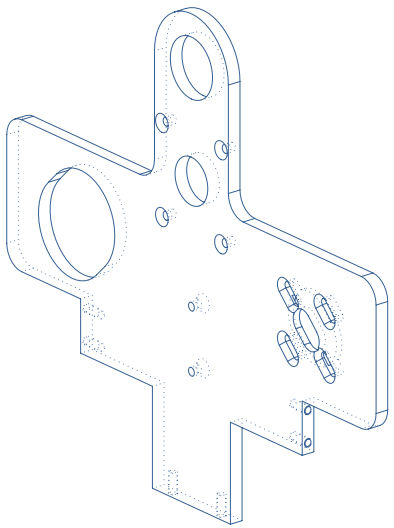
ROBOT ESFERICO

TITLE

PLACA PENDULO EXTERIOR

DRAWN BY	JLAS	SIZE	FILE NAME		REV
CHECKED BY	-	A3	placa_pendulo_ext		
APPROVED BY	-	SCALE 1:1	SHEET 2 OF 2	PLANO MEC-021	





ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

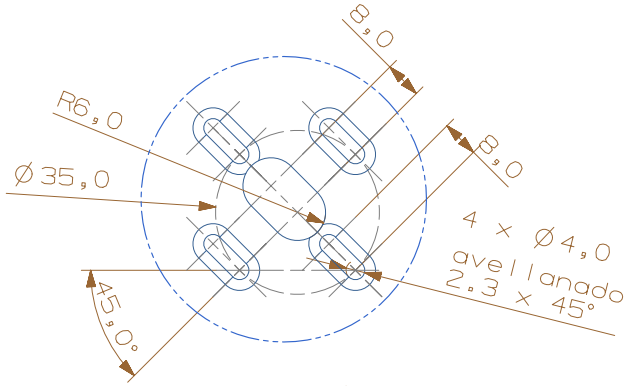
FACULTAD DE INGENIERIA

ROBOT ESFERICO

TITLE

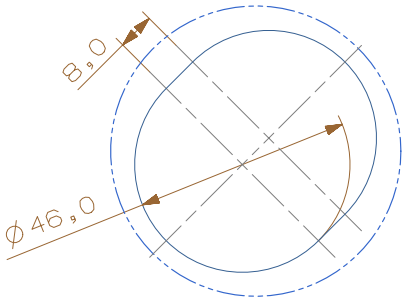
PLACA PENDULO INTERIOR

DRAWN BY	JLAS	SIZE	FILE NAME		REV
CHECKED BY	-	A3	placa_pendulo_int		
APPROVED BY	-	SCALE 1:2	SHEET 1 OF 3	PLANO MEC-022	

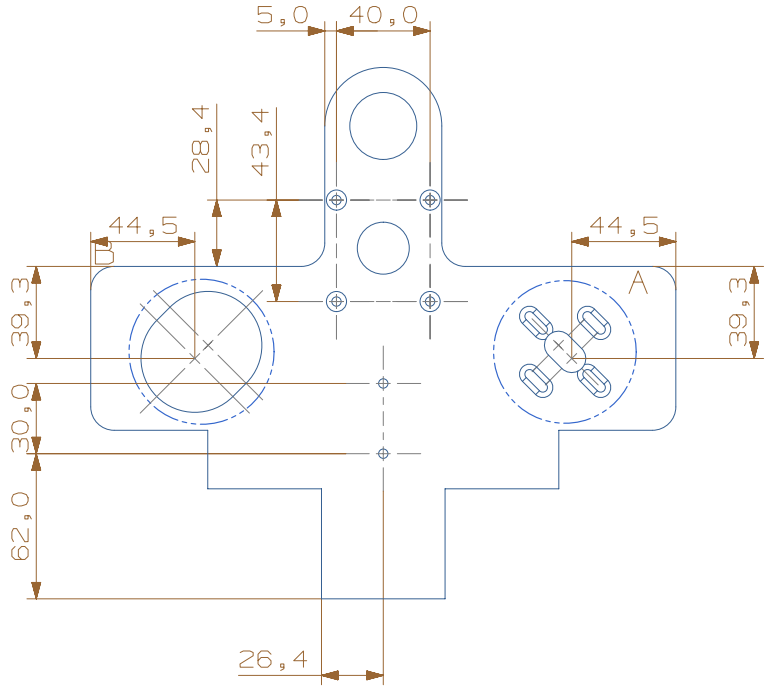


DETAIL A
SCALE 1:1

perforaciones ovaladas de
8mm entre centros a 45°



DETAIL B
SCALE 1:1



ALL DIMENSIONS IN MM

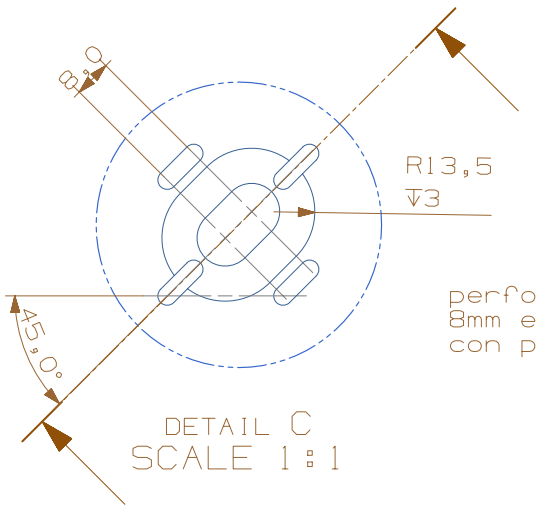
DRAWN BY	JLAS
CHECKED BY	-
APPROVED BY	-

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO
FACULTAD DE INGENIERIA
ROBOT ESFERICO

TITLE
PLACA PENDULO INTERIOR

SIZE	FILE NAME	REV
A3	placa_pendulo_int	

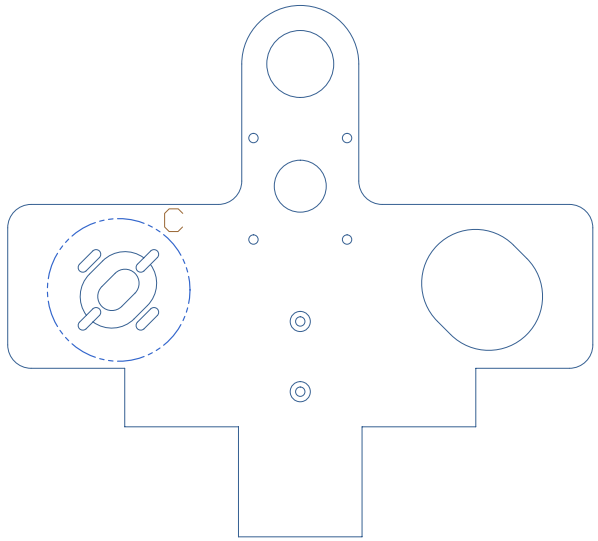
SCALE 1:2	SHEET 2 OF 3	PLANO MEC-022

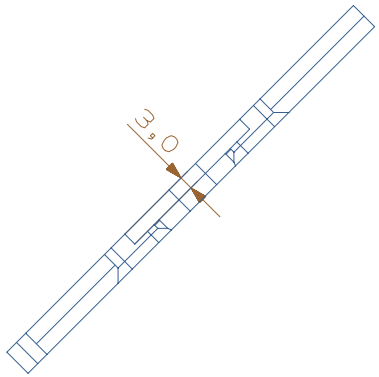


perforacion ovalada de
8mm entre centros a 45°
con profundidad de 3mm

DETAIL C
SCALE 1:1

D





SECTION D-D

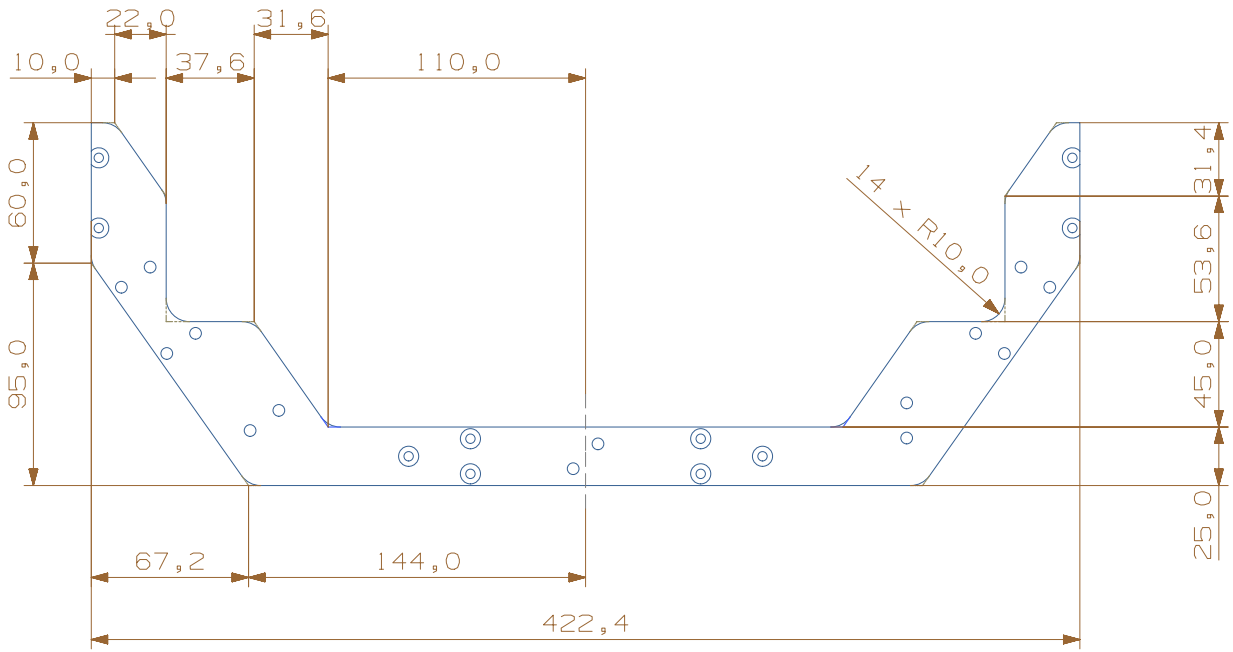
ALL DIMENSIONS IN MM

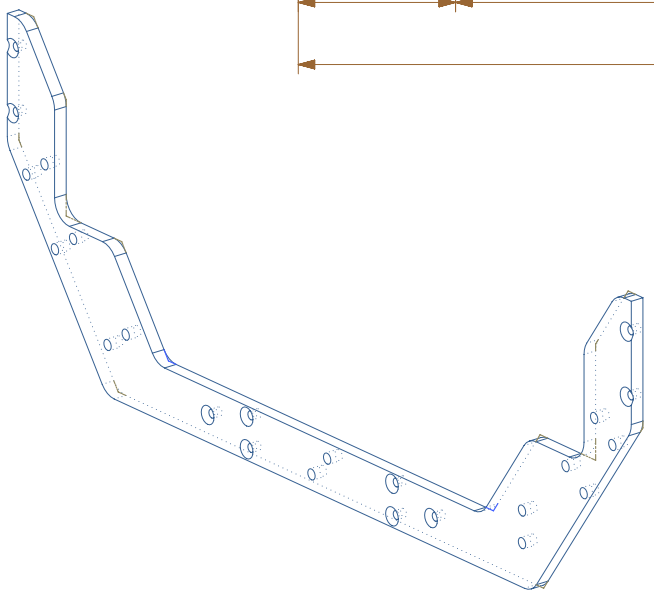
UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO
FACULTAD DE INGENIERIA

ROBOT ESFERICO

TITLE
PLACA PENDULO
INTERIOR

DRAWN BY	JLAS	SIZE	FILE NAME		REV
CHECKED BY	-	A3	placa_pendulo_int		
APPROVED BY	-	SCALE 1:2	SHEET 3 OF 3	PLANO MEC-022	





ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

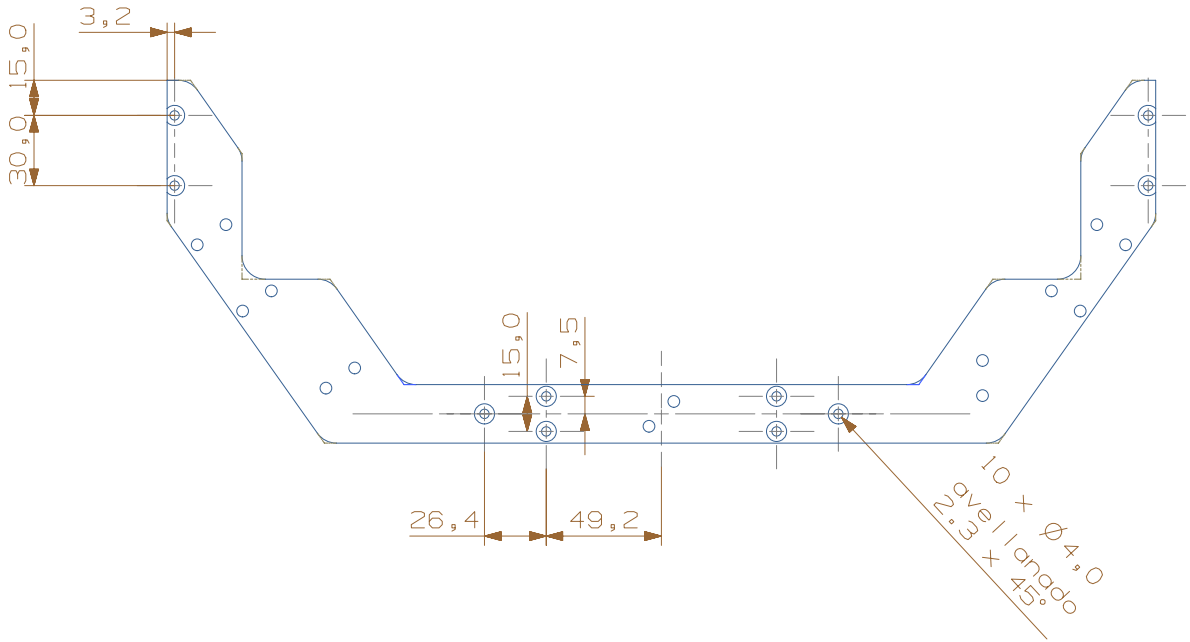
FACULTAD DE INGENIERIA

ROBOT ESFERICO

TITLE

PLACA PENDULO LATERAL

DRAWN BY	JLAS	SIZE	FILE NAME		REV
CHECKED BY	-	A3	placa_pendulo_lat		
APPROVED BY	-	SCALE 1:2	SHEET 1 OF 3	PLANO MEC-023	



ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

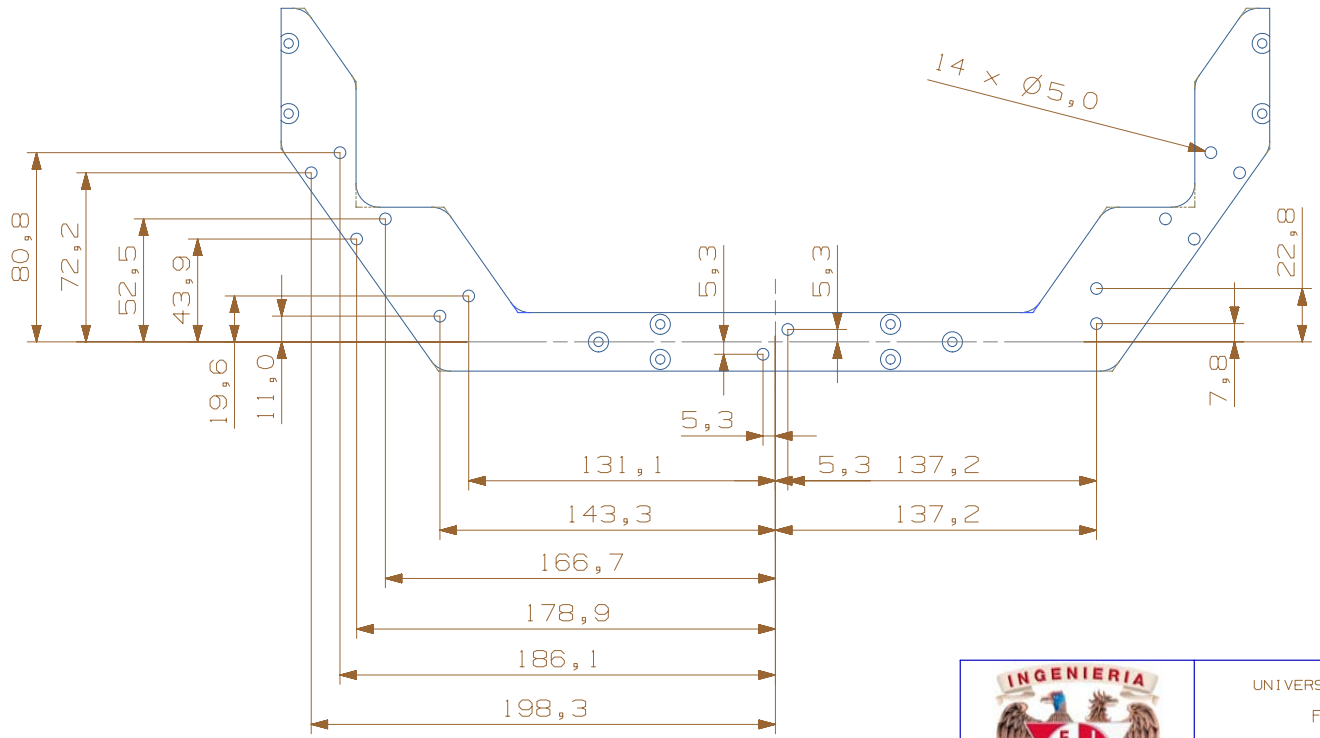
FACULTAD DE INGENIERIA

ROBOT ESFERICO

TITLE

PLACA PENDULO LATERAL

DRAWN BY	JLAS	SIZE	FILE NAME		REV
CHECKED BY	-	A3	placa_pendulo_lat		
APPROVED BY	-	SCALE 1:2	SHEET 2 OF 3	PLANO MEC-023	



ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

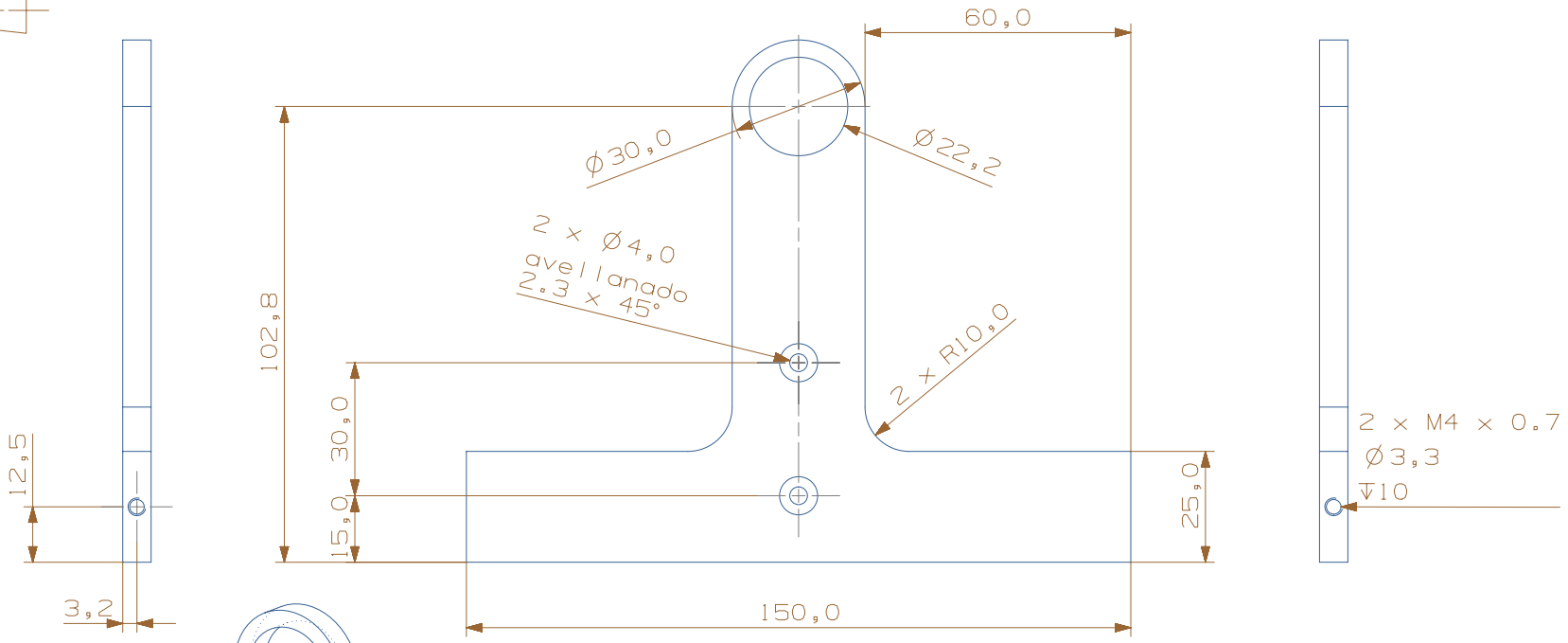
FACULTAD DE INGENIERIA

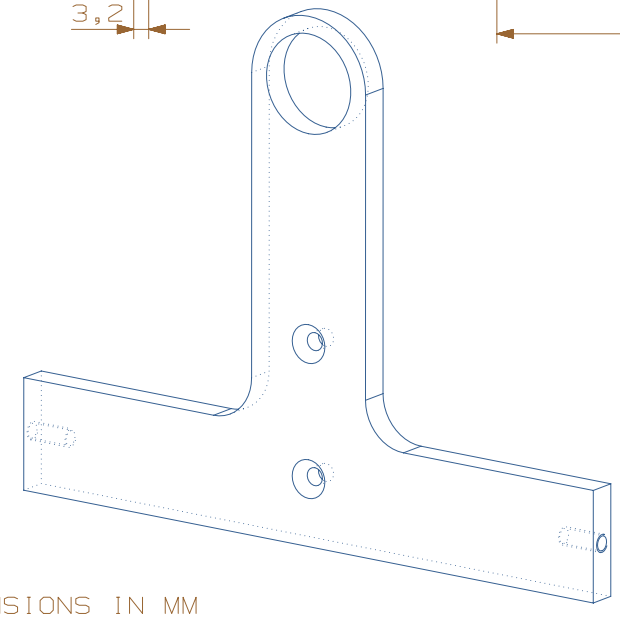
ROBOT ESFERICO

TITLE

PLACA PENDULO LATERAL

DRAWN BY	JLAS	SIZE	FILE NAME		REV
CHECKED BY	-	A3	placa_pendulo_lat		
APPROVED BY	-	SCALE 1:2	SHEET 3 OF 3	PLANO MEC-023	





ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

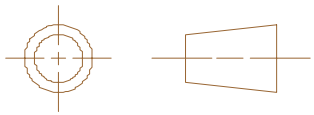
FACULTAD DE INGENIERIA

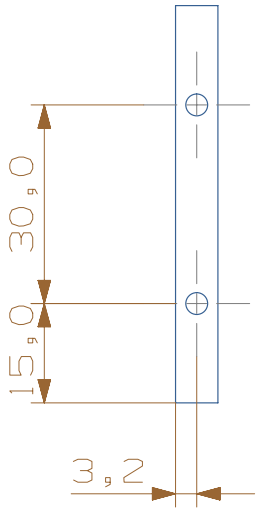
ROBOT ESFERICO

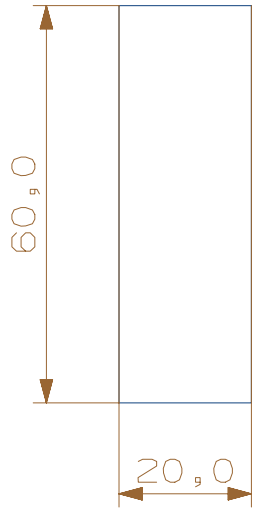
TITLE

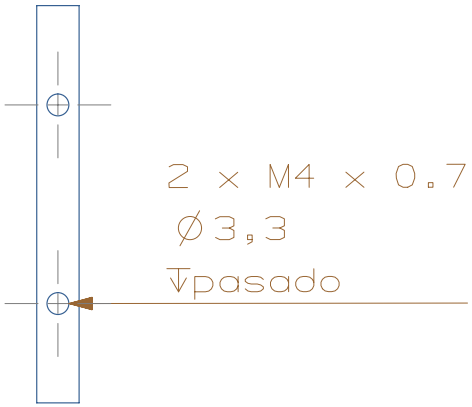
PLACA PENDULO
MEDIA

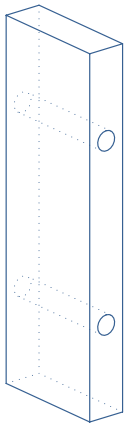
DRAWN BY	JLAS	SIZE	FILE NAME		REV
CHECKED BY	-	A3	placa_pendulo_med		
APPROVED BY	-	SCALE 1:1	SHEET 1 OF 1	PLANO MEC-024	











ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

ROBOT ESFERICO

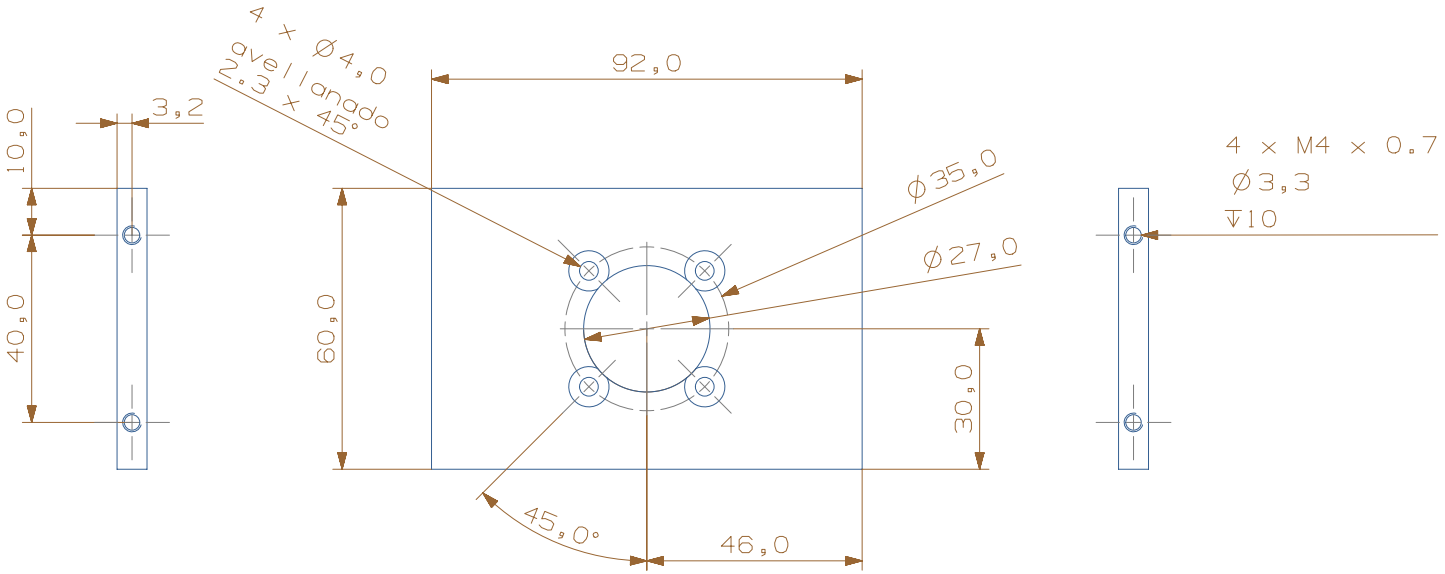
TITLE

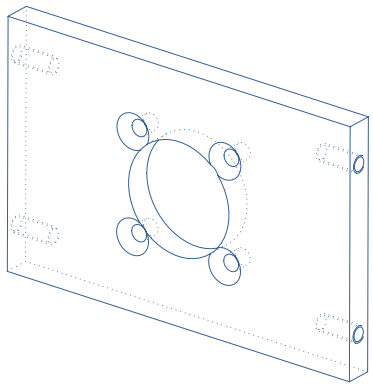
PLACA PENDULO
UNION

DRAWN BY	JLAS
CHECKED BY	-
APPROVED BY	-

SIZE	FILE NAME	REV
A4	placa_pendulo_union	

SCALE 1:1	SHEET 1 OF 1	PLANO MEC-025





ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

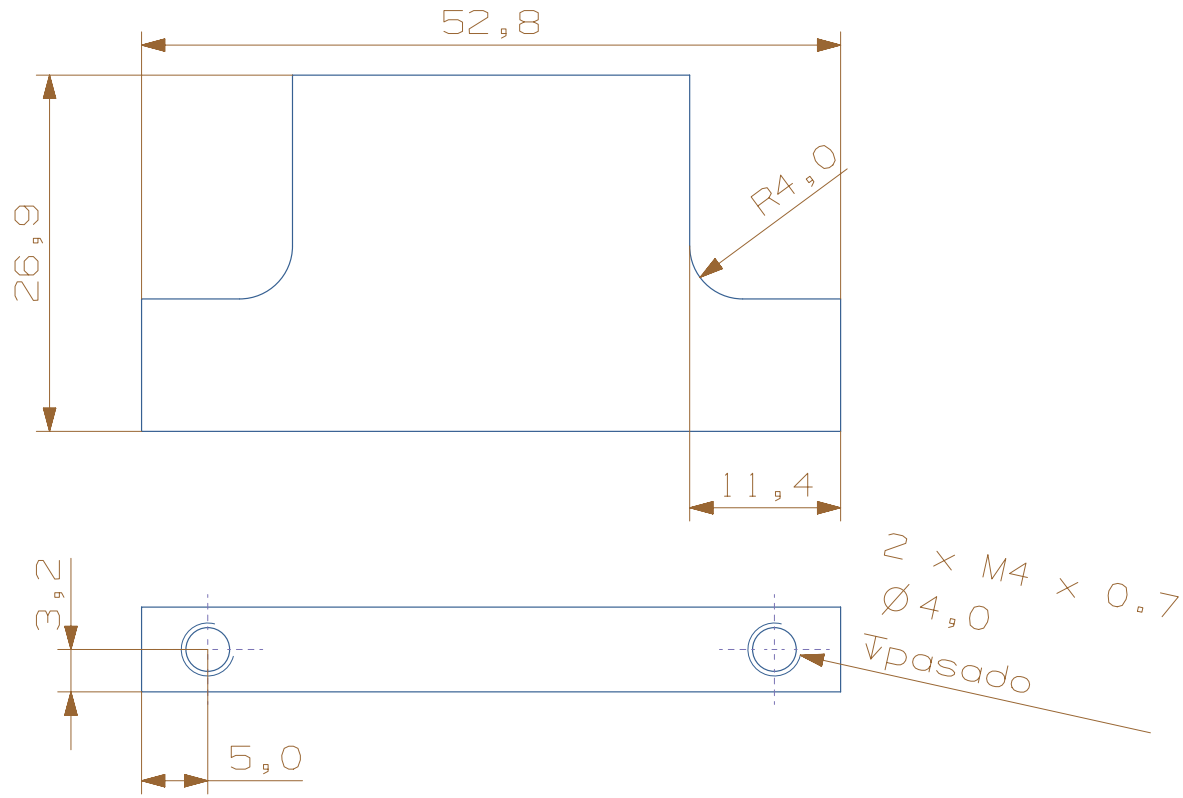
FACULTAD DE INGENIERIA

ROBOT ESFERICO

TITLE

PLACA SOPORTE MOTOR VERTICAL

DRAWN BY	JLAS	SIZE	FILE NAME		REV
CHECKED BY	-	A3	placa_soporte_motor_vert		
APPROVED BY	-	SCALE 1:1	SHEET 1 OF 1	PLANO MEC-026	



UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

ROBOT ESFERICO

TITLE

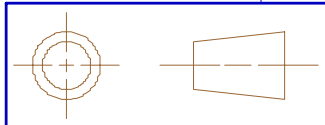
PLACA SOPORTE VOLANTE DELANTERA/TRASERA INFERIOR

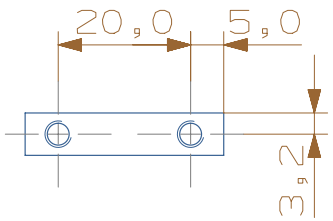
DRAWN BY	JLAS
CHECKED BY	-
APPROVED BY	-

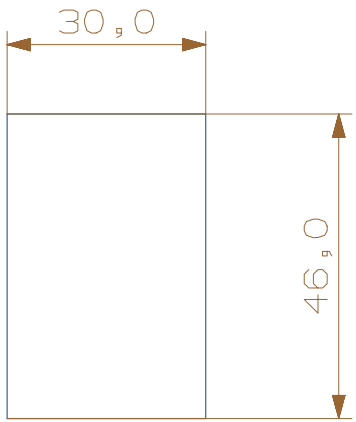
SIZE	FILE NAME	REV
A4	placa_soporte_volante_del_tras_inf	

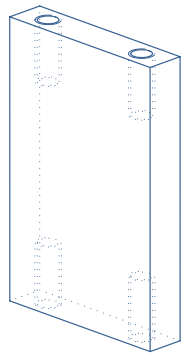
SCALE 2:1	SHEET 1 OF 1	PLANO MEC-027

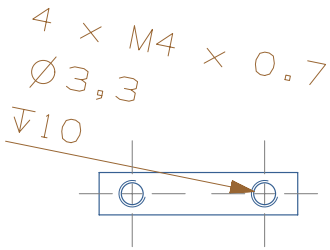
ALL DIMENSIONS IN MM











ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

ROBOT ESFERICO

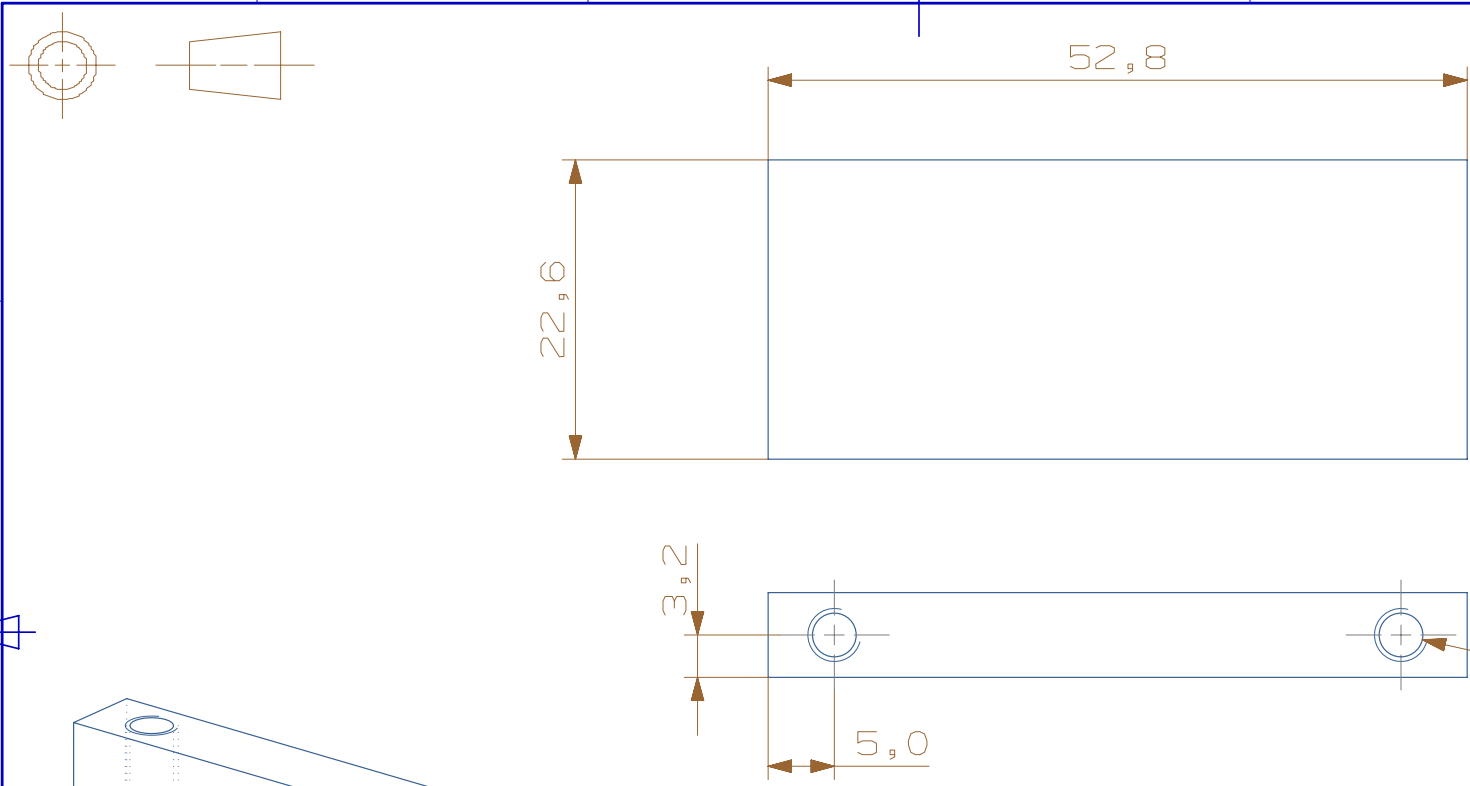
TITLE

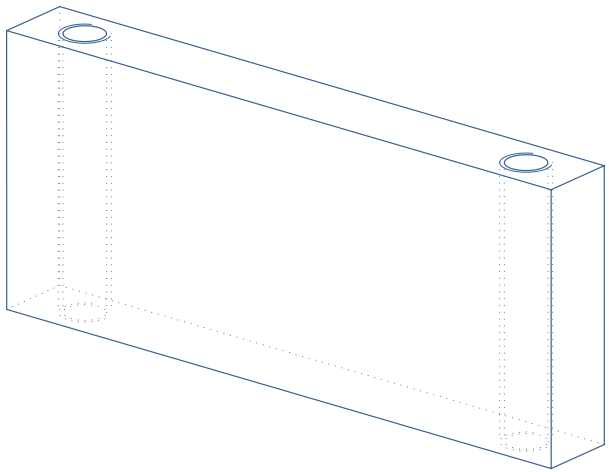
PLACA SOPORTE VOLANTE DELANTERA/TRASERA MEDIA

DRAWN BY	JLAS
CHECKED BY	-
APPROVED BY	-

SIZE	FILE NAME	REV
A4	placa_soporte_volante_del_tras_med	

SCALE 1:1	SHEET 1 OF 1	PLANO MEC-028





ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

ROBOT ESFERICO

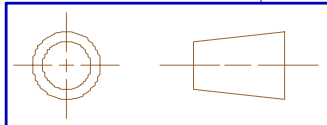
TITLE

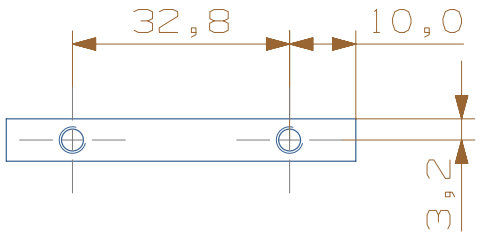
PLACA SOPORTE VOLANTE DELANTERA/TRASERA SUPERIOR

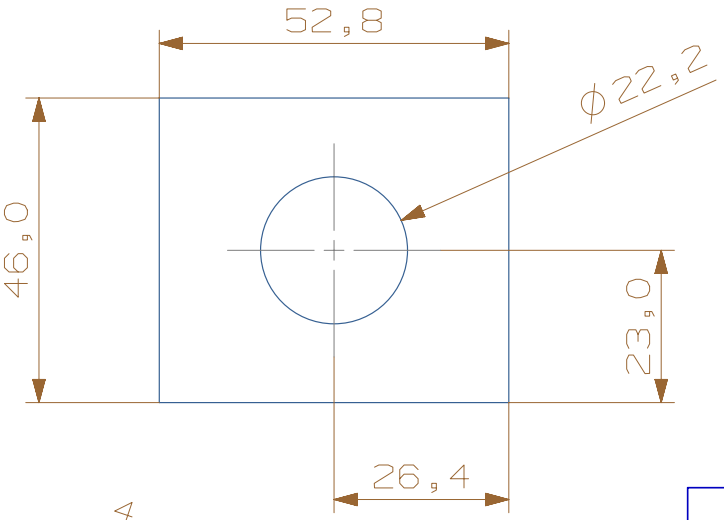
DRAWN BY	JLAS
CHECKED BY	-
APPROVED BY	-

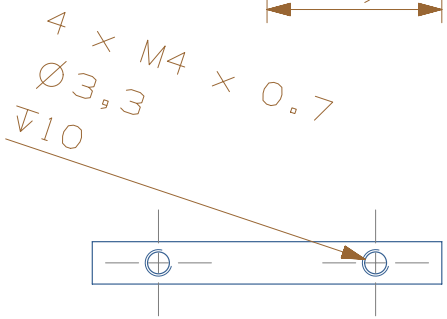
SIZE	FILE NAME	REV
A4	placa_soporte_volante_del_tras_sup	

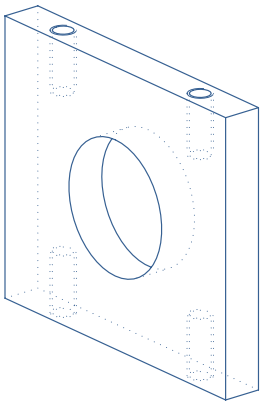
SCALE 2:1	SHEET 1 OF 1	PLANO MEC-029











ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

ROBOT ESFERICO

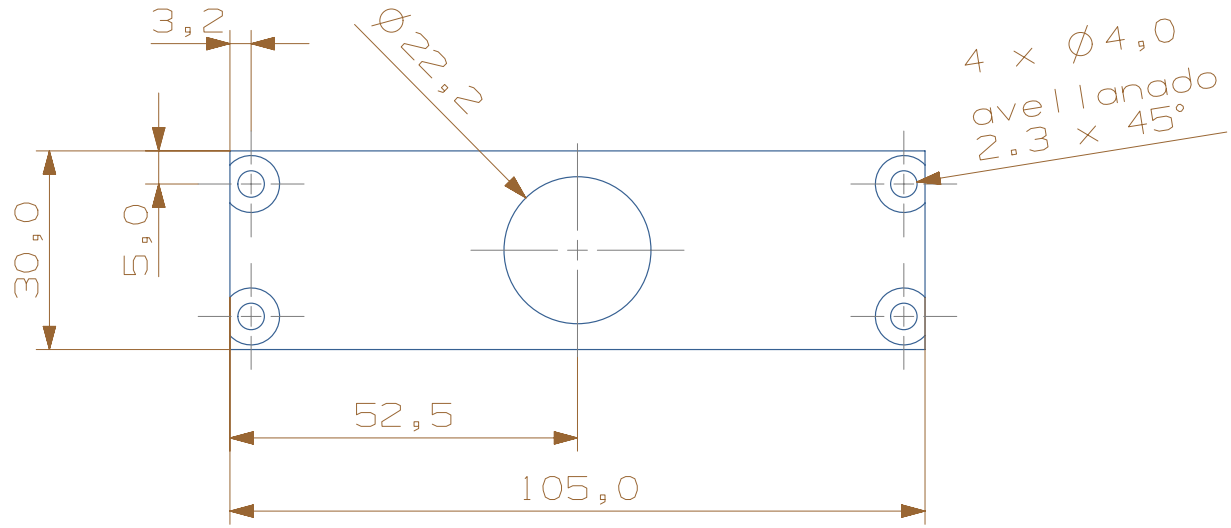
TITLE

PLACA SOPORTE VOLANTE EXTERIOR

DRAWN BY	JLAS
CHECKED BY	-
APPROVED BY	-

SIZE	FILE NAME	REV
A4	placa_soporte_volante_ext	

SCALE 1:1	SHEET 1 OF 1	PLANO MEC-030



UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

ROBOT ESFERICO

TITLE

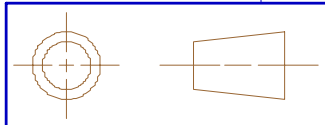
PLACA SOPORTE VOLANTE INFERIOR

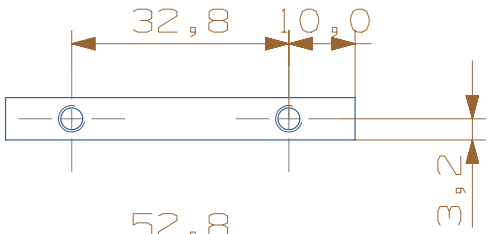
DRAWN BY	JLAS
CHECKED BY	-
APPROVED BY	-

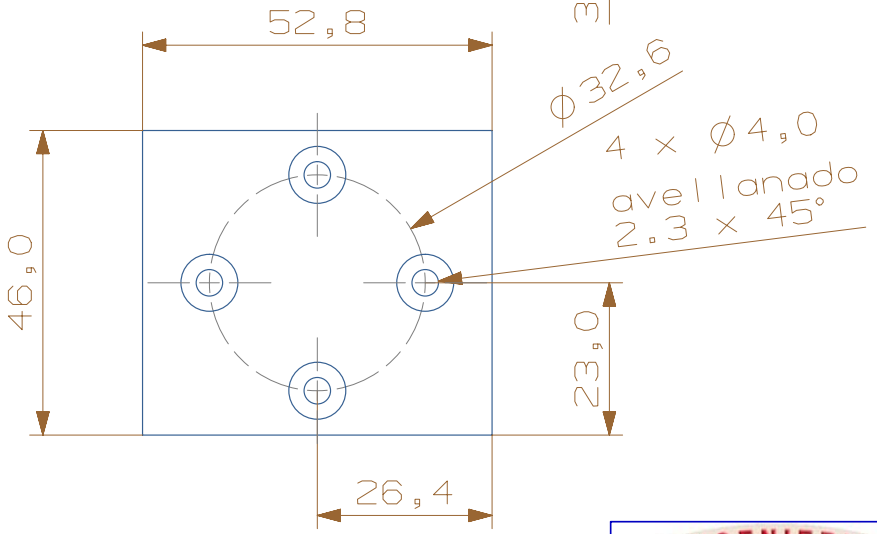
SIZE	FILE NAME	REV
A4	placa_soporte_volante_inf	

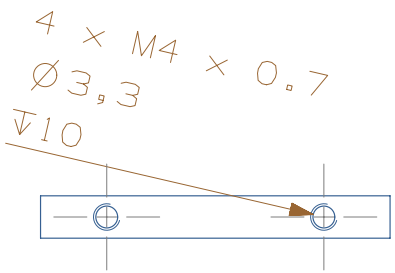
SCALE 1:1 | SHEET 1 OF 1 | PLANO MEC-031

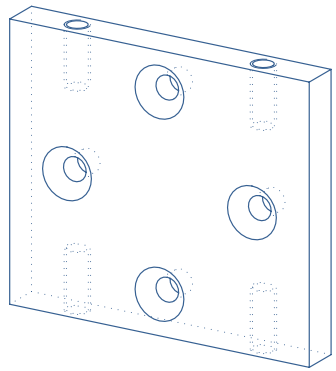
DIMENSIONES EN MM











ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

ROBOT ESFERICO

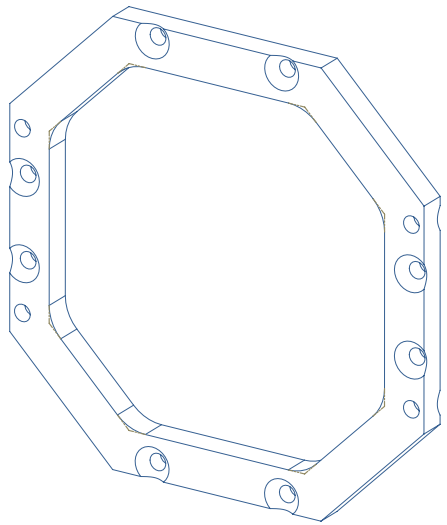
TITLE

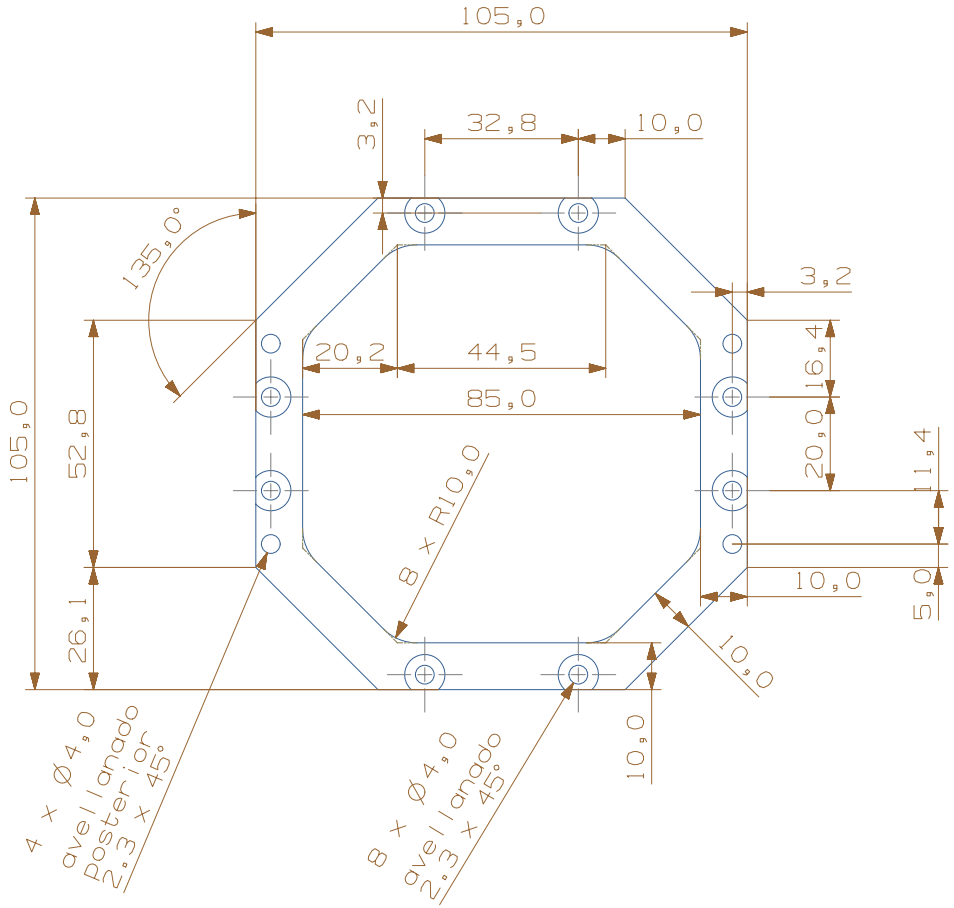
PLACA SOPORTE VOLANTE INTERIOR

DRAWN BY	JLAS
CHECKED BY	-
APPROVED BY	-

SIZE	FILE NAME	REV
A4	placa_soporte_volante_int	

SCALE 1:1	SHEET 1 OF 1	PLANO MEC-032





ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

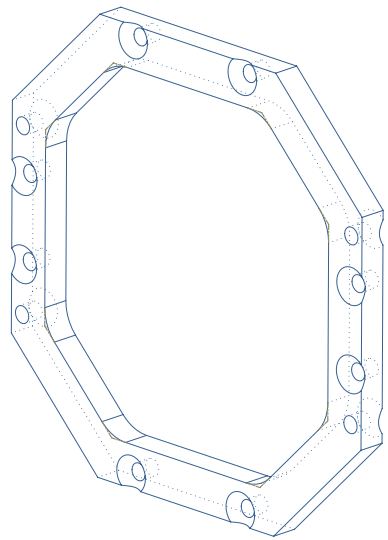
FACULTAD DE INGENIERIA

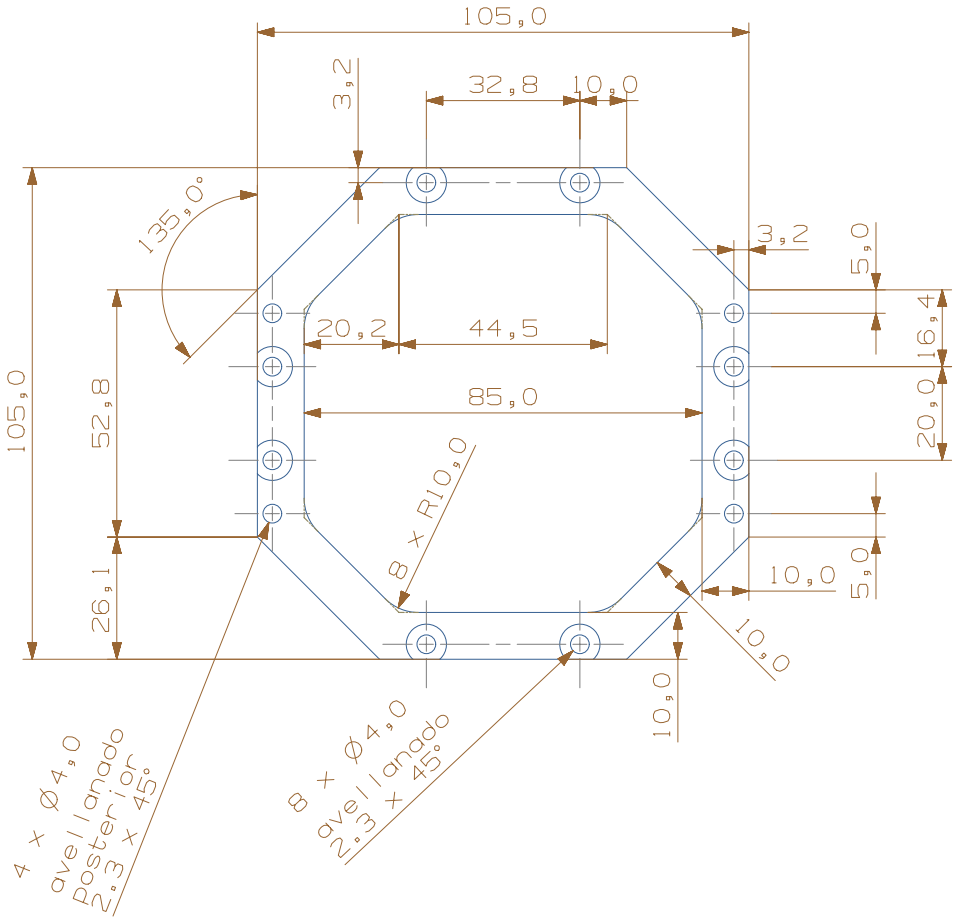
ROBOT ESFERICO

TITLE

PLACA SOPORTE VOLANTE
MEDIA INFERIOR

DRAWN BY	JLAS	SIZE	FILE NAME	REV
CHECKED BY	-	A3	placa_soporte_volante_med_inf	
APPROVED BY	-	SCALE 1:1	SHEET 1 OF 1	PLANO MEC-033





ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

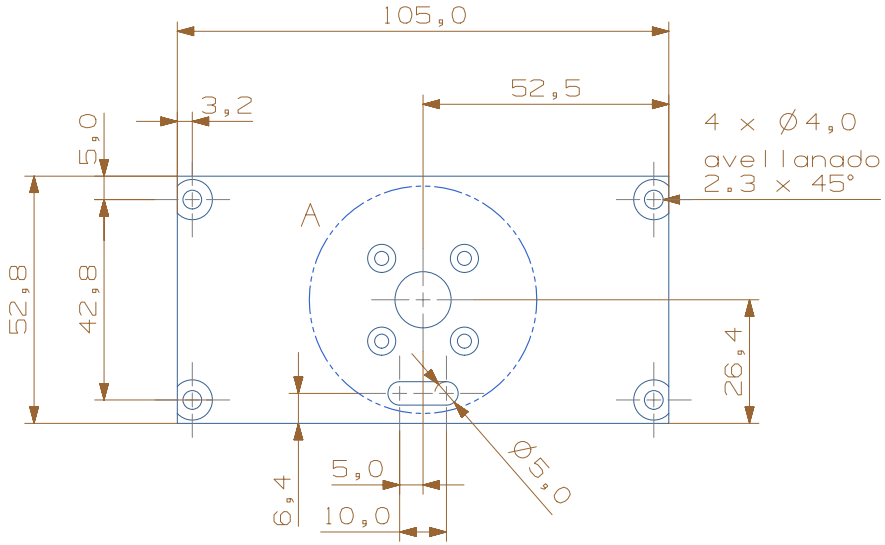
FACULTAD DE INGENIERIA

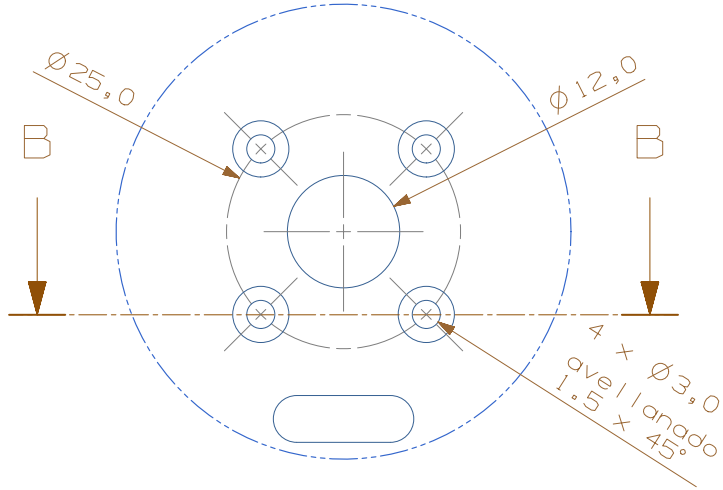
ROBOT ESFERICO

TITLE

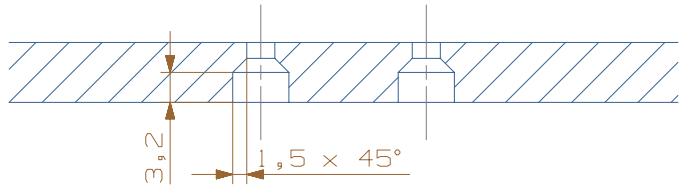
PLACA SOPORTE VOLANTE
MEDIA SUPERIOR

DRAWN BY	JLAS	SIZE	FILE NAME	REV
CHECKED BY	-	A3	placa_soporte_volante_med_sup	
APPROVED BY	-	SCALE 1:1	SHEET 1 OF 1	PLANO MEC-034

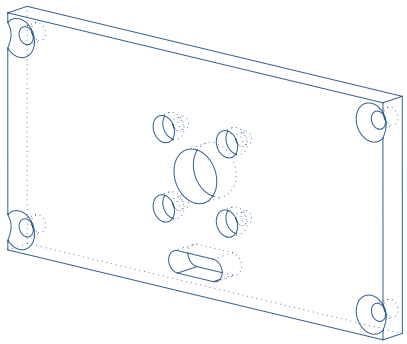




DETAIL A
SCALE 2:1



SECTION B-B



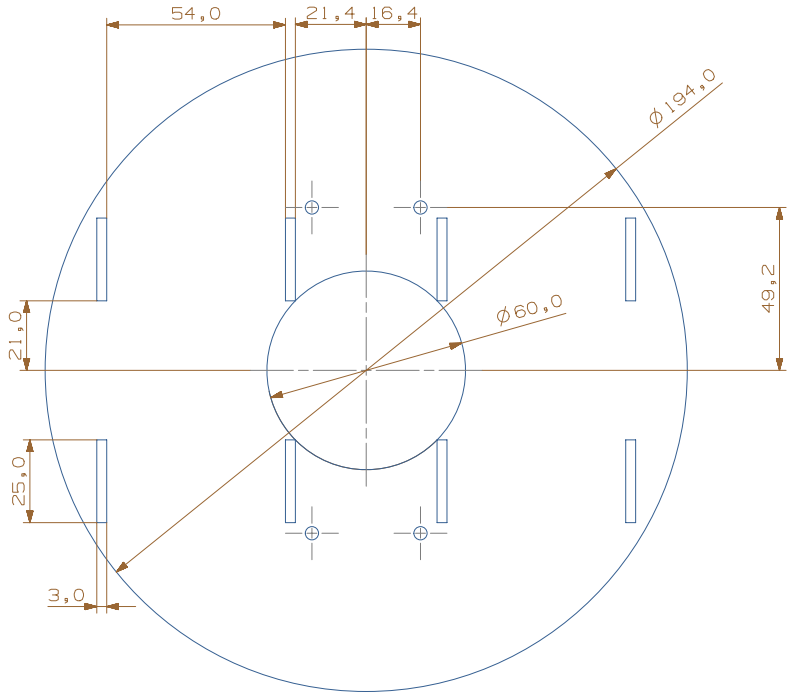
ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO
FACULTAD DE INGENIERIA

ROBOT ESFERICO

TITLE
PLACA SOPORTE VOLANTE SUPERIOR

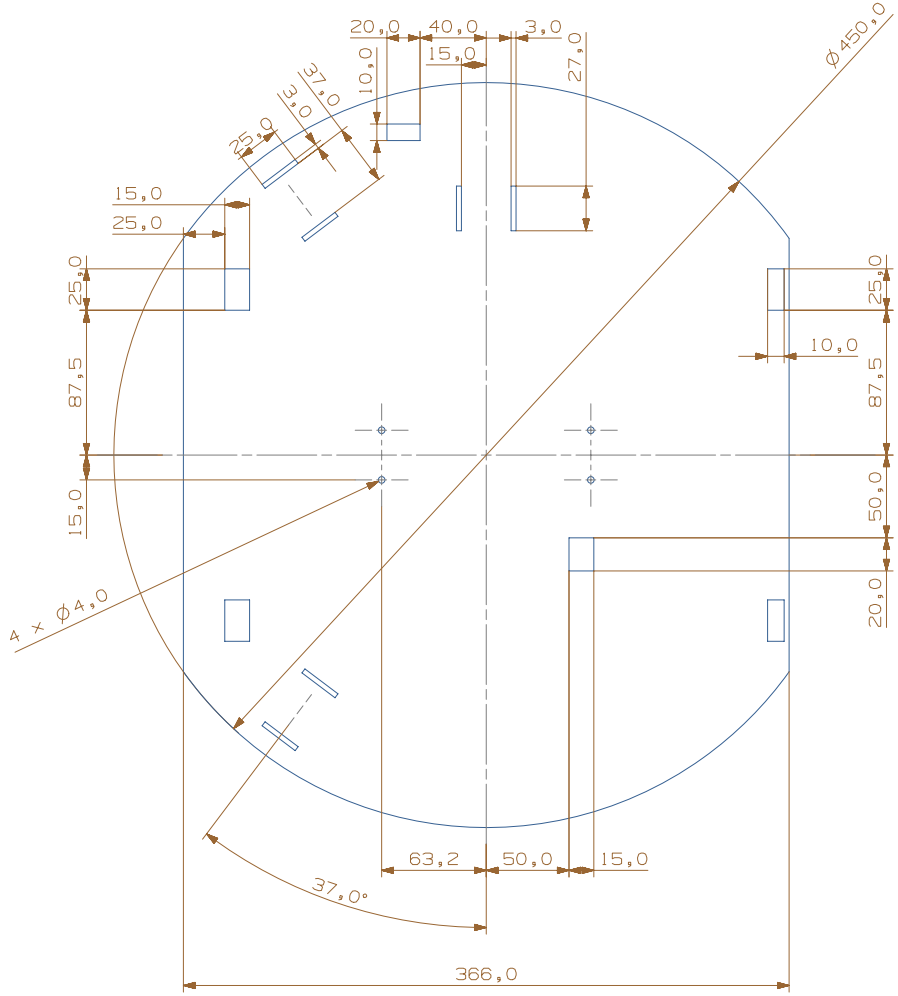
DRAWN BY	JLAS	SIZE	FILE NAME		REV
CHECKED BY	-	A3	placa_soporte_volante_sup		
APPROVED BY	-	SCALE 1:1	SHEET 1 OF 1	PLANO MEC-035	



lamina de acrilico de 6mm

ALL DIMENSIONS IN MM

		UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO			
		FACULTAD DE INGENIERIA			
		ROBOT ESFERICO			
		TITLE: PLACA MONTAJE INFERIOR			
DRAWN BY	JLAS	SIZE	FILE NAME		REV
CHECKED BY	-	A2	placa_montaje_inf		Ø
APPROVED BY	-	SCALE 1:1	SHEET 1 OF 1	PLANO MEC-036	

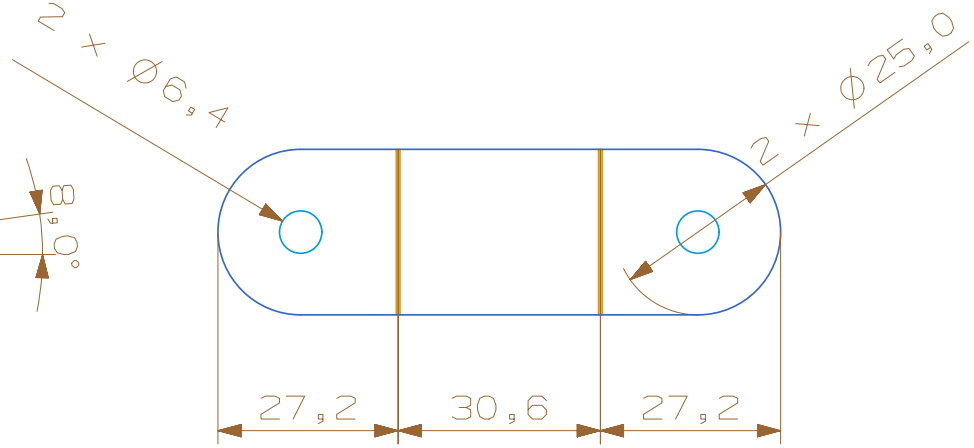


lamina de acrilico de 6mm

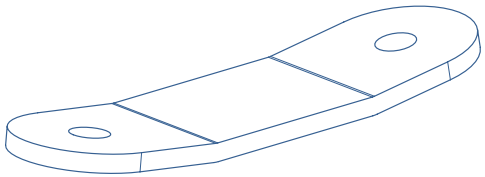
ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO					
FACULTAD DE INGENIERIA					
ROBOT ESFERICO					
TITLE: PLACA MONTAJE SUPERIOR					
DRAWN BY	JLAS	SIZE	FILE NAME		REV
CHECKED BY	-	A2	placa_montaje_sup		
APPROVED BY	-	SCALE 1:2	SHEET 1 OF 1	PLANO MEC-037	

lamina de acero calibre 12



placa desdoblada



ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

ROBOT ESFERICO

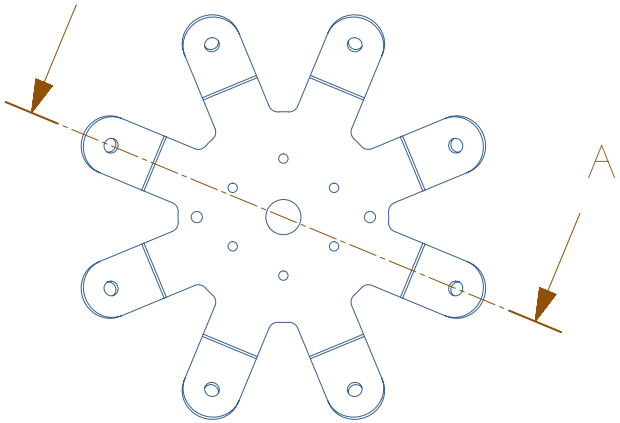
TITLE

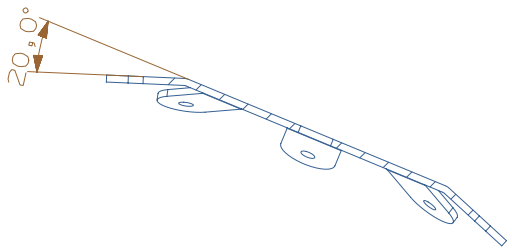
PLACA UNION ESFERA

DRAWN BY	JLAS
CHECKED BY	-
APPROVED BY	-

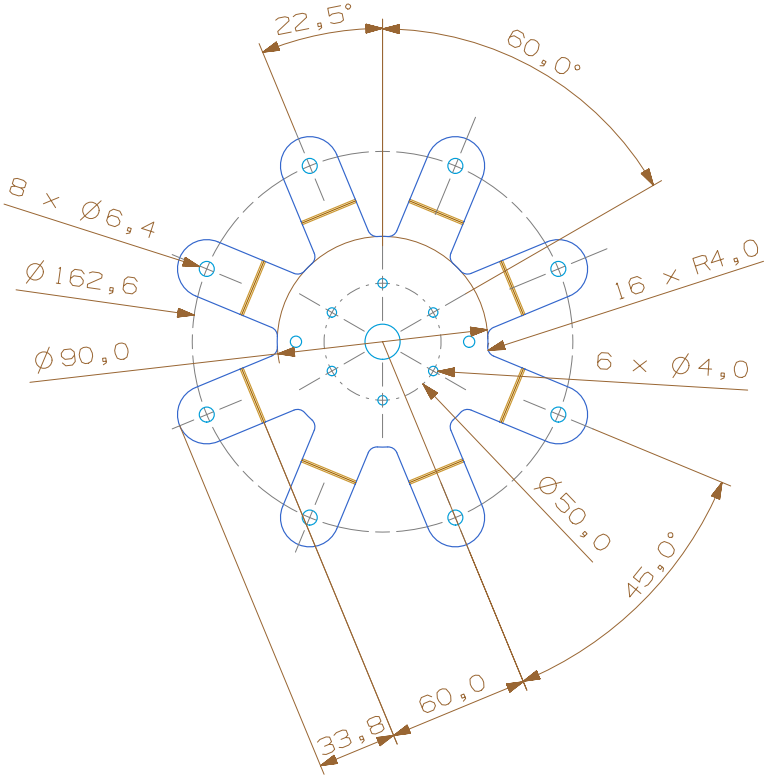
SIZE	FILE NAME	REV
A4	placa_union_esfera	

SCALE 1:1	SHEET 1 OF 1	PLANO MEC-038

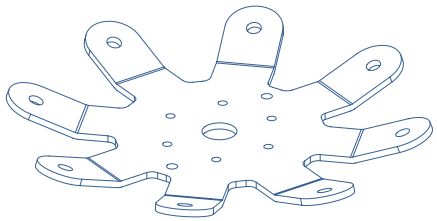




SECTION A-A



lamina de acero calibre 12

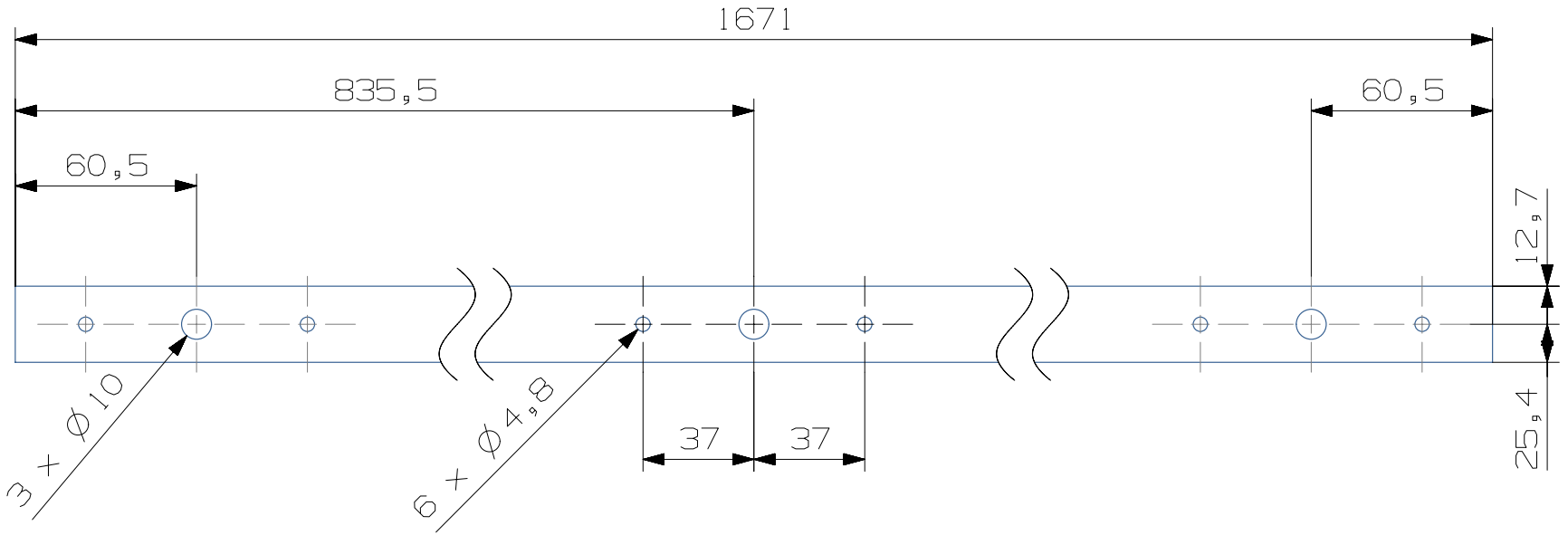


ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO
FACULTAD DE INGENIERIA
ROBOT ESFERICO

TITLE
PLACA UNION ESFERA LATERAL

DRAWN BY	JLAS	SIZE	FILE NAME		REV
CHECKED BY	-	A3	placa_union_esfera_lat		
APPROVED BY	-	SCALE 1:2	SHEET 1 OF 1	PLANO MEC-039	



lamina flexible de
acero templado calibre 22

ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

ROBOT ESFERICO

TITLE

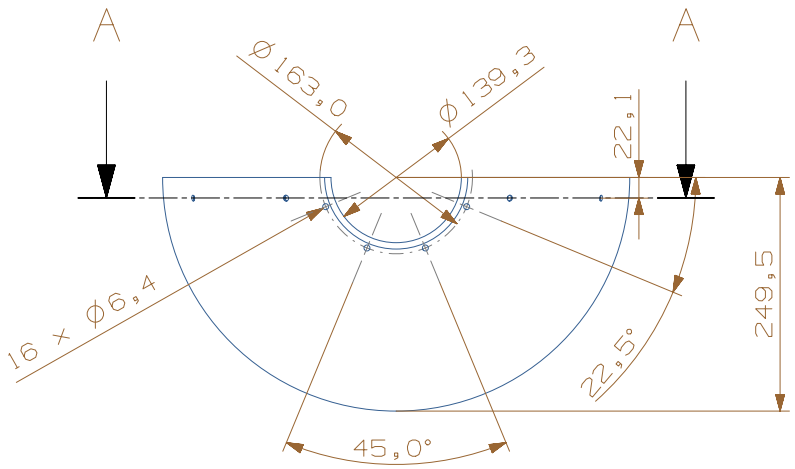
ANILLO

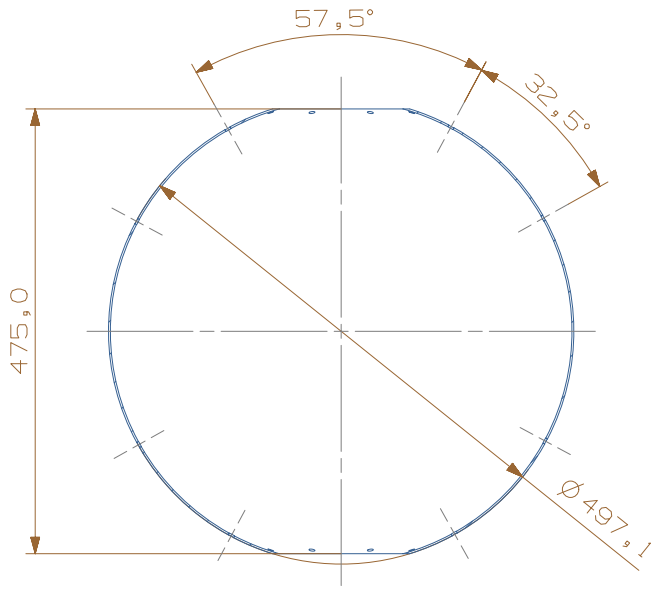
DRAWN BY	JLAS
CHECKED BY	-
APPROVED BY	-

SIZE	FILE NAME	REV
A4	anillo	

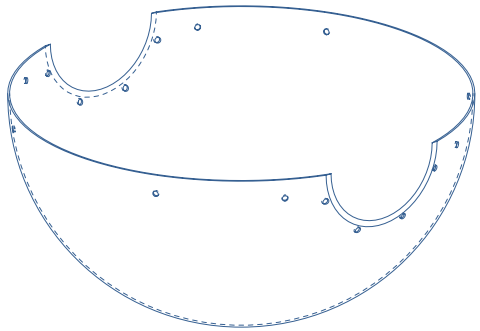
SCALE 1:1 | SHEET 1 OF 1 | PLANO MEC-040

perforaciones radiales





SECTION A-A

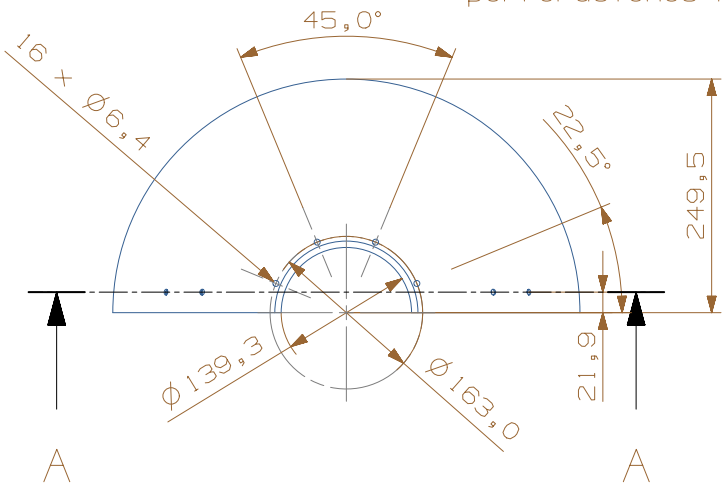


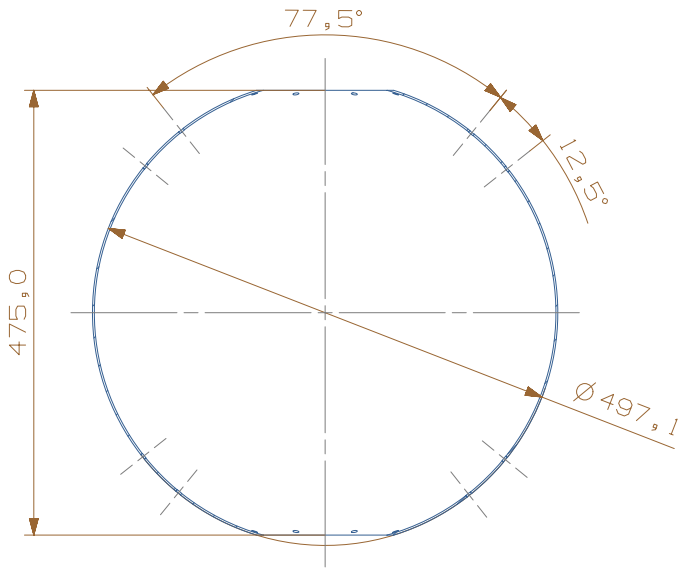
esfera de policarbonato
transparente de 2mm de espesor

ALL DIMENSIONS IN MM

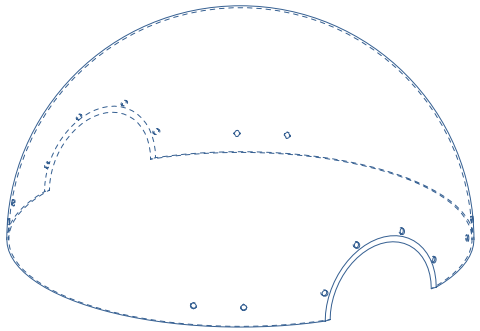
		UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO			
		FACULTAD DE INGENIERIA			
		ROBOT ESFERICO			
		TITLE: SEMIESFERA INFERIOR			
DRAWN BY	JLAS	SIZE A3	FILE NAME semiesfera_inf		REV
CHECKED BY	-				
APPROVED BY	-	SCALE 1:5	SHEET 1 OF 1	PLANO MEC-041	

perforaciones radiales





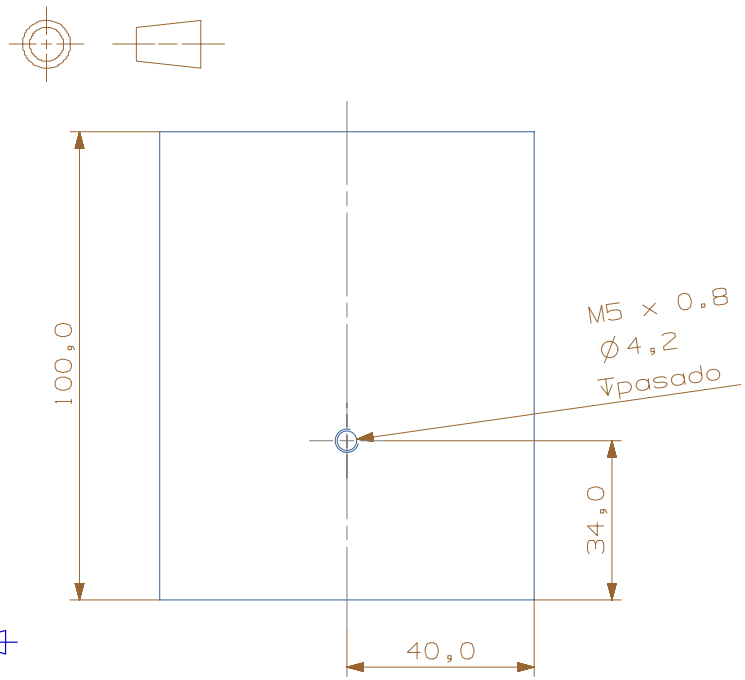
SECTION A-A

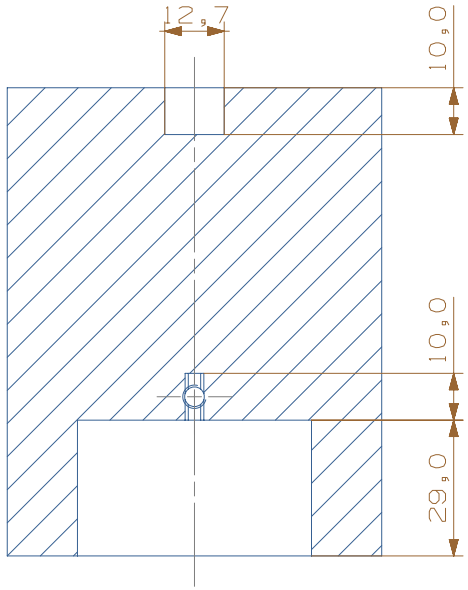


esfera de policarbonato
transparente de 2mm de espesor

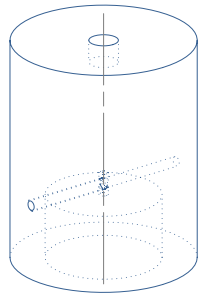
ALL DIMENSIONS IN MM

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO					
FACULTAD DE INGENIERIA					
ROBOT ESFERICO					
TITLE: SEMIESFERA SUPERIOR					
DRAWN BY	JLAS	SIZE A3	FILE NAME semiesfera_sup		REV
CHECKED BY	-				
APPROVED BY	-	SCALE 1:5	SHEET 1 OF 1	PLANO MEC-042	

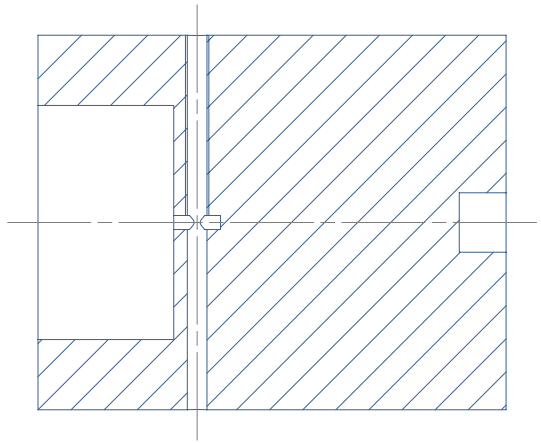




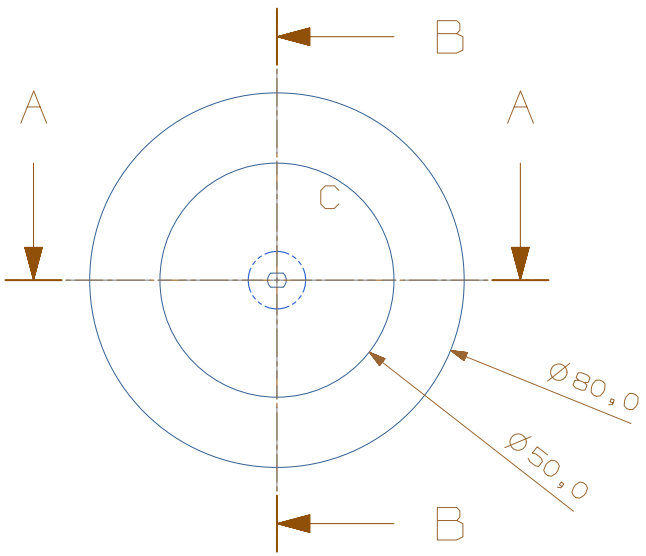
SECTION A-A

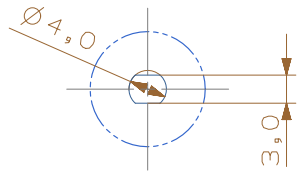


SCALE 1:2



SECTION B-B





DETAIL C
SCALE 2:1

ALL DIMENSIONS IN MM

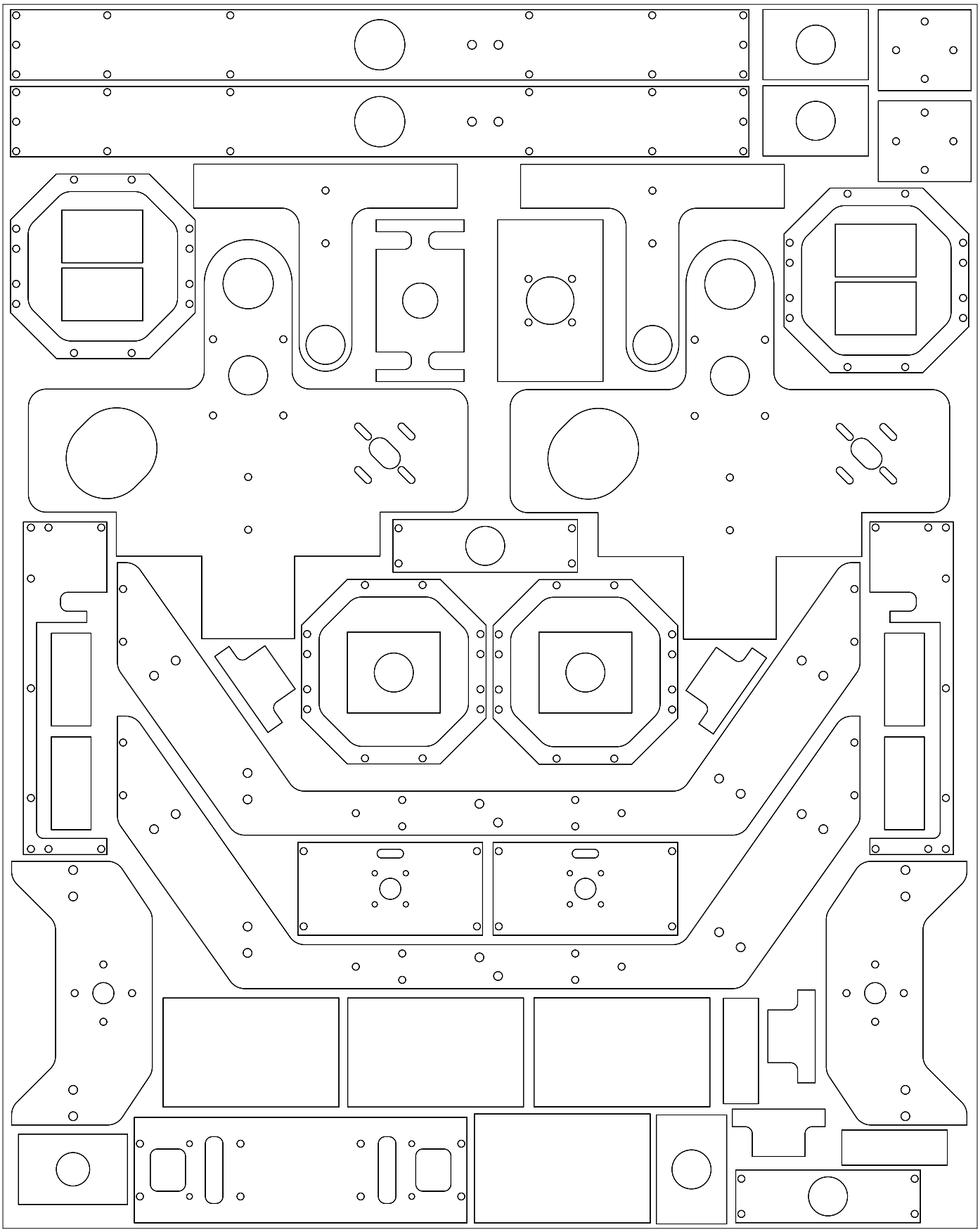
UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO
FACULTAD DE INGENIERIA

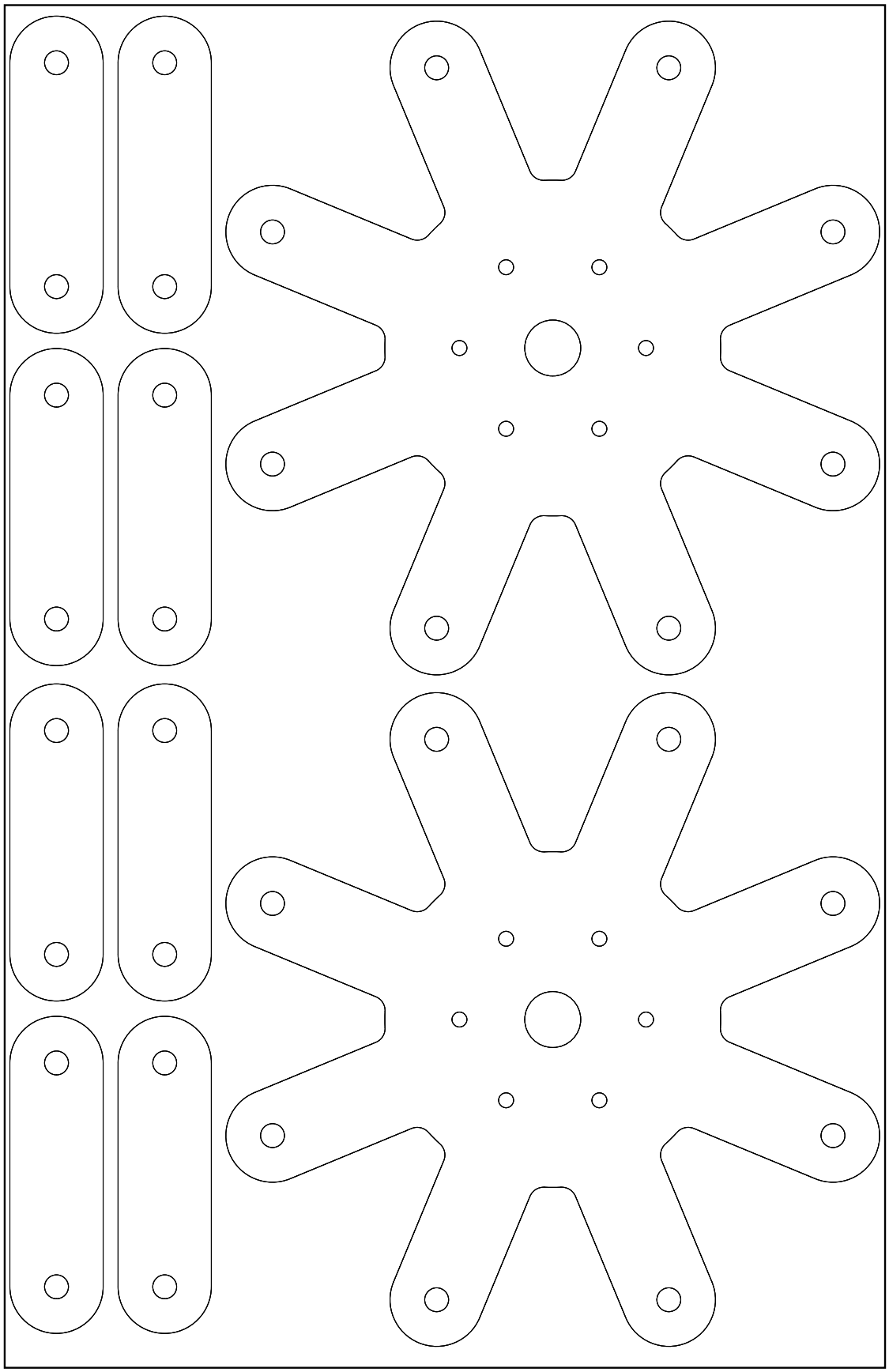
ROBOT ESFERICO

TITLE
VOLANTE DE INERCIA

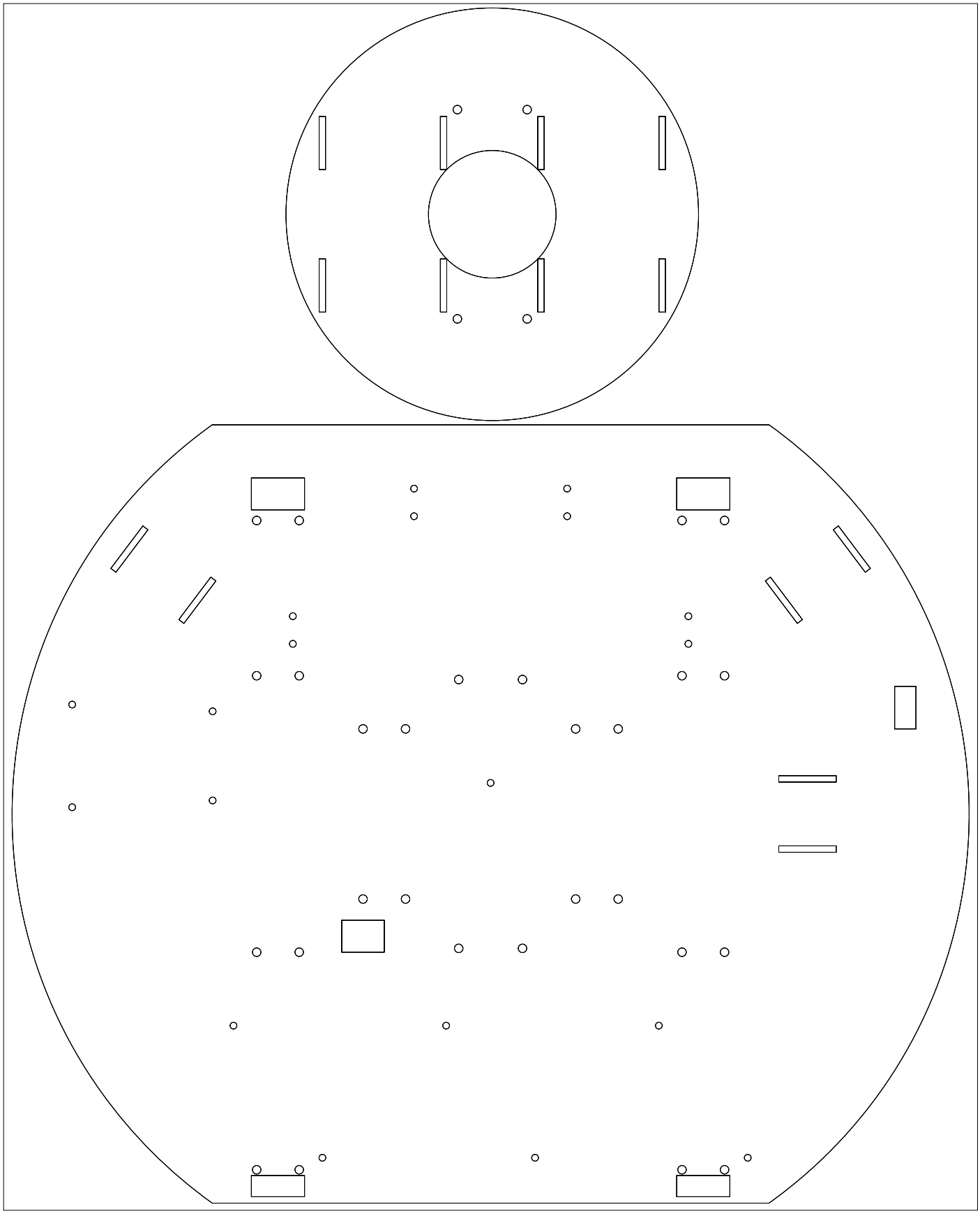
DRAWN BY	JLAS	SIZE	FILE NAME		REV
CHECKED BY	-	A3	volante		☐
APPROVED BY	-	SCALE 1:1	SHEET 1 OF 1	PLANO MEC-043	

B.3 Planos para corte con chorro de agua





B.4 Planos para corte con láser



Apéndice C

Listados de código

C.1 Código de modelado y control

C.1.1 Modelado

El modelado matemático del robot se realizó en Mathematica, por sus avanzadas capacidades de manipulación simbólica. El código de esta sección se puede ejecutar en Mathematica 9 o 10, y se encuentra disponible en repositorio del proyecto (a abril del 2015): `https://github.com/eigendreams/esferico`. El ejemplo mostrado es una plantilla usada para todas las simulaciones y experimentos, y no se detallan los cambios individuales realizados para cada posible variación presentada en otros capítulos. Se modela solamente la configuración del sistema consistente de un péndulo motor con dos volantes de inercia acoplados y usados como CMG, con condiciones iniciales todas nulas y entradas también nulas. Otras configuraciones también se han modelado y se incluyen en el repositorio. El modelado es procedural, y la adaptación a otros sistemas carentes de bucles cinemáticos es relativamente sencilla.

```
(*Consideramos que en el sistema tenemos solamente cuatro cuerpos, la
esfera, el pendulo, y dos volantes de inercia, quiza con posibles
distancias entre los centros de giro y los centro de masa de cada cuerpo, y
con un sistema de referencia acoplado al centro de masa de cada cuerpo, el
metodo de modelado lagrangiano intenta obtener primero las energias
cineticas y potenciales de todos los cuerpos, y usar sus expresiones
simbolicas para derivar las ecuaciones diferenciales del sistema, en el
caso de la energia cinetica, en cuerpos rigidos podemos distinguir dos
tipos de energias mecanicas, la rotacional y la traslacional*)
ClearAll["Global`*"]

(*Representaremos TODAS las rotaciones en el sistema a traves de matrices
de rotacion con argumento simbolicos. Si bien esto es molesto, nos da
varias ventajas, transformar entre coordenadas entre sistemas es trivial, y
la obtencion del vector de velocidad angular es una operacion sistematica.
Como desventaja, existen ambiguedades bien conocidas en el uso de matrices
```

```
de rotacion, en particular, la "direccion" en que las matrices transforman,
y el "orden" en que las matrices se multiplican para componer una rotacion
general consistente de multiples rotaciones parciales

Estas matrices de transformacion son las estandares usadas en matematicas,
para expresar rotaciones de vectores COLUMNA, en Mathematica, especificamos
funciones que quiza puedan tomar argumentos con la sintaxis de un guion
bajo luego del argumento, y el operador dos punto igual, o de evaluacion
demorada*)
Rx[\[Theta]_] := ( {
    {1, 0, 0},
    {0, Cos[\[Theta]], -Sin[\[Theta]]},
    {0, Sin[\[Theta]], Cos[\[Theta]]} } );
Ry[\[Theta]_] := ( {
    {Cos[\[Theta]], 0, Sin[\[Theta]]},
    {0, 1, 0},
    {-Sin[\[Theta]], 0, Cos[\[Theta]]} } );
Rz[\[Theta]_] := ( {
    {Cos[\[Theta]], -Sin[\[Theta]], 0},
    {Sin[\[Theta]], Cos[\[Theta]], 0},
    {0, 0, 1} } );

(*La funcion siguiente calcula el tensor de velocidad angular (GetWx) y de
este podemos obtener el vector de velocidad angular devolviendo algunos de
sus componentes en ubicaciones especificas con la funcion GetW, dada cierta
matriz de rotacion simbolica que describa las rotaciones que lleven de un
sistema inercial O a un sistema final P, y, dependiendo de la matriz de
rotacion usada, quiza obteniendo la velocidad angular de P en terminos de
O, o viceversa, diremos que \[Omega] es el prefijo de las velocidades
expresadas en terminos de O, en tanto \[CapitalOmega] es el prefijo de las
velocidades expresadas en terminos de P en lo sucesivo, donde O es el
sistema inercial, y P el sistema acoplado al cuerpo

Las velocidades angulares son importantes, porque se necesitan para
determinar las energias rotacionales de los cuerpos*)
GetWx[Rmt_] := D[Rmt, t].Transpose[Rmt];
GetW[Rmt_] := {{GetWx[Rmt][[3]][[2]]}, {GetWx[Rmt][[1]][[3]]}, {GetWx[Rmt]
    [[2]][[1]]}};

(*Rotaciones de la esfera*)
EsfPrimRot = Rz[\[Psi]sph[t]];
EsfSecRot = Ry[\[Theta]sph[t]];
EsfTerRot = Rx[\[Phi]sph[t]];

(*Rotaciones del pendulo*)
PendSecRot = Ry[\[Alpha]pend[t]];
PendPrimRot = Rx[\[Beta]pend[t]];

(*Primera rotacion al primer CMG de inercia desde el sistema del pendulo*)
Vol1Rot1 = Ry[\[CurlyPhi]1[t]];

(*Primera rotacion al segundo CMG de inercia desde el sistema del pendulo*)
Vol2Rot1 = Ry[\[CurlyPhi]2[t]];

(*Segunda rotacion del primer volante de inercia desde el sistema del CMG*)
Vol1Rot2 = Rz[\[Gamma]1[t]];

(*Segunda rotacion del segundo volante de inercia desde el sistema del
CMG*)
Vol2Rot2 = Rz[\[Gamma]2[t]];

(*Las rotaciones se composicionan en el orden inverso al cual se declara
que suceden. Esto tiene sentido, porque podemos pensar que el orden de
rotaciones, digamos, para el pendulo, podria haber sido
RPend=PenSecRot.PendPrimRot.EsfTerRot.EsfSecRot.EsfPrimRot

Pero hacer esto estaria mal, porque estariamos expresando siempre
rotaciones en torno a los ejes del sistema inercial O, cuando lo que
queremos es expresar composciones de rotaciones, cada una de ellas rotando
algun vector de posicion, con respecto al sistema acoplado al ultimo
cuerpo, y asi sucesivamente. Por un artilugio matematico, es posible
demostrar que la forma de expresar esta rotacion se consigue invirtiendo el
orden en que las rotaciones parciales se aplican*)
REsf = EsfPrimRot.EsfSecRot.EsfTerRot;
RPend = REsf.PendPrimRot.PendSecRot;
RVol1 = RPend.Vol1Rot1.Vol1Rot2;
RVol2 = RPend.Vol2Rot1.Vol2Rot2;

(*Delta de posicion del centro de giro de la esfera al centro de masa del
pendulo*)
DVecPend = RPend.{{0}, {0}, {-radioPendulo}};
```

```
(*Delta de posicion del centro de giro de la esfera al centro de masa de la
esfera*)
DVecEsf = REsf.{{0}, {0}, {offsetEsf}};

(*Delta de posicion del centro de giro de la esfera al centro de masa del
volante 1*)
DVecVol1 = RPend.{{0}, {-radioVolante}, {-offsetVol}};

(*Delta de posicion del centro de giro de la esfera al centro de masa del
volante 2*)
DVecVol2 = RPend.{{0}, {radioVolante}, {-offsetVol}};

(*Velocidad angular de O en terminos del sistema de la esfera, y del
sistema de la esfera en O*)
\[CapitalOmega]Esf = -GetW[Transpose[REsf]] // Simplify;
\[Omega]Esf = GetW[REsf] // Simplify;

(*Lo mismo para el pendulo*)
\[CapitalOmega]Pend = -GetW[Transpose[RPend]] // Simplify;
\[Omega]Pend = GetW[RPend] // Simplify;

(*Volante 1*)
\[CapitalOmega]Vol1 = -GetW[Transpose[RVol1]] // Simplify;
\[Omega]Vol1 = GetW[RVol1] // Simplify;

(*Volante 2*)
\[CapitalOmega]Vol2 = -GetW[Transpose[RVol2]] // Simplify;
\[Omega]Vol2 = GetW[RVol2] // Simplify;

(*Tensores de inercia de los cuerpos del sistema, se asume que existen
solamente tres clases de cuerpos, la esfera, el pendulo motriz, y cada
volante de inercia. Esta suposicion en razonable porque estas clases de
cuerpos engloban TODOS los grados de libertad rotacionales del sistema.
Estos tensores de usan para obtener las energia cineticas rotacionales (que
toman por parametro las inercias angulares).

Los tensores de inercia deben ser todos expresados con respecto al CENTRO
DE MASA de cada cuerpo, la posible aplicacion del teorema de los ejes
paralelos, donde tenga sentido, se incluye implicitamente en las
expresiones de energia traslacional (que toman por parametro la masa) de
los cuerpos en rotacion. Como los cuerpos se asumen simetricos al menos
sobre dos planos, los tensores son todos diagonales.

Para evitar transformar los tensores de inercia a otros sistemas de
referencia, la velocidad que se usa en el calculo de la energia rotacional
es la de O en terminos del sistema acoplado al cuerpo analizado*)
InertiaPendulo = ( {
    {IxxP, 0, 0},
    {0, IyyP, 0},
    {0, 0, IzzP} } );
InertiaSph = ( {
    {IxxS, 0, 0},
    {0, IyyS, 0},
    {0, 0, IzzS} } );
InertiaVolante = ( {
    {IxxV, 0, 0},
    {0, IyyV, 0},
    {0, 0, IzzV} } );

(*Energia rotacional de la esfera*)
EROTESF = 0.5*(Transpose[\[CapitalOmega]Esf].InertiaSph.\[CapitalOmega]Esf)
    [[1]][[1]];

(*Energia rotacional del pendulo*)
EROTPEND = 0.5*(Transpose[\[CapitalOmega]Pend].InertiaPendulo.\
    [CapitalOmega]\Pend)[[1]][[1]];

(*Energias rotacionales de los volantes*)
EROTVOL1 = 0.5*(Transpose[\[CapitalOmega]Vol1].InertiaVolante.\
    [CapitalOmega]\Vol1)[[1]][[1]];
EROTVOL2 = 0.5*(Transpose[\[CapitalOmega]Vol2].InertiaVolante.\
    [CapitalOmega]\Vol2)[[1]][[1]];

(*Posicion del centro GEOMETRICO de la esfera, y luego de su centro de
masa*)
VecEsfera = {{x[t]}, {y[t]}, {radioEsfera}};
VecEsferaMasa = VecEsfera + DVecEsf;

(*Energia cinetica de la esfera, la posicion usada es la del centro de
masa*)
VelEsfera = D[VecEsferaMasa, t];
ETRASESF = (0.5*mEsf*Transpose[VelEsfera].VelEsfera)[[1]][[1]];
```

```
(*Energia cinetica del pendulo, igualmente, debe expresarse en termino del
centro de masa de la esfera, pero en este caso, se parte desde el CENTRO
GEOMETRICO de la esfera para llegar al centro de masa del pendulo*)
VecPend = VecEsfera + DVecPend;
VelPend = D[VecPend, t];
ETRASPEND = (0.5*mPend*Transpose[VelPend].VelPend)[[1]][[1]];

(*Energias cineticas de los volantes*)
VecVol1 = VecEsfera + DVecVol1;
VecVol2 = VecEsfera + DVecVol2;
VelVol1 = D[VecVol1, t];
VelVol2 = D[VecVol2, t];
ETRASVOL1 = (0.5*mVol*Transpose[VelVol1].VelVol1)[[1]][[1]];
ETRASVOL2 = (0.5*mVol*Transpose[VelVol2].VelVol2)[[1]][[1]];

(*Energias potenciales*)
gVector = Transpose[{{0}, {0}, {g}}];
EPOTESF    = (mEsf*gVector.VecEsferaMasa)[[1]][[1]];
EPOTPEND = (mPend*gVector.VecPend)[[1]][[1]];
EPOTVOL1 = (mVol*gVector.VecVol1)[[1]][[1]];
EPOTVOL2 = (mVol*gVector.VecVol2)[[1]][[1]];

(*Lagrangiano*)
T = EROTESF + EROTPEND + ETRASESF + ETRASPEND + EROTVOL1 + EROTVOL2 +
    ETRASVOL1 + ETRASVOL2;
V = EPOTESF + EPOTPEND + EPOTVOL1 + EPOTVOL2;
Lag = T - V // Chop;

(*Aplicacion de las ecuaciones de Euler Lagrange, parte izquierda*)
EULA1 = D[D[Lag, x'[t]], t] - D[Lag, x[t]] // Chop;
EULA2 = D[D[Lag, y'[t]], t] - D[Lag, y[t]] // Chop;
EULA3 = D[D[Lag, \[Psi]sph'[t]], t] - D[Lag, \[Psi]sph[t]] // Chop;
EULA4 = D[D[Lag, \[Theta]sph'[t]], t] - D[Lag, \[Theta]sph[t]] // Chop;
EULA5 = D[D[Lag, \[Phi]sph'[t]], t] - D[Lag, \[Phi]sph[t]] // Chop;
EULA6 = D[D[Lag, \[Alpha]pend'[t]], t] - D[Lag, \[Alpha]pend[t]] // Chop;
EULA7 = D[D[Lag, \[Beta]pend'[t]], t] - D[Lag, \[Beta]pend[t]] // Chop;
EULA8 = D[D[Lag, \[CurlyPhi]1'[t]], t] - D[Lag, \[CurlyPhi]1[t]] // Chop;
EULA9 = D[D[Lag, \[CurlyPhi]2'[t]], t] - D[Lag, \[CurlyPhi]2[t]] // Chop;
EULA10 = D[D[Lag, \[Gamma]1'[t]], t] - D[Lag, \[Gamma]1[t]] // Chop;
EULA11 = D[D[Lag, \[Gamma]2'[t]], t] - D[Lag, \[Gamma]2[t]] // Chop;

(*Restricciones, estas especifican una condicion de no deslizamiento, de
manera que podamos decir que entre la esfera y el suelo el contacto siempre
garantiza que el movimiento de la esfera en sus coordenadas lineales y el
que se obtien emultiplicando su radio por su velocidad angular sean siempre
iguales*)
Restriccion1 = (x'[t] - radioEsfera*(\[Omega]Esf[[2]][[1]]));
Restriccion2 = (y'[t] + radioEsfera*(\[Omega]Esf[[1]][[1]]));

(*Coeficientes de los multiplicadores de Lagrange, de los cuales habra
tantos como se tengan restricciones, sobre cada uno de los grados de
libertad del sistema, y especifican la proyeccion de la fuerza que
representa el multiplcador de lagrange sobre esa coordenada generalizada,
otra forma que tambien puede funcionar, paro no es tractable para el modelo
completo con el voltaje de los motores, habria sido especificar fricciones
viscosas con un coeficiente extremadamente alto, y asumir que el
multiplicador de lagrange es en cada caso igual a la fuerza de friccion
viscosa, que es el producto del coeficiente de friccion por la restriccion,
que no se asume ya necesariamente cero en todo momento, un coeficiente de
friccion que tienda a infinito brindaria el mismo comportamiento que el
obtenible con los multplicadores*)
a11 = D[Restriccion1, x'[t]];
a12 = D[Restriccion1, y'[t]];
a13 = D[Restriccion1, \[Psi]sph'[t]];
a14 = D[Restriccion1, \[Theta]sph'[t]];
a15 = D[Restriccion1, \[Phi]sph'[t]];
a16 = D[Restriccion1, \[Alpha]pend'[t]];
a17 = D[Restriccion1, \[Beta]pend'[t]];
a18 = D[Restriccion1, \[CurlyPhi]1'[t]];
a19 = D[Restriccion1, \[CurlyPhi]2'[t]];
a110 = D[Restriccion1, \[Gamma]1'[t]];
a111 = D[Restriccion1, \[Gamma]2'[t]];

a21 = D[Restriccion2, x'[t]];
a22 = D[Restriccion2, y'[t]];
a23 = D[Restriccion2, \[Psi]sph'[t]];
a24 = D[Restriccion2, \[Theta]sph'[t]];
a25 = D[Restriccion2, \[Phi]sph'[t]];
a26 = D[Restriccion2, \[Alpha]pend'[t]];
a27 = D[Restriccion2, \[Beta]pend'[t]];
a28 = D[Restriccion2, \[CurlyPhi]1'[t]];
```

```
a29 = D[Restriccion2, \[CurlyPhi]2'[t]];
a210 = D[Restriccion2, \[Gamma]1'[t]];
a211 = D[Restriccion2, \[Gamma]2'[t]];

(*Forma de expresar las restricciones como fricciones viscosas, se asume
que cada multiplicador de lagrange es igual a la restriccion por algun
coeficiente de friccion arbitrariamente alto, como la ecuacion de
restriccion no incluye mas terminos que los primeros cinco grados de
libertad, solo tiene sentido evaluar estos*)
\[Lambda]1v = fvis (Restriccion1);
\[Lambda]2v = fvis (Restriccion2);
fx = \[Lambda]1v*a11 + \[Lambda]2v*a21;
fy = \[Lambda]1v*a12 + \[Lambda]2v*a22;
f\[Psi] = \[Lambda]1v*a13 + \[Lambda]2v*a23;
f\[Theta] = \[Lambda]1v*a14 + \[Lambda]2v*a24;
f\[Phi] = \[Lambda]1v*a15 + \[Lambda]2v*a25;

(*sin embargo, para propositos del modelado lineal con el modelo del motor,
no consideraremos fricciones viscosas, y por ende, haremos uso de los
multiplciadores de lagrange de la forma usual*)
fx = 0;
fy = 0;
f\[Psi] = 0;
f\[Theta] = 0;
f\[Phi] = 0;

(*Modelos de los motores, en el caso del pendulo, en el caso de los
volantes, como no se pretende controlar de estos el angulo de forma precisa
(el angulo del cuerpo del volante, perpendicular al eje de giro), no se usa
ningun modelo y se asume que se usaran en lazo abierto de forma que se
respete alguna velocidad de rotacion de los volantes, en el caso de los
motores brushless que hacen girar a los volantes a la altisima velocidad
requerida para su funcionamiento, se asume que la considerable inercia y
los ESC son suficientes para regular la velocidad.

Se asume un modelo lineal para los motores, y de segundo orden una vez
conectado el motor con la inercia del sistema sobre las coordenadas
generalizadas pertinentes, para hallar k1mot y k2mot basta con considerar
que, a velocidad de rotacion cero, el torque de los motores debe ser el
indicado en la hoja de datos a rotor parado al voltaje nominal, por lo cual
Trotorparado = k1mot*Vnom

Y para determinar k2mot, basta con considerar que a la velocidad a rotor
libre, el torque de salida es cero, por lo que
0 = Vnom - k2mot*\[Omega]libre*)
TorquePenduloAlfa[t] = k1mot (VoltAlfa[t] - k2mot*\[Alpha]pend'[t]);
TorquePenduloBeta[t] = k1mot (VoltBeta[t] - k2mot*\[Beta]pend'[t]);

(*Fuerzas generalizadas sobre cada grado de libertad o coordenada
generalizada en el sistema, as adicionan fricciones viscosas sobre los
grados de libertad en los que esto tenga sentido, donde la fuerzas con f
minuscula son de friccion viscosa, bien con el aire (fEsfXAir, fEsfYAir),
bien por la rotacion de la esfera alrededor de Z (fEsfZ), bien las posibles
sustitituciones de las restricciones por fricciones viscoas (fx, fy,
f\[Psi], f\[Theta], f\[Phi]), o bien las fricciones de los motores del
pendulo, que son los unicos que se modelan distintos de fuentes perfectas
de torque (fPendA y fPendB)*)
F1 = \[Lambda]1[t]*a11 + \[Lambda]2[t]*a21 - fEsfXAir*x'[t] - fx;
F2 = \[Lambda]1[t]*a12 + \[Lambda]2[t]*a22 - fEsfYAir*y'[t] - fy;
F3 = \[Lambda]1[t]*a13 + \[Lambda]2[t]*a23 - fEsfZ*\[Omega]Esf[[3]][[1]] -
    f\[Psi];
F4 = \[Lambda]1[t]*a14 + \[Lambda]2[t]*a24 - f\[Theta];
F5 = \[Lambda]1[t]*a15 + \[Lambda]2[t]*a25 - f\[Phi];
F6 = \[Lambda]1[t]*a16 + \[Lambda]2[t]*a26 + TorquePenduloAlfa[t] -
    fPendA*\[Alpha]pend'[t];
F7 = \[Lambda]1[t]*a17 + \[Lambda]2[t]*a27 + TorquePenduloBeta[t] -
    fPendB*\[Beta]pend'[t];
F8 = \[Lambda]1[t]*a18 + \[Lambda]2[t]*a28 + TorqueVol1CMG[t];
F9 = \[Lambda]1[t]*a19 + \[Lambda]2[t]*a29 + TorqueVol2CMG[t];
F10 = \[Lambda]1[t]*a110 + \[Lambda]2[t]*a210 + TorqueVol1Fast[t];
F11 = \[Lambda]1[t]*a111 + \[Lambda]2[t]*a211 + TorqueVol2Fast[t];

(*Composicion de ecuaciones de EULA, todas estas ecuaciones son nulas,
siempre deben igualar a cero*)
eq1 = EULA1 - F1;
eq2 = EULA2 - F2;
eq3 = EULA3 - F3;
eq4 = EULA4 - F4;
eq5 = EULA5 - F5;
eq6 = EULA6 - F6;
eq7 = EULA7 - F7;
eq8 = EULA8 - F8;
```

```
eq9 = EULA9 - F9;
eq10 = EULA10 - F10;
eq11 = EULA11 - F11;

(*Condicion de no deslizamiento*)
eqR1 = Restriccion1;
eqR2 = Restriccion2;

(*Condiciones de entrada al sistema, debe haber tantas como torques de
entrada al sistema, en este caso, sustituyendo para el controlador obtenido
en retroalimentacion de estados*)
eqp1 = VoltAlfa[t];
eqp2 = VoltBeta[t] - uBeta;
eqv3 = \[CurlyPhi]1'[t];
eqv4 = \[CurlyPhi]2'[t];
eqv5 = \[Gamma]1'[t];
eqv6 = \[Gamma]2'[t];

(*Control de angulo \[Phi] por medio de retroalimentacion de estados y
reexpresado como una sola funcion, esta funcion se obtiene mas abajo, pero
se pone aqui para acelerar las simulaciones, porque el tiempo de calculo
necesario para obtener esta expresion*)
uBeta = -((0.4278062687504448 + 0.03236710202684646*Cos[\[Phi]sph[t]] -
    0.013726746118715057*Cos[2*(\[Beta]pend[t] +
    \[Phi]sph[t])])*(36*Derivative[1][\[Phi]sph][t]*(3 +
    (-0.026093055573967 - 0.13890332234250014*Sin[\[Beta]pend[t] +
    \[Phi]sph[t]]*Derivative[1][\[Beta]pend][t] +
    0.03236710202684647*Sin[\[Phi]sph[t]]*Derivative[1][\[Phi]sph][t] -
    0.13890332234250014*Sin[\[Beta]pend[t] +
    \[Phi]sph[t]]*Derivative[1][\[Phi]sph][t])/(0.4278062687504448 +
    0.03236710202684646*Cos[\[Phi]sph[t]] -
    0.013726746118715057*Cos[2*(\[Beta]pend[t] + \[Phi]sph[t])])) +
    \[Phi]sph[t]*(81 + (100*(-0.419163868 +
    0.16569095399999997*Cos[\[Beta]pend[t] +
    \[Phi]sph[t]])^2)/(0.4278062687504448 +
    0.03236710202684646*Cos[\[Phi]sph[t]] -
    0.013726746118715057*Cos[2*(\[Beta]pend[t] + \[Phi]sph[t])])^2 +
    36*((0.6363151721109496*Cos[\[Phi]sph[t]] -
    1.079433903203164*Cos[2*(\[Beta]pend[t] + \[Phi]sph[t])] -
    0.06945166117125007*Cos[\[Beta]pend[t] +
    \[Phi]sph[t]]*Derivative[1][\[Beta]pend][t]^2 +
    0.016183551013423236*Cos[\[Phi]sph[t]]*Derivative[1][\[Phi]sph][t]^2 -
    0.06945166117125007*Cos[\[Beta]pend[t] +
    \[Phi]sph[t]]*Derivative[1][\[Phi]sph][t]^2 +
    Derivative[1][\[Beta]pend][t]*(0.2197544529559321*Sin[\[Beta]pend[t] +
    \[Phi]sph[t]] - 0.13890332234250014*Cos[\[Beta]pend[t] +
    \[Phi]sph[t]]*Derivative[1][\[Phi]sph][t]))/(0.4278062687504448 +
    0.03236710202684646*Cos[\[Phi]sph[t]] -
    0.013726746118715057*Cos[2*(\[Beta]pend[t] + \[Phi]sph[t])]) -
    ((-0.032367102026846464*Sin[\[Phi]sph[t]] +
    0.027453492237430113*Sin[2*(\[Beta]pend[t] +
    \[Phi]sph[t])])*(0.6363151721109496*Sin[\[Phi]sph[t]] -
    0.5397169516015 82*Sin[2*(\[Beta]pend[t] + \[Phi]sph[t])] -
    0.06945166117125007*Sin[\[Beta]pend[t] +
    \[Phi]sph[t]]*Derivative[1][\[Beta]pend][t]^2 -
    0.026093055573967*Derivative[1][\[Phi]sph][t] +
    0.016183551013423236*Sin[\[Phi]sph[t]]*Derivative[1][\[Phi]sph][t]^2 -
    0.06945166117125007*Sin[\[Beta]pend[t] +
    \[Phi]sph[t]]*Derivative[1][\[Phi]sph][t]^2 +
    Derivative[1][\[Beta]pend][t]*(0.5559333463143228 -
    0.2197544529559321*Cos[\[Beta]pend[t] + \[Phi]sph[t]] -
    0.13890332234250014*Sin[\[Beta]pend[t] +
    \[Phi]sph[t]]*Derivative[1][\[Phi]sph][t])))/(0.4278062687504448 +
    0.03236710202684646*Cos[\[Phi]sph[t]] -
    0.013726746118715057*Cos[2*(\[Beta]pend[t] +
    \[Phi]sph[t])])^2))))/(90*(-0.419163868 +
    0.16569095399999997*Cos[\[Beta]pend[t] + \[Phi]sph[t]])) -
    \[Beta]pend[t];

(*Composicion de las ecuaciones del sistema y asignacion de condiciones
iniciales*)
eqs = {eq1 == 0, eq2 == 0, eq3 == 0, eq4 == 0, eq5 == 0, eq6 == 0,
    eq7 == 0, eq8 == 0, eq9 == 0, eq10 == 0, eq11 == 0, eqR1 == 0,
    eqR2 == 0, eqp1 == 0, eqp2 == 0, eqv3 == 0, eqv4 == 0, eqv5 == 0,
    eqv6 == 0};
eqsInit = {eqs,
    x[0] == 0, x'[0] == 0,
    y[0] == 0, y'[0] == 0,
    \[Psi]sph[0] == 0, \[Psi]sph'[0] == 0,
    \[Theta]sph[0] == 0, \[Theta]sph'[0] == 0,
    \[Phi]sph[0] == 0, \[Phi]sph'[0] == 0,
    \[Alpha]pend[0] == 0, \[Alpha]pend'[0] == 0,
```

```
    \[Beta]pend[0] == 0, \[Beta]pend'[0] == 0,
    \[CurlyPhi]1[0] == 0, \[CurlyPhi]1'[0] == 0,
    \[CurlyPhi]2[0] == 0, \[CurlyPhi]2'[0] == 0,
    \[Gamma]1[0] == 0, \[Gamma]1'[0] == 0,
    \[Gamma]2[0] == 0, \[Gamma]2'[0] == 0} // Chop;

(*Valores de parametros del robot*)
values = {
    (**)
    fEsfXAir -> 0,
    fEsfYAir -> 0,
    fEsfZ -> 0,
    fvis  -> 0,
    fPendA -> 0,
    fPendB -> 0,
    (**)
    IzzP -> 36604/1000000,
    IxxP -> 333496/1000000,
    IyyP -> 331565/1000000,
    mPend -> 5.148,
    radioPendulo -> 0.129,
    (**)
    IzzS -> 226247/1000000,
    IyyS -> 158962/1000000,
    IxxS -> 194951/1000000,
    mEsf -> 8.597,
    offsetEsf -> 0.018,
    radioEsfera -> 0.499/2,
    (**)
    IzzV -> 2*10^-3,
    IxxV -> 2*10^-5,
    IyyV -> 2*10^-5,
    mVol -> 2.4*2,
    radioVolante -> 0.10,
    offsetVol -> 0.05,
    (**)
    k1mot -> 30/12,
    k2mot -> 12 / (216*2*Pi/60),
    (**)
    g -> 9.81 };

(*Sustitucion de los valores de parametros, y solucion numerica de las
ecuaciones diferenciales generadas*)
eqsFin = eqsInit /. values;
solved = NDSolve[eqsFin, {x, y, \[Psi]sph, \[Theta]sph, \[Phi]sph,
    \[Alpha]pend, \[Beta]pend, \[CurlyPhi]1, \[CurlyPhi]2, \[Gamma]1,
    \[Gamma]2, \[Lambda]1, \[Lambda]2, VoltAlfa, VoltBeta, TorqueVol1CMG,
    TorqueVol2CMG, TorqueVol1Fast, TorqueVol2Fast},
    {t, 0, 10}, Method -> {"IndexReduction" -> True},
    Method -> {"EquationSimplification" -> "Residual"},
    AccuracyGoal -> 3, PrecisionGoal -> 3]

(*Graficas de variables de interes*)
timeEval = 10;
coors = Table[{Evaluate[{x[t]} /. solved][[1]][[1]],
    Evaluate[{y[t]} /. solved][[1]][[1]]}, {t, 0, timeEval, 0.1}];
ListPlot[coors, PlotRange -> All, ImageSize -> 400,
    PlotStyle -> {Thick, PointSize[Large]}, BaseStyle -> {20},
    AxesLabel -> {x, y}]
Plot[Evaluate[{x[t]} /. solved], {t, 0, timeEval}, PlotRange -> All,
    ImageSize -> 400, PlotStyle -> Thick, BaseStyle -> {20},
    AxesLabel -> {t, x}]
Plot[Evaluate[{y[t]} /. solved], {t, 0, timeEval}, PlotRange -> All,
    ImageSize -> 400, PlotStyle -> Thick, BaseStyle -> {20},
    AxesLabel -> {t, y}]
Plot[Evaluate[{\[Psi]sph[t]} /. solved], {t, 0, timeEval},
    PlotRange -> All, ImageSize -> 400, PlotStyle -> Thick,
    BaseStyle -> {20}, AxesLabel -> {t, \[Psi]sph}]
Plot[Evaluate[{\[Theta]sph[t]} /. solved], {t, 0, timeEval},
    PlotRange -> All, ImageSize -> 400, PlotStyle -> Thick,
    BaseStyle -> {20}, AxesLabel -> {t, \[Theta]sph}]
Plot[Evaluate[{\[Phi]sph[t]} /. solved], {t, 0, timeEval},
    PlotRange -> All, ImageSize -> 400, PlotStyle -> Thick,
    BaseStyle -> {20}, AxesLabel -> {t, \[Phi]sph}]
Plot[Evaluate[{\[Alpha]pend[t]} /. solved], {t, 0, timeEval},
    PlotRange -> All, ImageSize -> 400, PlotStyle -> Thick,
    BaseStyle -> {20}, AxesLabel -> {t, \[Alpha]pend}]
Plot[Evaluate[{\[Beta]pend[t]} /. solved], {t, 0, timeEval},
    PlotRange -> All, ImageSize -> 400, PlotStyle -> Thick,
    BaseStyle -> {20}, AxesLabel -> {t, \[Beta]pend}]
Plot[Evaluate[{\[Theta]sph[t] + \[Alpha]pend[t]} /. solved],
    {t, 0, timeEval}, PlotRange -> All, ImageSize -> 400,
```

```
    PlotStyle -> Thick, BaseStyle -> {20},
    AxesLabel -> {t, sum\[Theta]\[Alpha]}]
Plot[Evaluate[{\[Phi]sph[t] + \[Beta]pend[t]} /. solved],
    {t, 0, timeEval}, PlotRange -> All, ImageSize -> 400,
    PlotStyle -> Thick, BaseStyle -> {20},
    AxesLabel -> {t, sum\[Phi]\[Beta]}]

(*Animacion 3D de la esfera, la animacion inicia detenida, en Linux, hay
que modificar el parametro AnimationRunning a True para que pueda verse en
movimiento*)
Animate[params = values;

(*Primer Volante de inercia*)
centroVolante1 = Transpose[(VecVol1 /. solved[[1]][[1]]
    /. {t -> varIter} /. solved)[[1]]][[1]] /. params;
directorVolante1 = Transpose[(EsfPrimRot.EsfSecRot.EsfTerRot.PendPrimRot.
    PendSecRot.\Vol1Rot1.{{0}, {0}, {1}} /. solved[[1]][[1]]
    /. {t -> varIter} /. solved)[[1]]][[1]] /. params;
inicioVolante1 = (-0.025 * 0.5 * directorVolante1 + centroVolante1)
    /. params;
finVolante1 = (0.025 * 0.5 * directorVolante1 + centroVolante1) /. params;
Volante1 = Graphics3D[{(*Green,Thick,Arrow[{centroVolante1,
    centroVolante1 + 10*(finVolante1- centroVolante1)}],*)FaceForm[Red],
    EdgeForm[Directive[Dashing[Large], Thick, Blue]],
    Cylinder[{inicioVolante1, finVolante1}, 0.5*radioVolante /. params],
    Axes -> True}];

(*Segundo Volante de inercia*)
centroVolante2 =Transpose[(VecVol2 /. solved[[1]][[1]] /. {t -> varIter}
    /. solved)[[1]]][[1]] /. params;
directorVolante2 = Transpose[(EsfPrimRot.EsfSecRot.EsfTerRot.PendPrimRot.
    PendSecRot.\ Vol2Rot1.{{0}, {0}, {-1}} /. solved[[1]][[1]]
    /. {t -> varIter} /. solved)[[1]]][[1]] /. params;
inicioVolante2 = (-0.025 * 0.5 * directorVolante2 + centroVolante2)
    /. params;
finVolante2 = (0.025 * 0.5 * directorVolante2 + centroVolante2) /. params;
Volante2 = Graphics3D[{(*Green,Thick,Arrow[{centroVolante2,
    centroVolante2 + 10*(finVolante2- centroVolante2)}],*)FaceForm[Red],
    EdgeForm[Directive[Dashing[Large], Thick, Blue]],
    Cylinder[{inicioVolante2, finVolante2}, 0.5*radioVolante /. params],
    Axes -> True}];

(*Pendulo de contrapeso*)
centroPendulo = Transpose[(VecPend /. solved[[1]][[1]] /. {t -> varIter}
    /. solved)[[1]]][[1]] /. params;
Pendulo = Graphics3D[Sphere[centroPendulo, 0.5*radioVolante]] /. params;

 (*Esfera de contenimiento*)
VecCentro = Transpose[(VecEsfera /. solved[[1]][[1]] /. {t -> varIter}
    /. solved)[[1]]][[1]] /. params;
directorEsfera = Transpose[(EsfTerRot.EsfSecRot.EsfPrimRot.{{0}, {0}, {.2}}
    /. solved[[1]][[1]] /. {t -> varIter} /. solved)[[1]]][[1]] /. params;
LinEsfera = Graphics3D[Arrow[{VecCentro, VecCentro + directorEsfera} ]];
Esfera = Graphics3D[{Opacity[.1], Sphere[VecCentro, radioEsfera
    /. params]}];
(**)
VecContacto = Transpose[(({{x[t]}, {y[t]}, {0}}) /. solved[[1]][[1]]
    /. {t -> varIter} /. solved)[[1]]][[1]] /. params;
Linea = Graphics3D[Line[{{0, 0, 0}, VecContacto}]];

(*Cilindro del eje de la esfera*)
otroDirectorEsfera = Transpose[(EsfTerRot.EsfSecRot.EsfPrimRot.
    {{.2}, {0}, {0}} /. solved[[1]][[1]] /. {t -> varIter}
    /. solved)[[1]]][[1]] /. params;
LinOtro = Graphics3D[Cylinder[{VecCentro - otroDirectorEsfera,
    VecCentro + otroDirectorEsfera}, 0.01 ]];
LinPendulo = Graphics3D[Cylinder[{VecCentro, centroPendulo}, 0.005 ]];
LinVolantes = Graphics3D[Cylinder[{centroVolante1, centroVolante2},
    0.005 ]];
Hub = Graphics3D[Sphere[VecCentro, 0.02]] /. params;
Arrow1 = Graphics3D[{Red, Arrow[{{0, 0, 0}, {0, .21, 0}}]}];
Arrow2 = Graphics3D[{Red, Arrow[{{0, 0, 0}, {0, -.08, 0}}]}];

Show[{Arrow1, Arrow2, Hub, LinVolantes, Volante1, Volante2, Pendulo,
    Esfera, LinOtro(*,Linea,LinEsfera,LinOtro*), LinPendulo},
    PlotRange -> {{-.5, .5}, {-.5, .5}, {0, .5}}, ImageSize -> 900,
    Axes -> {True, True, True}, AxesLabel -> {x, y, z}, Boxed -> False],
    {varIter, 0, 10}, AnimationRate -> 1, AnimationRunning -> False]
```

C.1.2 Linealización y control

Como continuación al código de modelado, también para Mathematica 9 o 10, en otro archivo. Se toma un modelo reducido del sistema en grados de libertad, para derivar un controlador del ángulo de "viraje" de la esfera. Como limitación importante, la simulación del controlador requiere que éste se represente como una función escalar para cada entrada al sistema. Además, como la asignación de polos en cualquier linealización de orden 5 o superior no puede hacerse de forma exacta, y los casos de orden 3 o 4 son relativamente complicados, se hace una linealización continua solamente de orden 2. El código genera a su vez el código del controlador en el lenguaje de programación Python, que se usa para implementar el sistema de control en el robot.

```
(*Es posible usar las ecuaciones obtenidas simbolicamente, para obtener una
linearizacion del sistema en todo punto de operacion, esto NO es lo mismo
que una linearizacion exacta del sistema, porque no hay ninguna garantia de
que las trayectorias en el sistema linealizado sean iguales o
transformables a las trayectorias del sistema verdadero, en el mejor de los
casos, se asume que se proporciona una mejor aproximacion para proposito s
de control unicamente, para la prueba de control, se uso el sistema de la
esfera con los volantes removidos, asumiendo que todas las fricciones son
nulas, salvo una friccion de la esfera con el aire por la velocidad en x e
y, que se asume razonable como de un Newton de oposicion para una velocidad
de 1 metro por segundo, este valor es relativamente irrelevante, pero su
existencia permite garantizar que a una entrada nula el sistema
eventualmente alcanzara el reposo luego de una cantidad, quiza
considerable, de tiempo*)

(*El sistema linealizado para propositos de control hara uso solo de un
subconjunto de las ecuaciones originales, considerando que solo existe
movimiento en y, \[Phi]Sph y \[Beta]pend, como en este caso, hay una
dependencia siempre lineal entre y e \[Phi]Sph, y no nos interesa el
multiplicador de lagrange, debemos eliminar y e sus derivadas, y el
multiplicador, haciendo las sustituciones y simplificaciones pertienentes*)
valuesLin = {
    fEsfXAir -> 0,
    fEsfYAir -> 0,
    fEsfZ -> 0,
    fvis  -> 0,
    fPendA -> 0,
    fPendB -> 0,
    (**)
    IzzP -> 36604/1000000,
    IxxP -> 333496/1000000,
    IyyP -> 331565/1000000,
    mPend -> 5.148,
    radioPendulo -> 0.129,
    (**)
    IzzS -> 226247/1000000,
    IyyS -> 158962/1000000,
    IxxS -> 194951/1000000,
    mEsf -> 8.597,
    offsetEsf -> 0.018,
    radioEsfera -> 0.499/2,
    (**)
    IzzV -> 0,
    IxxV -> 0,
    IyyV -> 0,
    mVol -> 0,
    radioVolante -> 0,
    offsetVol -> 0,
    (**)
    k1mot -> 30/12,
    k2mot -> 12 / (216*2*Pi/60),
    (**)
    g -> 9.81,
    (**)
    x[t] -> 0, x'[t] -> 0,
```

```
    \[Psi]sph[t] -> 0, \[Psi]sph'[t] -> 0,
    \[Theta]sph[t] -> 0, \[Theta]sph'[t] -> 0,
    \[Alpha]pend[t] -> 0, \[Alpha]pend'[t] -> 0,
    \[Lambda]1[t] -> 0, \[Lambda]1'[t] -> 0,
    VoltAlfa[t] -> 0, VoltAlfa'[t] -> 0 };
valuesLinReduce = {
    (**)
    IzzV -> 0,
    IxxV -> 0,
    IyyV -> 0,
    mVol -> 0,
    radioVolante -> 0,
    (**)
    x[t] -> 0, x'[t] -> 0,
    \[Psi]sph[t] -> 0, \[Psi]sph'[t] -> 0,
    \[Theta]sph[t] -> 0, \[Theta]sph'[t] -> 0,
    \[Alpha]pend[t] -> 0, \[Alpha]pend'[t] -> 0,
    \[Lambda]1[t] -> 0, \[Lambda]1'[t] -> 0,
    VoltAlfa[t] -> 0, VoltAlfa'[t] -> 0 };
eqsLin = {eq2 == 0 && eq5 == 0 && eq7 == 0 && eqR2 == 0 && D[eqR2, t] == 0}
    /. valuesLinReduce // Chop;
solveEqsLin = Eliminate[eqsLin, {y'[t], y''[t], \[Lambda]2[t]}];

(*Y para hallar el sistema linealizado, debemos obtener las segundas
derivadas de las variables de interes*)
solveSph = Solve[solveEqsLin, {\[Phi]sph''[t], \[Beta]pend''[t]}]
    // Simplify // Chop;
\[Phi]pp = \[Phi]sph''[t] /. solveSph[[1]][[1]];
\[Beta]pp = \[Beta]pend''[t] /. solveSph[[1]][[2]];

(*Obteniendo las derivadas parciales de las variables de interes*)
\[Phi]pp\[Phi]p = D[\[Phi]pp, \[Phi]sph'[t]] /. {VoltBeta[t] -> 0} // Chop;
\[Phi]pp\[Phi]  = D[\[Phi]pp, \[Phi]sph[t]] /. {VoltBeta[t] -> 0} // Chop;
\[Phi]pp\[Beta]p = D[\[Phi]pp, \[Beta]pend'[t]] /. {VoltBeta[t] -> 0}
    // Chop;
\[Phi]pp\[Beta]  = D[\[Phi]pp, \[Beta]pend[t]] /. {VoltBeta[t] -> 0}
    // Chop;
\[Beta]pp\[Phi]p = D[\[Beta]pp, \[Phi]sph'[t]] /. {VoltBeta[t] -> 0}
    // Chop;
\[Beta]pp\[Phi]  = D[\[Beta]pp, \[Phi]sph[t]] /. {VoltBeta[t] -> 0}
    // Chop;
\[Beta]pp\[Beta]p = D[\[Beta]pp, \[Beta]pend'[t]] /. {VoltBeta[t] -> 0}
    // Chop;
\[Beta]pp\[Beta]  = D[\[Beta]pp, \[Beta]pend[t]] /. {VoltBeta[t] -> 0}
    // Chop;
\[Phi]ppV  = D[\[Phi]pp, VoltBeta[t]] // Chop;
\[Beta]ppV  = D[\[Beta]pp, VoltBeta[t]] // Chop;

(*Tratamos de controlar un sistema de segundo orden, por las limitaciones
de Mathematica y del codigo de Python, es poco conveniente (espcialmente en
el caso de Mathematica) tener como entrada cualquier otra cosa que no sea
una funcion suave, continua y al menos doblemente derivable en la region de
interes, lo cual nos restringe a decir que la funcion de entrada debe ser
una composicion racional de funciones seno, coseno, potencia, derivacion,
etcetera.

Como vector de estado, escogemos para las variables de interes
\[Phi]p y \[Phi] el orden
x = {{\[Phi]p},{\[Phi]}}

De ello, la matriz debe tener la forma *)
A = ( {
    {a, b},
    {1, 0} } );

(*Y la matriz de entrada debe tener la forma*)
B = {{c}, {0}};

(*Y la matriz K*)
K = {{k1, k2}};

(*con una A en lazo cerrado de*)
Ap = A - B.K;

(*con ecuacion caracteristica*)
eqc = Det[( {
      {s, 0},
      {0, s} } ) - Ap];

(*Y eigenvalores*)
eg1 = (Solve[eqc == 0, s] // Flatten)[[1]][[2]];
eg2 = (Solve[eqc == 0, s] // Flatten)[[2]][[2]];
```

```
(*Los eigenvalores osn las raices del sistema, y describen, para el sistema
de segundo orden, su comportamiento, para que el sistema sea estale y no
subamortiguado, podemos hacer que ambos eigenvalores sean iguales entre si,
a cierta cantidad real, que especifique un tiempo de asentamiento, digamos,
usando la regla de que 3 veces la constante de tiempo es suficiente para
lograr estabilidad*)
tdeseado = 2;
ct = 1/tdeseado;
ksolved = Solve[{eg1 == -3 ct, eg2 == -3 ct}, {k1, k2}] // Flatten;

(*Matriz de preescalamiento*)
k0 = {{0, 1}}.Inverse[(-A + B.K)].B /. ksolved;

(*Entonces, podemos obtener el codigo de Python para la entrada como*)
u = k0.{{-\[Phi]sph[t]}} - K.{{\[Phi]sph'[t]}, {\[Phi]sph[t]}} /. ksolved
    /. {add -> \[Phi]pp} /. valuesLin;
u1 = u[[1]][[1]] // FullSimplify;
u2 = u1 /. {a -> \[Phi]pp\[Phi]p, b -> \[Phi]pp\[Phi], c -> \[Phi]ppV}
    /. valuesLin /. {\[Beta]pend[t] -> beta, \[Beta]pend'[t] -> betap,
    \[Phi]sph[t] -> fi, \[Phi]sph'[t] -> fip, Cos -> cos, Sin -> sin};
u3 = u2 // FullSimplify;
u3text = ExportString[u3, "Text"];
StringReplace[u3text, {"[" -> "(", "]" -> ")", "^" -> "**",
    "fi" -> "self.fi", "beta" -> "self.beta"}]

(*Y necesitamos expresar u para el sistema modelado en Mathematica y ser
capaces de simular*)
(u1 /. {a -> \[Phi]pp\[Phi]p, b -> \[Phi]pp\[Phi], c -> \[Phi]ppV}
    /. valuesLin) // InputForm
```

C.1.3 Complejidad generada

Para dar una idea de la complejidad de las ecuaciones generadas con esta forma de modelado, se muestra una fracción de la expresión generada para la primera de las ecuaciones de Euler-Lagrange (sin sustituir parámetros por sus valores numéricos) sobre la coordenada generalizada x, comparativamente, una de las que tiene el menor numero de términos.

```
-\[Lambda]1[t] + fEsfXAir Derivative[1][x][t] +
    1. mEsf ((x^\[Prime]\[Prime])[t] +
    offsetEsf (-Cos[\[Phi]sph[t]] Cos[\[Psi]sph[t]] Sin[\[Theta]sph[t]]
    Derivative[1][\[Theta]sph][t]^2 -
    2 Cos[\[Theta]sph[t]] Cos[\[Psi]sph[t]] Sin[\[Phi]sph[t]]
    Derivative[1][\[Theta]sph][t] Derivative[1][\[Phi]sph][t] -
    Cos[\[Phi]sph[t]] Cos[\[Psi]sph[t]] Sin[\[Theta]sph[t]]
    Derivative[1][\[Phi]sph][t]^2 -
    Sin[\[Phi]sph[t]] Sin[\[Psi]sph[t]] Derivative[1][\[Phi]sph][t]^2 -
    2 Cos[\[Theta]sph[t]] Cos[\[Phi]sph[t]] Sin[\[Psi]sph[t]]
    Derivative[1][\[Theta]sph][t] Derivative[1][\[Psi]sph][t] +
    2 Cos[\[Phi]sph[t]] Cos[\[Psi]sph[t]] Derivative[1][\[Phi]sph][t]
    Derivative[1][\[Psi]sph][t] +
    2 Sin[\[Theta]sph[t]] Sin[\[Phi]sph[t]] Sin[\[Psi]sph[t]]
    Derivative[1][\[Phi]sph][t] Derivative[1][\[Psi]sph][t] -
    Cos[\[Phi]sph[t]] Cos[\[Psi]sph[t]] Sin[\[Theta]sph[t]]
    Derivative[1][\[Psi]sph][t]^2 -
    Sin[\[Phi]sph[t]] Sin[\[Psi]sph[t]] Derivative[1][\[Psi]sph][t]^2 +
    Cos[\[Theta]sph[t]] Cos[\[Phi]sph[t]] Cos[\[Psi]sph[t]]
    (\[Theta]sph^\[Prime]\[Prime])[t] -
    Cos[\[Psi]sph[t]] Sin[\[Theta]sph[t]] Sin[\[Phi]sph[t]]
    (\[Phi]sph^\[Prime]\[Prime])[t] +
    Cos[\[Phi]sph[t]] Sin[\[Psi]sph[t]] (\[Phi]sph^\[Prime]\[Prime])[t] +
    Cos[\[Psi]sph[t]] Sin[\[Phi]sph[t]] (\[Psi]sph^\[Prime]\[Prime])[t] -
    Cos[\[Phi]sph[t]] Sin[\[Theta]sph[t]] Sin[\[Psi]sph[t]]
    (\[Psi]sph^\[Prime]\[Prime])[t])) + 1. mVol ((x^\[Prime]\[Prime])[t] -
    radioVolante (-Cos[\[Beta]pend[t]] (Cos[\[Psi]sph[t]]
    Sin[\[Theta]sph[t]] Sin[\[Phi]sph[t]] -
    Cos[\[Phi]sph[t]] Sin[\[Psi]sph[t]]) Derivative[1][\[Beta]pend][t]^2 -
    Sin[\[Beta]pend[t]] (Cos[\[Phi]sph[t]] Cos[\[Psi]sph[t]]
    Sin[\[Theta]sph[t]] +
    Sin[\[Phi]sph[t]] Sin[\[Psi]sph[t]]) Derivative[1][\[Beta]pend][t]^2 +
```

```
2 Cos[\[Beta]pend[t]] Derivative[1][\[Beta]pend][t]
(Cos[\[Theta]sph[t]] Cos[\[Phi]sph[t]] Cos[\[Psi]sph[t]]
Derivative[1][\[Theta]sph][t] -
Cos[\[Psi]sph[t]] Sin[\[Theta]sph[t]] Sin[\[Phi]sph[t]]
Derivative[1][\[Phi]sph][t] +
Cos[\[Phi]sph[t]] Sin[\[Psi]sph[t]] Derivative[1][\[Phi]sph][t] +
Cos[\[Psi]sph[t]] Sin[\[Phi]sph[t]] Derivative[1][\[Psi]sph][t] -
Cos[\[Phi]sph[t]] Sin[\[Theta]sph[t]] Sin[\[Psi]sph[t]]
Derivative[1][\[Psi]sph][t]) -
2 Sin[\[Beta]pend[t]] Derivative[1][\[Beta]pend][t]
(Cos[\[Theta]sph[t]] Cos[\[Psi]sph[t]] Sin[\[Phi]sph[t]]
Derivative[1][\[Theta]sph][t] +
Cos[\[Phi]sph[t]] Cos[\[Psi]sph[t]] Sin[\[Theta]sph[t]]
Derivative[1][\[Phi]sph][t] +
Sin[\[Phi]sph[t]] Sin[\[Psi]sph[t]] Derivative[1][\[Phi]sph][t] -
Cos[\[Phi]sph[t]] Cos[\[Psi]sph[t]] Derivative[1][\[Psi]sph][t] -
Sin[\[Theta]sph[t]] Sin[\[Phi]sph[t]] Sin[\[Psi]sph[t]]
Derivative[1][\[Psi]sph][t]) - Sin[\[Beta]pend[t]]
(Cos[\[Psi]sph[t]] Sin[\[Theta]sph[t]] Sin[\[Phi]sph[t]] -
Cos[\[Phi]sph[t]] Sin[\[Psi]sph[t]])(\[Beta]pend^\[Prime]\[Prime])[t] +
Cos[\[Beta]pend[t]] (Cos[\[Phi]sph[t]] Cos[\[Psi]sph[t]]
Sin[\[Theta]sph[t]] +
Sin[\[Phi]sph[t]] Sin[\[Psi]sph[t]])(\[Beta]pend^\[Prime]\[Prime])[t] +
Sin[\[Beta]pend[t]] (-Cos[\[Phi]sph[t]] Cos[\[Psi]sph[t]]
Sin[\[Theta]sph[t]] Derivative[1][\[Theta]sph][t]^2 -
2 Cos[\[Theta]sph[t]] Cos[\[Psi]sph[t]] Sin[\[Phi]sph[t]]
Derivative[1][\[Theta]sph][t] Derivative[1][\[Phi]sph][t] -
Cos[\[Phi]sph[t]] Cos[\[Psi]sph[t]] Sin[\[Theta]sph[t]]
Derivative[1][\[Phi]sph][t]^2 - Sin[\[Phi]sph[t]]
Sin[\[Psi]sph[t]] Derivative[1][\[Phi]sph][t]^2 -
2 Cos[\[Theta]sph[t]] Cos[\[Phi]sph[t]] Sin[\[Psi]sph[t]]
Derivative[1][\[Theta]sph][t] Derivative[1][\[Psi]sph][t] +
2 Cos[\[Phi]sph[t]] Cos[\[Psi]sph[t]] Derivative[1][\[Phi]sph][t]
Derivative[1][\[Psi]sph][t] + 2 Sin[\[Theta]sph[t]] Sin[\[Phi]sph[t]]
Sin[\[Psi]sph[t]] Derivative[1][\[Phi]sph][t]
Derivative[1][\[Psi]sph][t] -
Cos[\[Phi]sph[t]] Cos[\[Psi]sph[t]] Sin[\[Theta]sph[t]]
Derivative[1][\[Psi]sph][t]^2 -
Sin[\[Phi]sph[t]] Sin[\[Psi]sph[t]] Derivative[1][\[Psi]sph][t]^2 +
Cos[\[Theta]sph[t]] Cos[\[Phi]sph[t]] Cos[\[Psi]sph[t]]
(\[Theta]sph^\[Prime]\[Prime])[t] -
Cos[\[Psi]sph[t]] Sin[\[Theta]sph[t]] Sin[\[Phi]sph[t]]
(\[Phi]sph^\[Prime]\[Prime])[t] +
Cos[\[Phi]sph[t]] Sin[\[Psi]sph[t]] (\[Phi]sph^\[Prime]\[Prime])[t] +
Cos[\[Psi]sph[t]] Sin[\[Phi]sph[t]] (\[Psi]sph^\[Prime]\[Prime])[t] -
Cos[\[Phi]sph[t]] Sin[\[Theta]sph[t]] Sin[\[Psi]sph[t]]
(\[Psi]sph^\[Prime]\[Prime])[t]) +
Cos[\[Beta]pend[t]] (-Cos[\[Psi]sph[t]] Sin[\[Theta]sph[t]]
Sin[\[Phi]sph[t]] Derivative[1][\[Theta]sph][t]^2 +
2 Cos[\[Theta]sph[t]] Cos[\[Phi]sph[t]] Cos[\[Psi]sph[t]]
Derivative[1][\[Theta]sph][t] Derivative[1][\[Phi]sph][t] -
Cos[\[Psi]sph[t]] Sin[\[Theta]sph[t]] Sin[\[Phi]sph[t]]
Derivative[1][\[Phi]sph][t]^2 +
Cos[\[Phi]sph[t]] Sin[\[Psi]sph[t]] Derivative[1][\[Phi]sph][t]^2 -
2 Cos[\[Theta]sph[t]] Sin[\[Phi]sph[t]] Sin[\[Psi]sph[t]]
Derivative[1][\[Theta]sph][t] Derivative[1][\[Psi]sph][t] +
2 Cos[\[Psi]sph[t]] Sin[\[Phi]sph[t]] Derivative[1][\[Phi]sph][t]
Derivative[1][\[Psi]sph][t] - 2 Cos[\[Phi]sph[t]] Sin[\[Theta]sph[t]]
Sin[\[Psi]sph[t]] Derivative[1][\[Phi]sph][t]
Derivative[1][\[Psi]sph][t] - Cos[\[Psi]sph[t]] Sin[\[Theta]sph[t]]
Sin[\[Phi]sph[t]] Derivative[1][\[Psi]sph][t]^2 +
Cos[\[Phi]sph[t]] Sin[\[Psi]sph[t]] Derivative[1][\[Psi]sph][t]^2 +
Cos[\[Theta]sph[t]] Cos[\[Psi]sph[t]] Sin[\[Phi]sph[t]]
(\[Theta]sph^\[Prime]\[Prime])[t] +
Cos[\[Phi]sph[t]] Cos[\[Psi]sph[t]] Sin[\[Theta]sph[t]]
(\[Phi]sph^\[Prime]\[Prime])[t] +
Sin[\[Phi]sph[t]] Sin[\[Psi]sph[t]] (\[Phi]sph^\[Prime]\[Prime])[t] -
Cos[\[Phi]sph[t]] Cos[\[Psi]sph[t]] (\[Psi]sph^\[Prime]\[Prime])[t] -
Sin[\[Theta]sph[t]] Sin[\[Phi]sph[t]] Sin[\[Psi]sph[t]]
(\[Psi]sph^\[Prime]\[Prime])[t])) -
offsetVol (-Cos[\[Theta]sph[t]] Cos[\[Psi]sph[t]] Sin[\[Alpha]pend[t]]
Derivative[1][\[Alpha]pend][t]^2 -
Cos[\[Alpha]pend[t]] (-Sin[\[Beta]pend[t]] (Cos[\[Psi]sph[t]]
Sin[\[Theta]sph[t]] Sin[\[Phi]sph[t]] -
Cos[\[Phi]sph[t]] Sin[\[Psi]sph[t]]) +
Cos[\[Beta]pend[t]] (Cos[\[Phi]sph[t]] Cos[\[Psi]sph[t]]
Sin[\[Theta]sph[t]] + Sin[\[Phi]sph[t]] Sin[\[Psi]sph[t]]))
Derivative[1][\[Alpha]pend][t]^2 -
2 Cos[\[Alpha]pend[t]] Cos[\[Psi]sph[t]] Sin[\[Theta]sph[t]]
Derivative[1][\[Alpha]pend][t] Derivative[1][\[Theta]sph][t] -
Cos[\[Theta]sph[t]] Cos[\[Psi]sph[t]] Sin[\[Alpha]pend[t]]
Derivative[1][\[Theta]sph][t]^2 -
```

```
2 Cos[\[Alpha]pend[t]] Cos[\[Theta]sph[t]] Sin[\[Psi]sph[t]]
Derivative[1][\[Alpha]pend][t] Derivative[1][\[Psi]sph][t] +
2 Sin[\[Alpha]pend[t]] Sin[\[Theta]sph[t]] Sin[\[Psi]sph[t]]
Derivative[1][\[Theta]sph][t] Derivative[1][\[Psi]sph][t] -
Cos[\[Theta]sph[t]] Cos[\[Psi]sph[t]] Sin[\[Alpha]pend[t]]
Derivative[1][\[Psi]sph][t]^2 -
2 Sin[\[Alpha]pend[t]] Derivative[1][\[Alpha]pend][t]
(-Cos[\[Beta]pend[t]] (Cos[\[Psi]sph[t]] Sin[\[Theta]sph[t]]
Sin[\[Phi]sph[t]] -
Cos[\[Phi]sph[t]] Sin[\[Psi]sph[t]]) Derivative[1][\[Beta]pend][t] -
Sin[\[Beta]pend[t]] (Cos[\[Phi]sph[t]] Cos[\[Psi]sph[t]]
Sin[\[Theta]sph[t]] + Sin[\[Phi]sph[t]] Sin[\[Psi]sph[t]])
Derivative[1][\[Beta]pend][t] + ...
```

C.2 Código en el robot

Se listan los nodos principales usados en el control y manejo del robot, exceptuando algunas bibliotecas generadas por otros autores. Como se usa la plataforma ROS para el desarrollo de los algoritmos y programas, los programas se limitan a los lenguajes de programación C++ y Python. Los programas escritos en Python pueden ser optimizados en versiones posteriores reescribiéndolos en C++. Un proyecto de ROS necesita de una modesta cantidad de infraestructura de código propio, y de la presencia de ROS instalado en un sistema operativo basado en Linux. El proyecto completo y sus dependencias externas pueden encontrarse en el repositorio (a abril del 2015): `https://github.com/eigendreams/esferico`.

C.2.1 Interfaz con el *joystick*

Este programa debe residir en la computadora de tierra, con ROS, o alguna implementación equivalente. El nodo se encarga de procesar la información de un *joystick* (en particular, el controlador de Xbox 360), y mandar los mensajes de valores de ángulo y velocidad deseada al robot, o bien, los valores de PWM de salida a los motores, en valores de -10000 a 10000, según la dirección.

```
#!/usr/bin/env python
# -*- coding: utf8 -*-

import rospy

from std_msgs.msg import Bool
from std_msgs.msg import Int16
from std_msgs.msg import Float32
from sensor_msgs.msg import Joy

from ino_mod import *

class Control_interface:
    #
    def __init__(self, node_name_override = 'control_interface'):
        #
        rospy.init_node(node_name_override)
        self.nodename = rospy.get_name()
        rospy.loginfo("Node starting with name %s", self.nodename)
```

```
        self.rate = float(rospy.get_param("rate", 5))
        # Nombres de los botones de Joy
        self.axes_names     = {'left_stick_hor':0, 'left_stick_ver':1, 'LT':2, '
            right_stick_hor':3, 'right_stick_ver':4, 'RT':5, 'cross_hor':6, 'cross_ver
            ':7}
        self.buttons_names  = {'A':0, 'B':1, 'X':2, 'Y':3, 'LB':4, 'RB':5, 'back':6, '
            start':7, 'power':8, 'btn_stick_left':9, 'btn_stick_right':10}
        # Publicadores a los motores, SOLO para el modo manual
        self.m1     = rospy.Publisher("m1", Int16)                # salida al motor 1
        self.m2     = rospy.Publisher("m2", Int16)                # salida al motor 1
        # Publicadores para valores controlados, bandera de comunicacion, y modo de
            control (0 automatico, 1 manual)
        self.veldespub  = rospy.Publisher("vel_delante_des", Float32)
        self.angdespub  = rospy.Publisher("ang_lateral_des", Float32)
        self.alpub      = rospy.Publisher("al", Int16)
        self.conmode    = rospy.Publisher("con_mode", Int16)
        #
        self.btog = 0
        self.lastb = 0
        self.lastbtog = 0
        self.btogchanged = 0
        #
        self.powtog = 0
        self.lastpow = 0
        self.lastpowtog = 0
        self.powtogchanged = 0
        #
        self.lastrb = 0
        self.lastlb = 0
        self.rtval = 0
        self.ltval = 0
        #
        self.angle_des = 0
        self.vel_des    = 0
        # Proteccion por timeout de Joy
        self.inittime = rospy.get_time()
        self.timelastjoy = -1000
        self.timed_out = True
        #
        self.joySub = rospy.Subscriber("joy", Joy, self.joyCb)
        #
    def joyCb(self, data):
        #
        self.timelastjoy = millis(self.inittime)
        self.timed_out   = False
        # Quiero detectar cambion del boton de power
        self.lastpowtog = self.powtog
        if (data.buttons[self.buttons_names['power']] is 1 and self.lastpow is 0):
            self.powtog = int(not self.powtog)
        self.powtogchanged = self.lastpowtog or not (self.powtog == self.lastpowtog)
        self.lastpow = data.buttons[self.buttons_names['power']]
        # y del boton b
        self.lastbtog = self.btog
        if (data.buttons[self.buttons_names['B']] is 1 and self.lastb is 0):
            self.btog = int(not self.btog)
        self.btogchanged = 0 or not (self.btog == self.lastbtog)
        self.lastb = data.buttons[self.buttons_names['B']]
        # Corrigiendo escala y signo de los gatillos
        self.rtval = (data.axes[self.axes_names['RT']] - 1) / -2.
        self.ltval = (data.axes[self.axes_names['LT']] - 1) / -2.
        #
        self.lastlb = data.buttons[self.buttons_names['LB']]
        self.lastrb = data.buttons[self.buttons_names['RB']]
        #
        self.angle_des   = -data.axes[self.axes_names['left_stick_hor']]
        self.vel_des     = data.axes[self.axes_names['right_stick_ver']]
        #
    def update(self):
        # Proteccion por timeout
        if ((millis(self.inittime) - self.timelastjoy) > 1000):
            if not self.timed_out :
                self.angle_des = 0
                self.vel_des = 0
                self.powtog = 0
                self.btog = 0
            self.timed_out = True
        #
        self.conmode.publish(self.btog)
        self.alpub.publish(self.powtog)
        # Obteniendo valores de escalamiento para el modo manual
        self.angmultval = constrain( (1 + self.ltval) * (1 + self.lastlb), 1, 4)
        self.velmultval = constrain( (1 + self.rtval) * (1 + self.lastrb), 1, 4)
```

```
        #
        if (self.btog is 1):
            # Modo manual
            self.m1.publish(constrain(self.vel_des * 7 * self.velmultval * 100 - self.
                angle_des * 7 * self.angmultval * 100,-3000, 3000))
            self.m2.publish(constrain(self.vel_des * 7 * self.velmultval * 100 + self.
                angle_des * 7 * self.angmultval * 100,-3000, 3000))
        else:
            # Modo controlado
            self.angdespub.publish(self.angle_des * 1.0)
            self.veldespub.publish(self.vel_des   * 1.0)
            #
    def spin(self):
                #
        r = rospy.Rate(self.rate)
        #
        while not rospy.is_shutdown():
            self.update()
            r.sleep()
        #
if __name__ == '__main__':
    """ main """
    control_interface = Control_interface()
    control_interface.spin()
```

C.2.2 Lectura de los codificadores

Este nodo hace interfaz con algunos de los pines de un BeagleBone Black, para leer codificadores absolutos de efecto Hall AS5043, usando el protocolo SPI emulado por software. Sólo se leen los 10 bits de información de posición angular. Por limitaciones del sistema, debe ser ejecutado como administrador.

```
#!/usr/bin/env python
# -*- coding: utf8 -*-

import rospy
import time
usleep = lambda x: time.sleep(x/1000000.0)
from std_msgs.msg import Int16
import Adafruit_BBIO.GPIO as GPIO

class Read_encoders:
    #
    def __init__(self, node_name_override = 'read_encoders'):
                #
        rospy.init_node(node_name_override)
        self.nodename = rospy.get_name()
        rospy.loginfo("Node starting with name %s", self.nodename)
        self.rate = float(rospy.get_param("rate", 10))
        # Pines de los encoders
        self.cs1 = "P9_31"
        self.cs2 = "P9_29"
        self.do  = "P9_25"
        self.clk = "P9_23"
        #
        GPIO.cleanup()
        GPIO.setup(self.cs1, GPIO.OUT)
        GPIO.setup(self.cs2, GPIO.OUT)
        GPIO.setup(self.do,  GPIO.IN)
        GPIO.setup(self.clk, GPIO.OUT)
        #
        self.closeComm(self.cs1)
        self.closeComm(self.cs2)
        #
        self.e1val = 0
        self.e2val = 0
        #
        self.e1Pub = rospy.Publisher("e1", Int16)
        self.e2Pub = rospy.Publisher("e2", Int16)
        #
    def readSingle(self, pincsn):
```

```
        # Lectura de la trama de datos de los encoders, se toman los primeros 10 bits (
            posicion absoluta)
        self.chainData = 0
        GPIO.output(pincsn,   GPIO.LOW)
        GPIO.output(self.clk, GPIO.LOW)
        for k in range(16):
            GPIO.output(self.clk, GPIO.HIGH)
            self.chainData = (self.chainData << 1) | GPIO.input(self.do)
            GPIO.output(self.clk, GPIO.LOW)
        self.closeComm(pincsn)
        return (self.chainData >> 6)
        #
    def closeComm(self, pincsn):
        #
        GPIO.output(pincsn,   GPIO.HIGH)
        GPIO.output(self.clk, GPIO.HIGH)
        #
    def update(self):
                #
        self.e1val = self.readSingle(self.cs1)
        self.e2val = self.readSingle(self.cs2)
        self.e1Pub.publish(self.e1val)
        self.e2Pub.publish(self.e2val)
        #
    def spin(self):
                #
        r = rospy.Rate(self.rate)
        while not rospy.is_shutdown():
            self.update()
            r.sleep()
        #
if __name__ == '__main__':
    """ main """
    readencs = Read_encoders()
    readencs.spin()
```

C.2.3 Biblioteca de funciones

Algunas funciones útiles que replican la sintaxis de las bibliotecas usadas por Arduino.

```
#!/usr/bin/env python
# -*- coding: utf8 -*-
##############################################################################
from math import *
##############################################################################
import rospy
##############################################################################
#
def map(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min + 0.) / (in_max - in_min + 0.) + out_min
    #
def millis(init_time):
    return 1000. * (rospy.get_time() - init_time)
    #
def constrain(x, min, max):
    if (x > max):
        return max
    if (x < min):
        return min
    return x
    #
def sign(a):
    return int((a > 0) - (a < 0))
```

C.2.4 Biblioteca de procesamiento de los codificadores

La lectura de los codificadores en valores entre 0 y 1023 es útil solamente para movimientos limitados de los motores a menos de una revolución completa. Es necesario transformar esta lectura a un valor de ángulo en radianes, de forma que el movimiento continuo en una dirección se traduzca a un ángulo creciente o decreciente, quizá más allá de un valor de 2π. Por ello, se necesita comparar el cambio de ángulo, y sumar o restar respecto a alguna posición escogida como la inicial.

```
#!/usr/bin/env python
# -*- coding: utf8 -*-
##############################################################################
from math import *
##############################################################################
from ino_mod import *
##############################################################################
#
class Encoder:
    #
    def __init__(self, settings = {'offset' : -1}):
        #
        self.offset = settings['offset']
        self.times  = 0
        self.angle  = 0
        self.laps   = 0
        self.output_angle = 0
        self.corrected_change = 0
        #
    def compute(self, measure):
        #
        if (measure == -1):
            #
            self.output_angle = self.output_angle + self.corrected_change
            return self.output_angle
        #
        if (self.times == 0):
            self.internal_angle_last    = map(measure, 0., 1023., 0., 2. * pi)
            self.internal_angle         = map(measure, 0., 1023., 0., 2. * pi)
            if (self.offset < 0):
                self.internal_offset    = map(measure, 0., 1023., 0., 2. * pi)
            else:
                self.internal_offset    = map(self.offset, 0., 1023., 0., 2. * pi)
        #
        self.internal_angle_last        = self.internal_angle
        self.internal_angle             = map(measure, 0., 1023., 0., 2. * pi)
        #
        self.change = self.internal_angle - self.internal_angle_last
        if (abs(self.change) > pi): # es un cambio muy drastico, asumiremos que hubo un
              salto,  max RPM de 300 a 20 hz
            if (self.internal_angle_last > self.internal_angle): # ej saltar de 900 a
                 100 -> 100 -800 -> change de 800
                self.laps = self.laps + 1
            if (self.internal_angle_last <= self.internal_angle): # ej saltar de 100 a
                 900 -> 900 - 100 -> change de 800
                self.laps = self.laps - 1
        #
        self.output_angle_last          = self.output_angle
        self.output_angle               = 2 * pi * self.laps + self.internal_angle -
             self.internal_offset
        #
        self.corrected_change           = self.output_angle - self.output_angle_last
        #
        self.times                      = self.times + 1
        #
        return self.output_angle
        #
    def resetting(self, settings):
        #
        self.offset = settings['offset']
```

C.2.5 Biblioteca para filtrado Kalman

Esta implementación se ha considerado para filtrar la variable alimentada como entrada, y observar sus primeras dos derivadas. Se asumen modelos proporcionales a la identidad para las matrices de correlación de los errores.

```
#!/usr/bin/env python
# -*- coding: utf8 -*-
#################################################################################
from math import *
#################################################################################
from ino_mod import *
#################################################################################
from numpy import *
from numpy.linalg import inv, det
#################################################################################
import rospy
#################################################################################
#
class Kfilter:
    #
    def __init__(self, settings = {'Q':10. , 'R':10. , 'P0':10. , 'rate':10.}):
        #
        self.dt = 1. / float(settings['rate'])
        self.X = array([[0.], [0.], [0.]])
        self.Yn1 = array([[0.], [0.], [0.]])
        self.Yn1n1 = array([[0.], [0.], [0.]])
        self.P = diag((float(settings['P0']), float(settings['P0
            '])))
        self.A = array([[1., self.dt, 0.5 * self.dt * self.dt], [0., 1., self.dt], [0.,
             0., 1.]])
        self.Q = array([[settings['Q'], 0., 0.] , [0., settings['Q'], 0.], [0., 0.,
            settings['Q']]])
        self.Y = array([[0.], [0.], [0.]])
        self.H = array([[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]])
        self.R = array([[settings['R'], 0., 0.] , [0., settings['R'], 0.], [0., 0.,
            settings['R']]])
        #
        self.times = 0
        #
    def kf_predict(self, _X, _P, _A, _Q):
        _X = dot(_A, _X)
        _P = dot(_A, dot(_P, _A.T)) + _Q
        return (_X, _P)
        #
    def kf_update(self, _X, _P, _Y, _H, _R):
        _IM = dot(_H, _X)
        _IS = _R + dot(_H, dot(_P, _H.T))
        _K  = dot(_P, dot(_H.T, inv(_IS)))
        _X  = _X + dot(_K, (_Y - _IM))
        _P  = _P - dot(_K, dot(_IS, _K.T))
        return (_X, _P, _K, _IM, _IS)
        #
    def compute(self, measure):
        #
        self.times = self.times + 1
        #
        if (self.times < 2):
            self.Y = array([[measure], [0], [0]])
            return self.Y
            #
        # INPUT
        self.Yn1n1[0, 0] = self.Yn1[0, 0]
        self.Yn1n1[1, 0] = self.Yn1[1, 0]
        self.Yn1n1[2, 0] = self.Yn1[2, 0]
        #
        self.Yn1[0, 0] = self.Y[0, 0]
        self.Yn1[1, 0] = self.Y[1, 0]
        self.Yn1[2, 0] = self.Y[2, 0]
        # TIME UPDATE
        (self.X, self.P) = self.kf_predict(self.X, self.P, self.A, self.Q)
        #
        self.Y[0, 0] = measure
        self.Y[1, 0] = (measure - self.Yn1[0, 0]) / self.dt
        self.Y[2, 0] = (self.Yn1[1, 0] - self.Yn1n1[1, 0]) / self.dt
        #
```

```
        #(self.X, self.P, self.K, self.IM, self.IS, self.LH) = self.kf_update(self.X,
            self.P, self.Y, self.H, self.R)
        (self.X, self.P, self.K, self.IM, self.IS) = self.kf_update(self.X, self.P,
            self.Y, self.H, self.R)
        #
        return self.X
        #
    def resetting(self, settings):
        #
        self.P = diag((float(settings['P0']), float(settings['P0']), float(settings['P0
            '])))
        self.Q = array([[settings['Q'], 0., 0.] , [0., settings['Q'], 0.], [0., 0.,
            settings['Q']]])
        self.R = array([[settings['R'], 0., 0.] , [0., settings['R'], 0.], [0., 0.,
            settings['R']]])
        self.dt = 1 / float(settings['rate'])
```

C.2.6 Biblioteca de protección de los motores

Esta biblioteca trata de introducir la idea de "crédito" que un motor puede usar para moverse por algún tiempo a una entrada de voltaje más allá de algún valor considerado como seguro, de forma que una vez agotado, deba "pagarlo" con tiempo en voltajes inferiores a tal valor o se penalizara la salida máxima al motor. De esta forma, se trata de evitar el sobrecalentamiento del motor, en ausencia de sensores de corriente o de temperatura.

```
#!/usr/bin/env python
# -*- coding: utf8 -*-

##############################################################################
from math import *
##############################################################################
from ino_mod import *
##############################################################################
import rospy
from std_msgs.msg import Int16
from std_msgs.msg import Float32
##############################################################################

class Profile:
    #
    def __init__(self, settings = {'max_output' : 30, 'max_speed' : 100, 'rate' : 10, '
        heal_time_at_0pc' : 20, 'stable_point_pc' : 20}):
        #
        self.max_output         = float(settings['max_output'])
        self.max_speed          = float(settings['max_speed'])
        self.rate               = float(settings['rate'])
        self.heal_time_at_0pc   = float(settings['heal_time_at_0pc'])
        self.stable_point_pc    = float(settings['stable_point_pc'])
        #
        self.output_actual      = 0.
        self.output_des         = 0.
        self.last_output_actual = 0.
        self.last_output_des    = 0.
        self.output_limit       = 0.
        #
        self.x_extend           = self.stable_point_pc
        self.y_extend           = -100. / self.heal_time_at_0pc
        self.m_signed           = self.y_extend / self.x_extend
        self.y0_offset          = 100. / self.heal_time_at_0pc
        #
    def gainFcn(self, data):
        #
        return (self.y0_offset + self.m_signed * abs(data))
        #
    def compute(self, data):
        #
        self.last_output_actual = self.output_actual
        self.output_des = data
```

```
        self.gain_val = self.gainFcn(self.output_actual) / self.rate
        self.output_limit = constrain(self.output_limit + self.gain_val, 0, self.
            max_output)
        #
        if abs(self.output_des) > abs(self.output_limit):
            self.output_actual = self.output_limit * sign(self.output_des)
        else:
            self.output_actual = self.output_des
        #
        self.output_change = self.output_actual - self.last_output_actual
        #
        if (abs(self.output_change) > self.max_speed / self.rate):
            #
            self.output_actual = constrain(self.output_actual, self.last_output_actual
                - self.max_speed / self.rate, self.last_output_actual + self.max_speed
                / self.rate)
        #
        return self.output_actual
        #
    def resetting(self, settings):
        #
        self.max_output          = float(settings['max_output'])
        self.max_speed           = float(settings['max_speed'])
        self.rate                = float(settings['rate'])
        self.heal_time_at_0pc    = float(settings['heal_time_at_0pc'])
        self.stable_point_pc     = float(settings['stable_point_pc'])
        #
        self.x_extend            = self.stable_point_pc
        self.y_extend            = -100. / self.heal_time_at_0pc
        self.m_signed            = self.y_extend / self.x_extend
        self.y0_offset           = 100. / self.heal_time_at_0pc
        #
```

C.2.7 Interfaz a microcontrolador

Este nodo toma los mensajes de salida a los motores del péndulo y estado de comunicación, y crea la cadena de texto con suma de verificación que se envía al microcontrolador a través del protocolo serial. Si no se reciben mensajes en un periodo de tiempo de 2 segundos, se manda una cadena de apagado de los motores.

```
#!/usr/bin/env python
# -*- coding: utf8 -*-

import rospy
import time
import serial
from ino_mod import *
from std_msgs.msg import Int16

class Push_servos:
    #
    def __init__(self, node_name_override = 'push_servos'):
        #
        rospy.init_node(node_name_override)
        self.nodename = rospy.get_name()
        rospy.loginfo("Node starting with name %", self.nodename)
        self.rate = float(rospy.get_param("rate", 10))
        #
        self.ser = serial.Serial('/dev/ttyO4', 115200)
        #
        self.m1val = 0
        self.m2val = 0
        self.alval = 0
        # 0x7530 = 30000 usado como identificador de inicio de la trama
        # id, id, al, al, m1, m1, m2, m2, sum, sum
        self.datar = [0x75,0x30,0,0,0,0,0,0,0,0]
        #
        self.inittime = rospy.get_time()
        self.timelastal = -1000
        self.timelastnal = -1000
```

```
        #
        self.m1sub = rospy.Subscriber("m1", Int16, self.m1cb)
        self.m2sub = rospy.Subscriber("m2", Int16, self.m2cb)
        self.alsub = rospy.Subscriber("al", Int16, self.alcb)
        #
    def m1cb(self,data):
        #
        self.m1val = data.data
        #
    def m2cb(self, data):
        #
        self.m2val = data.data
        #
    def alcb(self, data):
        #
        self.timelastal = millis(self.inittime)
        self.alval = data.data
        #
    def update(self):
                # if al has not been received, shtudown
        if ((millis(self.inittime) - self.timelastal) > 2000):
            self.alval = 0
            self.m1val = 0
            self.m2val = 0
            self.timelastnal = millis(self.inittime)
        # wait a little after each disconnect to avoid jitter
        if ((millis(self.inittime) - self.timelastnal) < 2000):
            self.alval = 0
            self.m1val = 0
            self.m2val = 0
        #
        self.datar[2]  = (self.alval >> 8) & 255
        self.datar[3]  = (self.alval >> 0) & 255
        self.datar[4]  = (self.m1val >> 8) & 255
        self.datar[5]  = (self.m1val >> 0) & 255
        self.datar[6]  = (self.m2val >> 8) & 255
        self.datar[7]  = (self.m2val >> 0) & 255
        self.datar[8]  = ((self.alval + self.m1val + self.m2val) >> 8) & 255
        self.datar[9]  = ((self.alval + self.m1val + self.m2val) >> 0) & 255
        #
        self.ser.write(self.datar)
        #
    def spin(self):
                #
        r = rospy.Rate(self.rate)
        while not rospy.is_shutdown():
            self.update()
            r.sleep()
        #
if __name__ == '__main__':
    """ main """
    push_servos = Push_servos()
    push_servos.spin()
```

C.2.8 Nodo de control

Este nodo es el más complejo del sistema, y se encarga de procesar y filtrar los datos de los codificadores y las IMU, leer la velocidad y ángulo deseado, calcular las salidas de control, publicar los datos de estado de variables de interés, y hacer una interfaz de reconfiguración para cambiar los parámetros de control en tiempo real. Si bien este código podría distribuirse en varios nodos, el retraso en el tiempo de comunicaciones y la carga de procesamiento adicional se ha evaluado demasiado alta para ello.

```
#!/usr/bin/env python
# -*- coding: utf8 -*-
#
#############################################################################
```

```
from math import *
################################################################################
import rospy
from std_msgs.msg import Bool
from std_msgs.msg import Int16
from std_msgs.msg import Float32
from dynamic_reconfigure.server import Server
################################################################################
from volti.cfg import VOLConfig
################################################################################
from ino_mod import *
from kfilter import *
from pid_pos import *
from pid_vel import *
from encoder import *
from profile import *
################################################################################
from numpy import *
from numpy.linalg import inv
################################################################################
from volti.msg import float32_3
from volti.msg import float32_12
from volti.msg import pid_ext
################################################################################
#
class Control:
    #
    def __init__(self, node_name_default = 'control'):
        #
        self.GRLinit(node_name_default)
        self.SRVinit()
        #
        self.pid_vel_m1          = PID_vel(self.vel_settings)
        self.pid_vel_m2          = PID_vel(self.vel_settings)
        #
        #self.pid_vel_vel          = PID_vel(self.vel_settings)
        #self.pid_pos_ang          = PID_pos(self.pos_settings)
        #
        self.integral_ang        = 0
        self.integral_ang_int    = 0
        self.integral_vel        = 0
        #
        # filtro de entradas del encoder
        self.kf_settings         = {'Q' : 10, 'R' : 10, 'P0' : 10, 'rate' : self.rate}
        #
        # filtros kalman de los encoders
        self.filter_e1           = Kfilter(self.kf_settings)
        self.filter_e2           = Kfilter(self.kf_settings)
        self.filter_plate        = Kfilter(self.kf_settings)
        self.filter_pendu        = Kfilter(self.kf_settings)
        #
        # objeto de lectura del encoder
        self.enc_settings = {'offset' : -1}
        #
        self.enc_1 = Encoder(self.enc_settings)
        self.enc_2 = Encoder(self.enc_settings)
        #
        # Objeto de limitacion de la salida a los motores
        self.profile_settings = {'max_output' : 30, 'max_speed' : 10000, 'rate' : self.
            rate, 'heal_time_at_0pc' : 20, 'stable_point_pc' : 20}
        #
        self.profile_m1 = Profile(self.profile_settings)
        self.profile_m2 = Profile(self.profile_settings)
        #
        self.value_m1 = 0
        self.angle_m1 = 0
        self.speed_m1 = 0
        self.accel_m1 = 0
        #
        self.value_m2 = 0
        self.angle_m2 = 0
        self.speed_m2 = 0
        self.accel_m2 = 0
        #
        self.vel_del_des = 0
        self.ang_lat_des = 0
        #
        self.rollPlate  = 0
        self.pitchPlate = 0
        self.rollPendu  = 0
        self.pitchPendu = 0
        #
```

```
        self.con_mode = 0
        #
        self.powsub = rospy.Subscriber("con_mode", Int16, self.modecb)
        #
        # Asociaciones con publicadores y suscriptores
        self.e1_proc_msg = float32_3()
        self.m1     = rospy.Publisher("m1",      Int16)
        self.e1pub = rospy.Publisher("e1_proc", float32_3)
        #
        self.e2_proc_msg = float32_3()
        self.m2     = rospy.Publisher("m2",      Int16)
        self.e2pub = rospy.Publisher("e2_proc", float32_3)
        #
        self.normalizedkp0rps = 1
        self.normalizedkp1rps = 1
        self.normalizedkpvel  = 1
        self.normalizedkmvel  = 1
        #
        self.rplate_proc_msg = float32_3()
        self.rpendu_proc_msg = float32_3()
        self.rollPenduPub = rospy.Publisher("rpendu", float32_3)
        self.rollPlatePub = rospy.Publisher("rplate", float32_3)
        #
        self.pidangmsg    = pid_ext()
        self.pidvelm1msg  = pid_ext()
        self.pidvelm2msg  = pid_ext()
        self.pidangpub    = rospy.Publisher("pidang", pid_ext)
        self.pidvelm1vel  = rospy.Publisher("pidvm1", pid_ext)
        self.pidvelm2vel  = rospy.Publisher("pidvm2", pid_ext)
        #
        self.err_proc_msg = float32_3()
        self.errPub       = rospy.Publisher("errpro", float32_3)
        self.avg_vel_pub  = rospy.Publisher("avgvel", Float32)
        self.exp_pub      = rospy.Publisher("expdec", Float32)
        #
        self.e1           = rospy.Subscriber("e1",    Int16, self.e1cb)  # entrada del
            encoder 1
        self.e2           = rospy.Subscriber("e2",    Int16, self.e2cb)  # entrada del
            encoder 1
        #
        self.veldeldessub = rospy.Subscriber("vel_delante_des", Float32, self.
            veldeldescb)
        self.anglatdessub = rospy.Subscriber("ang_lateral_des", Float32, self.
            anglatdescb)
        #
        self.subImuPlate  = rospy.Subscriber('imu_plate_3', float32_3, self.imuplatecb)
        self.subImuPendu  = rospy.Subscriber('imu_pendu_3', float32_3, self.imupenducb)
        #
        self.srv = Server(VOLConfig, self.SRVcallback)
        #
    def modecb(self, data):
        #
        self.con_mode = data.data
        #
    def veldeldescb(self, data):
        #
        self.vel_del_des = data.data
        #
    def anglatdescb(self, data):
        #
        self.ang_lat_des = data.data
        #
    def imuplatecb(self, data):
        #
        self.rollPlate  = data.data[0] - 0.06
        self.pitchPlate = data.data[1]
        #
    def imupenducb(self, data):
        #
        #self.rollPendu  = self.angleRange(data.data[0] + 3.1416)
        if (data.data[0] < 0):
            self.rollPendu = data.data[0] + 3.1416
        if (data.data[0] > 0):
            self.rollPendu = data.data[0] - 3.1416
        self.rollPendu = self.rollPendu + 0.05
        self.pitchPendu = data.data[1]
        #
    def GRLinit(self, node_name_default):
        #
        rospy.init_node(node_name_default)
        self.nodename = rospy.get_name()
        rospy.loginfo("Started node %s", self.nodename)
```

```
        #
        self.rate = float(rospy.get_param("rate", 10))
        self.times = 0
        #
    def SRVinit(self):
        #
        # pid_out da el valor de salida al motor de -100 a 100
        # pid_ang da el valor del angulo del encoder en su marco de referencia (lafrom
            std_msgs.msg import Bool IMU da el angulo de la esfera, en general, dado
        # que cada motor puede dar una cantidad potencialmente infinita de vueltas, no
            es posible hacer una correspondencia uno a uno
        # con el angulo real del motor. Los encoders se consideran digitales de 0 a
            1023 en una vuelta completa
        # pid_vel da el valor de la velocidad del angulo del encoder en su marco de
            referencia, se calcula en base al valor anterior
        #
        self.pos_settings = {'kp0rps' : 0, 'kp1rps' : 0, 'ki' : 0, 'kd' : 0, 'ka' : 0,
            'ks' : 0, 'umbral' : 0, 'umbral_int' : 0, 'umbral_oof' : 0, 'div_minimal'
            : 1, 'div_ang2vel' : 1, 'div_modes' : 0, 'range' : 0, 'rate' : self.rate}
        self.vel_settings = {'kp' : 0, 'ki' : 0, 'kd' : 0, 'km' : 0, 'ka' : 0, 'ks' :
            0, 'umbral' : 0, 'ki_dec' : 0, 'range' : 0, 'rate' : self.rate}
        #
    def SRVcallback(self, config, level):
        #
        rospy.loginfo("reconfiguring")
        #
        #values catched from mouse event from rqt_gui
        self.normalizedkp0rps = float(config['kp0rps_pos'])
        self.normalizedkp1rps = float(config['kp1rps_pos'])
        self.normalizedkpvel  = float(config['kp_vel'])
        self.normalizedkmvel  = float(config['km_vel'])
        #
        self.pos_settings['kp0rps'] = float(config['kp0rps_pos'])
        self.pos_settings['kp1rps'] = float(config['kp1rps_pos'])
        self.pos_settings['ki']     = float(config['ki_pos'])
        self.pos_settings['kd']     = float(config['kd_pos'])
        self.pos_settings['ka']     = float(config['ka_pos'])
        self.pos_settings['ks']     = float(config['ks_pos'])
        # variables de limitacion adicionales
        self.pos_settings['umbral']     = float(config['umbral_pos'])
        self.pos_settings['umbral_int'] = float(config['umbral_int'])
        self.pos_settings['umbral_oof'] = float(config['umbral_oof'])
        #
        self.pos_settings['div_minimal'] = float(config['div_minimal'])
        self.pos_settings['div_ang2vel'] = float(config['div_ang2vel'])
        #self.pos_settings['div_modes']   = float(config['div_modes'])
        #
        self.pos_settings['range']      = float(config['range_pos'])
        # parametros de posicion, hacer parametros posteriormente, usar dynamic
        self.vel_settings['kp']     = float(config['kp_vel'])
        self.vel_settings['ki']     = float(config['ki_vel'])
        self.vel_settings['kd']     = float(config['kd_vel'])
        self.vel_settings['km']     = float(config['km_vel'])
        self.vel_settings['ka']     = float(config['ka_vel'])
        self.vel_settings['ks']     = float(config['ks_vel'])
        # variables de limitacion adicionales
        self.vel_settings['umbral'] = float(config['umbral_vel'])
        self.vel_settings['range']  = float(config['range_vel'])
        #
        #self.pid_pos_ang.resetting(self.pos_settings)
        #self.pid_vel_vel.resetting(self.vel_settings)
        self.pid_vel_m1.resetting(self.vel_settings)
        self.pid_vel_m2.resetting(self.vel_settings)
        #
        self.kf_settings['P0']   = float(config['P0'])
        self.kf_settings['Q']    = float(config['Q'])
        self.kf_settings['R']    = float(config['R'])
        self.kf_settings['rate'] = float(config['rate'])
        #
        self.filter_e1.resetting(self.kf_settings)
        self.filter_e1.resetting(self.kf_settings)
        self.filter_plate.resetting(self.kf_settings)
        self.filter_pendu.resetting(self.kf_settings)
        #
        return config
        #
    def e1cb(self, data):
        #
        self.value_m1 = data.data
        self.proc_m1  = self.enc_1.compute(self.value_m1)
        self.X_m1     = self.filter_e1.compute(self.proc_m1)
        #
```

```
        self.angle_m1 = self.X_m1[0, 0]
        self.speed_m1 = self.X_m1[1, 0]
        self.accel_m1 = self.X_m1[2, 0]
        #
        self.e1_proc_msg.data[0] = self.angle_m1
        self.e1_proc_msg.data[1] = self.speed_m1
        self.e1_proc_msg.data[2] = self.accel_m1
        #
    def e2cb(self, data):
        #
        self.value_m2 = data.data
        self.proc_m2  = self.enc_2.compute(self.value_m2)
        self.X_m2     = self.filter_e2.compute(self.proc_m2)
        #
        self.angle_m2 = self.X_m2[0, 0]
        self.speed_m2 = self.X_m2[1, 0]
        self.accel_m2 = self.X_m2[2, 0]
        #
        self.e2_proc_msg.data[0] = self.angle_m2
        self.e2_proc_msg.data[1] = self.speed_m2
        self.e2_proc_msg.data[2] = self.accel_m2
        #
    def getKp(self, speed):
        #
        return constrain(self.pos_settings['kp0rps'] - (self.pos_settings['kp0rps'] -
            self.pos_settings['kp1rps']) * abs(speed), 0, 10000)
        #
    def controller(self):
        #
        self.ang_plate = self.rollPlate
        self.ang_pendu = self.rollPendu
        #
        self.X_plate   = self.filter_plate.compute(self.ang_plate)
        self.X_pendu   = self.filter_pendu.compute(self.ang_pendu)
        #
        self.ang_plate = self.X_plate[0, 0]
        self.ang_pendu = self.X_plate[0, 0]
        #
        self.vel_plate = self.X_plate[1, 0]
        self.vel_pendu = self.X_pendu[1, 0]
        #
        self.ace_plate = self.X_plate[2, 0]
        self.ace_pendu = self.X_pendu[2, 0]
        #
        self.avg_vel_m1_m2     = (self.speed_m1 + self.speed_m2) / 2;
        self.avg_vel_m1_m2_abs = (abs(self.speed_m1) + abs(self.speed_m2)) / 2;
        #
        self.ang_control = self.ang_plate - self.ang_lat_des
        self.vel_control = self.vel_plate
        self.ace_control = self.ace_plate
        #
        self.decexp = exp(- abs(self.avg_vel_m1_m2) * self.pos_settings['div_minimal'])
        #
        #
        #
        self.rplate_proc_msg.data[0] = self.ang_plate
        self.rplate_proc_msg.data[1] = self.vel_plate
        self.rplate_proc_msg.data[2] = self.ace_plate
        #
        self.rpendu_proc_msg.data[0] = self.ang_pendu
        self.rpendu_proc_msg.data[1] = self.vel_pendu
        self.rpendu_proc_msg.data[2] = self.ace_pendu
        #
        #
        #
        self.err_proc_msg.data[0] = self.ang_control
        self.err_proc_msg.data[1] = self.vel_control
        self.err_proc_msg.data[2] = self.ace_control
        #
        #
        #
        self.exp_pub.publish(self.decexp)
        self.errPub.publish(self.err_proc_msg)
        self.avg_vel_pub.publish(self.avg_vel_m1_m2)
        self.e1pub.publish(self.e1_proc_msg)
        self.e2pub.publish(self.e2_proc_msg)
        self.rollPenduPub.publish(self.rpendu_proc_msg)
        self.rollPlatePub.publish(self.rplate_proc_msg)
        #
        #
        #
        self.fi     = self.ang_plate - self.ang_lat_des
```

```
self.beta  = self.ang_pendu - self.ang_plate
self.fip   = self.vel_plate
self.betap = self.vel_pendu - self.vel_plate
#
#
#
self.u = -((0.4278062687504448 + 0.032367102026846464*cos(self.fi) - \
0.013726746118715057*cos(2*(self.beta + self.fi)))*(self.fi*(81 + \
(2620.5349931237606*(2.529793316296556 - 1.*cos(self.beta + \
self.fi))**2)/(13.217317645416836 + 1.*cos(self.fi) - \
0.42409561743679086*cos(2*(self.beta + self.fi)))**2 + \
(-1165.5275189205913 + 460.72045151374016*cos(self.beta + \
self.fi))/(13.217317645416836 + 1.*cos(self.fi) - \
0.42409561743679086*cos(2*(self.beta + self.fi)))) + 36*self.fip*(3 + \
(-0.8061597714965171 + 1.0000000000000002*self.fip*sin(self.fi) + \
(-4.2914970338490175*self.betap - \
4.2914970338490175*self.fip)*sin(self.beta + \
self.fi))/(13.217317645416836 + 1.*cos(self.fi) - \
0.42409561743679086*cos(2*(self.beta + \
self.fi))))))/(90*(-0.419163868 + 0.16569095399999997*cos(self.beta + \
self.fi)))
#
#
#
self.sal_u_pc = self.u * 100 / 13.6
self.sal_u_pc_fix = sign(self.sal_u_pc) * 5 * self.decexp + self.sal_u_pc
#
#
if abs(self.ang_control) < self.pos_settings['umbral']:
    self.ang_control_tmp = 0
else:
    self.ang_control_tmp = self.ang_control
#
#
self.pidangmsg.kp = self.getKp(abs(self.avg_vel_m1_m2)) * self.ang_control_tmp
self.pidangmsg.kd = self.pos_settings['kd'] * self.vel_control
self.pidangmsg.ka = self.pos_settings['ka'] * self.ace_control
self.pidangmsg.ks = self.pos_settings['ks'] * sin(self.ang_control)
#
self.salida_m1_ang = self.pidangmsg.kp + self.pidangmsg.kd + self.pidangmsg.ka
    - self.pidangmsg.ks
#
self.integral_ang  = constrain(self.integral_ang + self.ang_control / self.rate
    , -1, 1)
if abs(self.ang_control) < self.pos_settings['umbral_int']:
    self.integral_ang = 0
#
self.integral_ang_int  = constrain(self.integral_ang_int + self.ang_control /
    self.rate, -1, 1)
if abs(self.ang_control) > self.pos_settings['umbral_int']:
    self.integral_ang_int = 0
#
self.pidangmsg.ki = 3 * self.integral_ang_int * self.decexp
#
self.salida_m1_ang = self.salida_m1_ang + self.pidangmsg.ki
#
if abs(self.ang_control) < self.pos_settings['umbral_oof']:
    self.salida_m1_ang = sign(self.salida_m1_ang) * 5 * self.decexp
else:
    self.salida_m1_ang = sign(self.salida_m1_ang) * 5 * self.decexp + self.
        salida_m1_ang
#
self.salida_m1_ang = constrain(self.salida_m1_ang, -self.pos_settings['range'],
     self.pos_settings['range'])
self.salida_m1_ang = self.salida_m1_ang + self.pos_settings['ki'] * self.
    integral_ang
self.pidangmsg.out = self.salida_m1_ang
#
#
#self.salida_m1_ang = constrain(self.sal_u_pc_fix  + self.pidangmsg.ki, -self.
    pos_settings['range'], self.pos_settings['range'])
#self.pidangmsg.out = self.salida_m1_ang
#
#
self.vel_m1_des = self.vel_del_des + self.ang_control / self.pos_settings['
    div_ang2vel']
self.vel_m2_des = self.vel_del_des - self.ang_control / self.pos_settings['
    div_ang2vel']
#
self.vel_m1_err = self.vel_m1_des - self.speed_m1
self.vel_m2_err = self.vel_m2_des - self.speed_m2
#
```

```
        #
        #
        self.pidvelm1msg.kp  = self.vel_settings['kp'] * self.vel_m1_err
        self.pidvelm1msg.kd  = self.vel_settings['kd'] * self.accel_m1
        self.pidvelm1msg.km  = self.vel_settings['km'] * self.vel_m1_des
        self.pidvelm1msg.sub = (self.vel_settings['kp'] + self.vel_settings['km']) *
            self.ang_control / self.pos_settings['div_ang2vel']
        #
        self.pidvelm2msg.kp  = self.vel_settings['kp'] * self.vel_m2_err
        self.pidvelm2msg.kd  = self.vel_settings['kd'] * self.accel_m2
        self.pidvelm2msg.km  = self.vel_settings['km'] * self.vel_m2_des
        self.pidvelm2msg.sub = (self.vel_settings['kp'] + self.vel_settings['km']) * -
            self.ang_control / self.pos_settings['div_ang2vel']
        #
        #
        #
        self.salida_m1_vel = self.pidvelm1msg.kp - self.pidvelm1msg.kd + self.
            pidvelm1msg.km
        self.salida_m2_vel = self.pidvelm2msg.kp - self.pidvelm2msg.kd + self.
            pidvelm2msg.km
        #
        self.salida_m1_vel = self.salida_m1_vel + (self.vel_settings['kp'] + self.
            vel_settings['km']) * self.ang_control / self.pos_settings['div_ang2vel']
        self.salida_m2_vel = self.salida_m2_vel + (self.vel_settings['kp'] + self.
            vel_settings['km']) * -self.ang_control / self.pos_settings['div_ang2vel']
        #
        self.salida_m1_vel = constrain(self.salida_m1_vel, -self.vel_settings['range'],
             self.vel_settings['range'])
        self.salida_m2_vel = constrain(self.salida_m2_vel, -self.vel_settings['range'],
             self.vel_settings['range'])
        #
        self.out_pos_m1 = self.profile_m1.compute( self.salida_m1_ang * self.decexp +
            self.salida_m1_vel )
        self.out_pos_m2 = self.profile_m2.compute( -self.salida_m1_ang * self.decexp +
            self.salida_m2_vel )
        #
        #
        self.pidvelm1msg.out = self.salida_m1_vel
        self.pidvelm2msg.out = self.salida_m2_vel
        #
        #
        self.pidvelm1vel.publish(self.pidvelm1msg)
        self.pidvelm2vel.publish(self.pidvelm2msg)
        #
        #
        if (self.con_mode is 1):
            return
        #
        self.m1.publish(int((self.out_pos_m1) * 100))
        self.m2.publish(int((self.out_pos_m2) * 100))
        #
    def update(self):
        #
        self.times = self.times + 1
        #
        self.controller()
        #
    def spin(self):
        #
        r = rospy.Rate(self.rate)
        while not rospy.is_shutdown():
            self.update()
            r.sleep()
            #
if __name__ == '__main__':
    #
    """ main """
    control = Control()
    control.spin()
    #
```

C.2.9 Nodo de recolección de datos

Se ha usado para suscribirse a las variables de interés durante las pruebas y generar un archivo de datos separado por comas para su importación en una suite de hojas

de cálculo, y permitir generar gráficas de tales pruebas.

```
#!/usr/bin/env python
# -*- coding: utf8 -*-
#
#############################################################################
import rospy
from    std_msgs.msg import Bool
from    std_msgs.msg import Int16
from    std_msgs.msg import Float32
from    std_msgs.msg import String
from    volti.msg    import float32_3
from    volti.msg    import float32_12
import time
import atexit
#############################################################################
#
class Logger:
    #
    def __init__(self, node_name_default = 'logger'):
        #
        self.f = open(str(time.strftime("%y%m%d%H%M%S")) + ".csv", 'w')
        atexit.register(self.onclose)
        #
        rospy.init_node(node_name_default)
        self.nodename = rospy.get_name()
        rospy.loginfo("Node starting with name %s", self.nodename)
        self.rate = int(rospy.get_param("rate", '100'))
        #
        self.times = 0
        #
        self.alertval = ""
        self.anglatdesval = 0.
        self.avgvelval = 0.
        self.e1val = 0
        self.e2val = 0
        self.e1pval = float32_3()
        self.e2pval = float32_3()
        self.erpval = float32_3()
        self.decval = 0.
        self.ipendval = float32_3()
        self.iplatval = float32_3()
        self.m1val = 0
        self.m2val = 0
        self.rpendval = float32_3()
        self.rplatval = float32_3()
        self.velveldesval = 0.
        #
        self.s1  = rospy.Subscriber("alert",           String,    self.alertcb)
        self.s2  = rospy.Subscriber("ang_lateral_des", Float32,   self.anglatdescb)
        self.s3  = rospy.Subscriber("avgvel",          Float32,   self.avgvelcb)
        self.s4  = rospy.Subscriber("e1",              Int16,     self.e1cb)
        self.s5  = rospy.Subscriber("e2",              Int16,     self.e2cb)
        self.s6  = rospy.Subscriber("e1_proc",         float32_3, self.e1pcb)
        self.s7  = rospy.Subscriber("e2_proc",         float32_3, self.e2pcb)
        self.s8  = rospy.Subscriber("errpro",          float32_3, self.erpcb)
        self.s9  = rospy.Subscriber("expdec",          Float32,   self.deccb)
        self.s10 = rospy.Subscriber("imu_pendu_3",     float32_3, self.ipendcb)
        self.s11 = rospy.Subscriber("imu_plate_3",     float32_3, self.iplatcb)
        self.s12 = rospy.Subscriber("m1",              Int16,     self.m1cb)
        self.s13 = rospy.Subscriber("m2",              Int16,     self.m2cb)
        self.s14 = rospy.Subscriber("rpendu",          float32_3, self.rpendcb)
        self.s15 = rospy.Subscriber("rplate",          float32_3, self.rplatcb)
        self.s16 = rospy.Subscriber("vel_delante_des", Float32,   self.velveldescb)
        #
    def alertcb(self, data):
        self.alertval = data.data
    def anglatdescb(self, data):
        self.anglatdesval = data.data
    def avgvelcb(self, data):
        self.avgvelval = data.data
    def e1cb(self, data):
        self.e1val = data.data
    def e2cb(self, data):
        self.e2val = data.data
    def e1pcb(self, data):
        self.e1pval = data
    def e2pcb(self, data):
        self.e2pval = data
    def erpcb(self, data):
        self.erpval = data
    def deccb(self, data):
        self.decval = data.data
```

```
    def ipendcb(self, data):
        self.ipendval = data
    def iplatcb(self, data):
        self.iplatval = data
    def m1cb(self, data):
        self.m1val = data.data
    def m2cb(self, data):
        self.m2val = data.data
    def rpendcb(self, data):
        self.rpendval = data
    def rplatcb(self, data):
        self.rplatval = data
    def velveldescb(self, data):
        self.velveldesval = data.data
    def update(self):
        #
        self.times = self.times + 1
        self.f.write(str(self.times / 10) + "." + str(((self.times %10) * 1000)/10).
            zfill(3)  + "," + \
        str(self.alertval).replace(',', '␣') + "," + \
        str(self.anglatdesval) + "," + \
        str(self.avgvelval) + "," + \
        str(self.e1val) + "," + \
        str(self.e2val) + "," + \
        str(self.e1pval.data[0]) + "," + \
        str(self.e1pval.data[1]) + "," + \
        str(self.e1pval.data[2]) + "," + \
        str(self.e2pval.data[0]) + "," + \
        str(self.e2pval.data[1]) + "," + \
        str(self.e2pval.data[2]) + "," + \
        str(self.erpval.data[0]) + "," + \
        str(self.erpval.data[1]) + "," + \
        str(self.erpval.data[2]) + "," + \
        str(self.decval) + "," + \
        str(self.ipendval.data[0]) + "," + \
        str(self.iplatval.data[0]) + "," + \
        str(self.m1val) + "," + \
        str(self.m2val) + "," + \
        str(self.rpendval.data[0]) + "," + \
        str(self.rpendval.data[1]) + "," + \
        str(self.rpendval.data[2]) + "," + \
        str(self.rplatval.data[0]) + "," + \
        str(self.rplatval.data[1]) + "," + \
        str(self.rplatval.data[2]) + "," + \
        str(self.velveldesval) + "," + \
        '\n')
        #
        #
    def onclose(self):
        #
        self.f.close()
        #
    def spin(self):
        #
        self.f.write("time␣,␣alert␣,␣anglatdes␣,␣avgvel␣,␣e1␣,␣e2␣,␣e1k␣,␣e1pk␣,␣e1ppk␣
            ,␣e2k␣,␣e2pk␣,␣e2ppk␣,␣errk␣,␣errpk␣,␣errppk␣,␣dec␣,␣ipenduroll␣,␣
            iplateroll␣,␣m1␣,␣m2␣,␣rpenduk␣,␣rpendupk␣,␣rpenduppk␣,rplatek␣,␣rplatepk␣
            ,␣rplateppk␣,␣veldeldes" + '\n')
        #
        r = rospy.Rate(self.rate)
        while not rospy.is_shutdown():
            self.update()
            r.sleep()
            #
            #
if __name__ == '__main__':
    #
    """␣main␣"""
    logger = Logger()
    logger.spin()
```

C.2.10 Filtrado de las unidades de medición inercial (IMU)

Sólo se muestra el código de "alto nivel" usado para el procesamiento de las IMU, usando el algoritmo DCM para la IMU Razor Stick. Las clases usadas para la lec-

tura de los sensores por medio del protocolo I^2C no se muestran, pero se incluyen en el repositorio del proyecto. Una implementación fuera de ROS podría delegar la tarea de lectura al microcontrolador, permitiendo de todas maneras hacer el procesamiento con esta clase.

Nodo de interfaz con ROS:

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include "std_msgs/Float32.h"

#include <sstream>

#include <iostream>
#include "imu_lib.h"
#include <cmath>

#include "volti_msgs/float32_3.h"

using namespace std;

float pi= 3.1415926535897;

////////////////////////////////////////////////////////////////////////////

int main(int argc, char **argv)
{
  ros::init(argc, argv, "imu_node");
  ros::NodeHandle n;

  ros::Publisher imu1_pub = n.advertise<volti_msgs::float32_3>("imu_pendu_3", 1);
  ros::Publisher imu2_pub = n.advertise<volti_msgs::float32_3>("imu_plate_3", 1);

  //publicar cada 20 ms
  ros::Rate loop_rate(50);

  // modulos de salida en el BBB
  IMU imu_plate(1);
  IMU imu_pendu(2);

  volti_msgs::float32_3 imu_plate_msg;
  volti_msgs::float32_3 imu_pendu_msg;

  int unsigned long times = 0;

  while (ros::ok())
  {
    imu_plate.loop();
    imu_pendu.loop();

    times++;

    if (times %5 == 1) {

    imu_plate_msg.data[0] = imu_plate.roll;
    imu_plate_msg.data[1] = imu_plate.pitch;
    imu_plate_msg.data[2] = imu_plate.yaw;
    imu1_pub.publish(imu_plate_msg);

    imu_pendu_msg.data[0] = imu_pendu.roll;
    imu_pendu_msg.data[1] = imu_pendu.pitch;
    imu_pendu_msg.data[2] = imu_pendu.yaw;
    imu2_pub.publish(imu_pendu_msg);

    }

    ros::spinOnce();
    loop_rate.sleep();
  }

  return 0;
}
```

Archivo de cabecera para `imu_lib.h`, detallando las firmas de los métodos de

procesamiento de la IMU:

```
#ifndef IMU_H_
#define IMU_H_

#include "ADXL345.h"
#include "gyro.h"
#include "hmc5883.h"

#define HW__VERSION_CODE 10724 // SparkFun "9DOF Sensor Stick" version "SEN-10724" (
    HMC5883L magnetometer)
#define OUTPUT__BAUD_RATE 57600
#define OUTPUT__DATA_INTERVAL 20  // in milliseconds
#define OUTPUT__MODE_CALIBRATE_SENSORS 0 // Outputs sensor min/max values as text for
    manual calibration
#define OUTPUT__MODE_ANGLES 1 // Outputs yaw/pitch/roll in degrees
#define OUTPUT__MODE_SENSORS_CALIB 2 // Outputs calibrated sensor values for all 9 axes
#define OUTPUT__MODE_SENSORS_RAW 3 // Outputs raw (uncalibrated) sensor values for all
    9 axes
#define OUTPUT__MODE_SENSORS_BOTH 4 // Outputs calibrated AND raw sensor values for all
     9 axes
#define OUTPUT__MODE_CALIBRATE_BOTH_SENSORS_ANGLES 5
#define OUTPUT__FORMAT_TEXT 0 // Outputs data as text
#define OUTPUT__FORMAT_BINARY 1 // Outputs data as binary float
#define OUTPUT__STARTUP_STREAM_ON true  // true or false
#define OUTPUT__HAS_RN_BLUETOOTH false  // true or false

#define ACCEL_X_MIN ((float) -250)
#define ACCEL_X_MAX ((float) 250)
#define ACCEL_Y_MIN ((float) -250)
#define ACCEL_Y_MAX ((float) 250)
#define ACCEL_Z_MIN ((float) -250)
#define ACCEL_Z_MAX ((float) 250)

#define MAGN_X_MIN ((float) -600)
#define MAGN_X_MAX ((float) 600)
#define MAGN_Y_MIN ((float) -600)
#define MAGN_Y_MAX ((float) 600)
#define MAGN_Z_MIN ((float) -600)
#define MAGN_Z_MAX ((float) 600)

#define GYRO_AVERAGE_OFFSET_X ((float) 0.0)
#define GYRO_AVERAGE_OFFSET_Y ((float) 0.0)
#define GYRO_AVERAGE_OFFSET_Z ((float) 0.0)

#define DEBUG__NO_DRIFT_CORRECTION false
#define DEBUG__PRINT_LOOP_TIME false

#ifndef HW__VERSION_CODE
  #error YOU HAVE TO SELECT THE HARDWARE YOU ARE USING! See "HARDWARE␣OPTIONS" in "USER
      ␣SETUP␣AREA" at top of Razor_AHRS.pde (or .ino)!
#endif

#define ACCEL_X_OFFSET ((ACCEL_X_MIN + ACCEL_X_MAX) / 2.0f)
#define ACCEL_Y_OFFSET ((ACCEL_Y_MIN + ACCEL_Y_MAX) / 2.0f)
#define ACCEL_Z_OFFSET ((ACCEL_Z_MIN + ACCEL_Z_MAX) / 2.0f)
#define ACCEL_X_SCALE (GRAVITY / (ACCEL_X_MAX - ACCEL_X_OFFSET))
#define ACCEL_Y_SCALE (GRAVITY / (ACCEL_Y_MAX - ACCEL_Y_OFFSET))
#define ACCEL_Z_SCALE (GRAVITY / (ACCEL_Z_MAX - ACCEL_Z_OFFSET))

#define MAGN_X_OFFSET ((MAGN_X_MIN + MAGN_X_MAX) / 2.0f)
#define MAGN_Y_OFFSET ((MAGN_Y_MIN + MAGN_Y_MAX) / 2.0f)
#define MAGN_Z_OFFSET ((MAGN_Z_MIN + MAGN_Z_MAX) / 2.0f)
#define MAGN_X_SCALE (100.0f / (MAGN_X_MAX - MAGN_X_OFFSET))
#define MAGN_Y_SCALE (100.0f / (MAGN_Y_MAX - MAGN_Y_OFFSET))
#define MAGN_Z_SCALE (100.0f / (MAGN_Z_MAX - MAGN_Z_OFFSET))

#define GYRO_GAIN 0.06957 // Same gain on all axes
#define GYRO_SCALED_RAD(x) (x * TO_RAD(GYRO_GAIN)) // Calculate the scaled gyro
    readings in radians per second

#define Kp_ROLLPITCH 0.02f
#define Ki_ROLLPITCH 0.00002f
#define Kp_YAW 1.2f
#define Ki_YAW 0.00002f

#define STATUS_LED_PIN 13  // Pin number of status LED
#define GRAVITY 256.0f // "1G reference" used for DCM filter and accelerometer
    calibration
#define TO_RAD(x) (x * 0.01745329252)  // *pi/180
#define TO_DEG(x) (x * 57.2957795131)  // *180/pi

#define ACCEL_ADDRESS ((short) 0x53) // 0x53 = 0xA6 / 2
```

```
#define MAGN_ADDRESS  ((short) 0x1E) // 0x1E = 0x3C / 2
#define GYRO_ADDRESS  ((short) 0x68) // 0x68 = 0xD0 / 2

class IMU
{   private:

    void init_matrix(float array[][3],float A[][3]);
    float constrain(float X,float A, float B);
    public:

    Adxl345* Acelerometro;
    Gyro* Gyrometro;
    MAG* Magnetometro;
    // Select your startup output mode and format here!
    short output_mode;
    short output_format;

    // If set true, an error message will be output if we fail to read sensor data.
    // Message format: "!ERR: reading <sensor>", followed by "\r\n".
    bool output_errors;  // true or false

    // Sensor variables
    float accel[3] = {0, 0, 0}; // Actually stores the NEGATED acceleration (equals
        gravity, if board not moving).
    float accel_min[3] = {0, 0, 0};
    float accel_max[3] = {0, 0, 0};

    float magnetom[3] = {0, 0, 0};
    float magnetom_min[3] = {0, 0, 0};
    float magnetom_max[3] = {0, 0, 0};
    float magnetom_tmp[3] = {0, 0, 0};

    float gyro[3] = {0, 0, 0};
    float gyro_average[3] = {0, 0, 0};
    short gyro_num_samples = 0;

    // DCM variables
    float MAG_Heading = 0;
    float Accel_Vector[3] = {0, 0, 0};// Store the acceleration in a vector
    float Gyro_Vector[3] = {0, 0, 0}; // Store the gyros turn rate in a vector
    float Omega_Vector[3] = {0, 0, 0}; // Corrected Gyro_Vector data
    float Omega_P[3] = {0, 0, 0}; // Omega Proportional correction
    float Omega_I[3] = {0, 0, 0}; // Omega Integrator
    float Omega[3] = {0, 0, 0};
    float errorRollPitch[3] = {0, 0, 0};
    float errorYaw[3] = {0, 0, 0};
    float DCM_Matrix[3][3] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};;
    float Update_Matrix[3][3]  = {{0, 1, 2}, {3, 4, 5}, {6, 7, 8}};
    float Temporary_Matrix[3][3] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};

    // Euler angles
    float yaw = 0;
    float pitch = 0;
    float roll = 0;

    // DCM timing in the main loop
    unsigned long timestamp = 0;
    unsigned long timestamp_old = 0;
    float G_Dt = 0; // Integration time for DCM algorithm

    // More output-state variables
    bool output_stream_on = 0;
    bool output_single_on = 0;
    short curr_calibration_sensor = 0;
    bool reset_calibration_session_flag = 0;
    short num_accel_errors = 0;
    short num_magn_errors = 0;
    short num_gyro_errors = 0;

    short module = 1;
    IMU(int module);

    void Compass_Heading();
    void Normalize(void);
    void Drift_correction(void);
    void Matrix_update(void);
    void Euler_angles(void);
    float Vector_Dot_Product(const float v1[3], const float v2[3]);
    void Vector_Cross_Product(float out[3], const float v1[3], const float v2[3]);
    void Vector_Scale(float out[3], const float v[3], float scale);
    void Vector_Add(float out[3], const float v1[3], const float v2[3]);
    void Matrix_Multiply(const float a[3][3], const float b[3][3], float out[3][3]);
```

```
    void Matrix_Vector_Multiply(const float a[3][3], const float b[3], float out[3]);
    void init_rotation_matrix(float m[3][3], float yaw, float pitch, float roll);
    void I2C_Init();
    void Accel_Init();
    void Read_Accel();
    void Magn_Init();
    void Read_Magn();
    void Gyro_Init();
    void Read_Gyro();
    void read_sensors();
    void reset_sensor_fusion();
    void compensate_sensor_errors();
    void check_reset_calibration_session();
    char readChar();
    void setup();
    void loop();

    float getPich() { return this->pitch; }
    float getRoll() { return this->roll; }
    float getYaw() { return this->yaw; }
};
#endif
```

Métodos de procesamiento de los datos de la IMU bajo el algoritmo DCM, con verificación adicional contra divisiones sobre cero o números fuera de rango flotante:

```
#include "imu_lib.h"
#include <cmath>
#include <cfloat>
#include <unistd.h>

/* This file is part of the Razor AHRS Firmware */

IMU::IMU(int module)
{
  this->module = module;
  this->output_mode = OUTPUT__MODE_ANGLES;
  this->output_format = OUTPUT__FORMAT_TEXT;
  this->output_errors = false;
  this->gyro_num_samples = 0;
  this->curr_calibration_sensor = 0;
  this->reset_calibration_session_flag = true;
  this->num_accel_errors = 0;
  this->num_magn_errors = 0;
  this->num_gyro_errors = 0;

  this->I2C_Init();

}

void IMU::Compass_Heading()
{
  float mag_x;
  float mag_y;
  float cos_roll;
  float sin_roll;
  float cos_pitch;
  float sin_pitch;

  cos_roll = cos(roll);
  sin_roll = sin(roll);
  cos_pitch = cos(pitch);
  sin_pitch = sin(pitch);

  // Tilt compensated magnetic field X
  mag_x = magnetom[0] * cos_pitch + magnetom[1] * sin_roll * sin_pitch + magnetom[2] *
      cos_roll * sin_pitch;
  // Tilt compensated magnetic field Y
  mag_y = magnetom[1] * cos_roll - magnetom[2] * sin_roll;
  // Magnetic Heading
  float MAG_Heading_TMP = atan2(-mag_y, mag_x);

  if (std::isfinite(MAG_Heading_TMP))
    MAG_Heading = MAG_Heading_TMP;
}

/* This file is part of the Razor AHRS Firmware */
```

```
// DCM algorithm

/**************************************************/
void IMU::Normalize(void)
{
  float error=0;
  float temporary[3][3];
  float renorm=0;

  error= -Vector_Dot_Product(&DCM_Matrix[0][0],&DCM_Matrix[1][0])*.5; //eq.19

  Vector_Scale(&temporary[0][0], &DCM_Matrix[1][0], error); //eq.19
  Vector_Scale(&temporary[1][0], &DCM_Matrix[0][0], error); //eq.19

  Vector_Add(&temporary[0][0], &temporary[0][0], &DCM_Matrix[0][0]);//eq.19
  Vector_Add(&temporary[1][0], &temporary[1][0], &DCM_Matrix[1][0]);//eq.19

  Vector_Cross_Product(&temporary[2][0],&temporary[0][0],&temporary[1][0]); // c= a x b
       //eq.20

  renorm= .5 *(3 - Vector_Dot_Product(&temporary[0][0],&temporary[0][0])); //eq.21
  Vector_Scale(&DCM_Matrix[0][0], &temporary[0][0], renorm);

  renorm= .5 *(3 - Vector_Dot_Product(&temporary[1][0],&temporary[1][0])); //eq.21
  Vector_Scale(&DCM_Matrix[1][0], &temporary[1][0], renorm);

  renorm= .5 *(3 - Vector_Dot_Product(&temporary[2][0],&temporary[2][0])); //eq.21
  Vector_Scale(&DCM_Matrix[2][0], &temporary[2][0], renorm);
}

/**************************************************/
void IMU::Drift_correction(void)
{
  float mag_heading_x;
  float mag_heading_y;
  float errorCourse;
  //Compensation the Roll, Pitch and Yaw drift.
  static float Scaled_Omega_P[3];
  static float Scaled_Omega_I[3];
  float Accel_magnitude;
  float Accel_weight;


  //*****Roll and Pitch***************

  // Calculate the magnitude of the accelerometer vector
  Accel_magnitude = sqrt(Accel_Vector[0]*Accel_Vector[0] + Accel_Vector[1]*Accel_Vector
       [1] + Accel_Vector[2]*Accel_Vector[2]);
  Accel_magnitude = Accel_magnitude / GRAVITY; // Scale to gravity.
  // Dynamic weighting of accelerometer info (reliability filter)
  // Weight for accelerometer info (<0.5G = 0.0, 1G = 1.0 , >1.5G = 0.0)
  Accel_weight = this->constrain(1 - 2*std::abs(1.0 - Accel_magnitude),0,1);  //

  Vector_Cross_Product(&errorRollPitch[0],&Accel_Vector[0],&DCM_Matrix[2][0]); //adjust
       the ground of reference
  Vector_Scale(&Omega_P[0],&errorRollPitch[0],Kp_ROLLPITCH*Accel_weight);

  Vector_Scale(&Scaled_Omega_I[0],&errorRollPitch[0],Ki_ROLLPITCH*Accel_weight);
  Vector_Add(Omega_I,Omega_I,Scaled_Omega_I);

  //*****YAW***************
  // We make the gyro YAW drift correction based on compass magnetic heading

  mag_heading_x = cos(MAG_Heading);
  mag_heading_y = sin(MAG_Heading);
  errorCourse=(DCM_Matrix[0][0]*mag_heading_y) - (DCM_Matrix[1][0]*mag_heading_x);  //
       Calculating YAW error
  Vector_Scale(errorYaw,&DCM_Matrix[2][0],errorCourse); //Applys the yaw correction to
       the XYZ rotation of the aircraft, depeding the position.

  Vector_Scale(&Scaled_Omega_P[0],&errorYaw[0],Kp_YAW);//.01proportional of YAW.
  Vector_Add(Omega_P,Omega_P,Scaled_Omega_P);//Adding  Proportional.

  Vector_Scale(&Scaled_Omega_I[0],&errorYaw[0],Ki_YAW);//.00001Integrator
  Vector_Add(Omega_I,Omega_I,Scaled_Omega_I);//adding integrator to the Omega_I
}

float IMU::constrain(float X, float A, float B) {
  if(X>=A && X<=B){
    return X;
  }else if(X<A){
```

```
    return A;
  }else{
    return B;
  }
}

void IMU::Matrix_update(void)
{
  Gyro_Vector[0]=1.0 * GYRO_SCALED_RAD(gyro[0]); //gyro x roll
  Gyro_Vector[1]=1.0 * GYRO_SCALED_RAD(gyro[1]); //gyro y pitch
  Gyro_Vector[2]=1.0 * GYRO_SCALED_RAD(gyro[2]); //gyro z yaw

  Accel_Vector[0]=accel[0];
  Accel_Vector[1]=accel[1];
  Accel_Vector[2]=accel[2];

  Vector_Add(&Omega[0], &Gyro_Vector[0], &Omega_I[0]);  //adding proportional term
  Vector_Add(&Omega_Vector[0], &Omega[0], &Omega_P[0]); //adding Integrator term

#if DEBUG__NO_DRIFT_CORRECTION == true // Do not use drift correction
  Update_Matrix[0][0]=0;
  Update_Matrix[0][1]=-G_Dt*Gyro_Vector[2];//-z
  Update_Matrix[0][2]=G_Dt*Gyro_Vector[1];//y
  Update_Matrix[1][0]=G_Dt*Gyro_Vector[2];//z
  Update_Matrix[1][1]=0;
  Update_Matrix[1][2]=-G_Dt*Gyro_Vector[0];
  Update_Matrix[2][0]=-G_Dt*Gyro_Vector[1];
  Update_Matrix[2][1]=G_Dt*Gyro_Vector[0];
  Update_Matrix[2][2]=0;
#else //Use drift correction
  Update_Matrix[0][0]=0;
  Update_Matrix[0][1]=-G_Dt*Omega_Vector[2];//-z
  Update_Matrix[0][2]=G_Dt*Omega_Vector[1];//y
  Update_Matrix[1][0]=G_Dt*Omega_Vector[2];//z
  Update_Matrix[1][1]=0;
  Update_Matrix[1][2]=-G_Dt*Omega_Vector[0];//-x
  Update_Matrix[2][0]=-G_Dt*Omega_Vector[1];//-y
  Update_Matrix[2][1]=G_Dt*Omega_Vector[0];//x
  Update_Matrix[2][2]=0;
#endif

  Matrix_Multiply(DCM_Matrix,Update_Matrix,Temporary_Matrix); //a*b=c

  for(short x=0; x<3; x++) //Matrix Addition (update)
  {
    for(short y=0; y<3; y++)
    {
      DCM_Matrix[x][y]+=Temporary_Matrix[x][y];
    }
  }
}

void IMU::Euler_angles(void) {
  float pitch_TMP = -asin(DCM_Matrix[2][0]);
  float roll_TMP = atan2(DCM_Matrix[2][1],DCM_Matrix[2][2]);
  float yaw_TMP = atan2(DCM_Matrix[1][0],DCM_Matrix[0][0]);
  //
  pitch = pitch_TMP;
  if (std::isfinite(roll_TMP))
    roll = roll_TMP;
  if (std::isfinite(yaw_TMP))
    yaw = yaw_TMP;
}

/* This file is part of the Razor AHRS Firmware */

// Computes the dot product of two vectors
float IMU::Vector_Dot_Product(const float v1[3], const float v2[3])
{
  float result = 0;

  for(short c = 0; c < 3; c++)
  {
    result += v1[c] * v2[c];
  }

  return result;
}

// Computes the cross product of two vectors
// out has to different from v1 and v2 (no in-place)!
void IMU::Vector_Cross_Product(float out[3], const float v1[3], const float v2[3])
```

```
{
  out[0] = (v1[1] * v2[2]) - (v1[2] * v2[1]);
  out[1] = (v1[2] * v2[0]) - (v1[0] * v2[2]);
  out[2] = (v1[0] * v2[1]) - (v1[1] * v2[0]);
}

// Multiply the vector by a scalar
void IMU::Vector_Scale(float out[3], const float v[3], float scale)
{
  for(short c = 0; c < 3; c++)
  {
    out[c] = v[c] * scale;
  }
}

// Adds two vectors
void IMU::Vector_Add(float out[3], const float v1[3], const float v2[3])
{
  for(short c = 0; c < 3; c++)
  {
    out[c] = v1[c] + v2[c];
  }
}

// Multiply two 3x3 matrices: out = a * b
// out has to different from a and b (no in-place)!
void IMU::Matrix_Multiply(const float a[3][3], const float b[3][3], float out[3][3])
{
  for(short x = 0; x < 3; x++)  // rows
  {
    for(short y = 0; y < 3; y++)  // columns
    {
      out[x][y] = a[x][0] * b[0][y] + a[x][1] * b[1][y] + a[x][2] * b[2][y];
    }
  }
}

// Multiply 3x3 matrix with vector: out = a * b
// out has to different from b (no in-place)!
void IMU::Matrix_Vector_Multiply(const float a[3][3], const float b[3], float out[3])
{
  for(short x = 0; x < 3; x++)
  {
    out[x] = a[x][0] * b[0] + a[x][1] * b[1] + a[x][2] * b[2];
  }
}

// Init rotation matrix using euler angles
void IMU::init_rotation_matrix(float m[3][3], float yaw, float pitch, float roll)
{
  float c1 = cos(roll);
  float s1 = sin(roll);
  float c2 = cos(pitch);
  float s2 = sin(pitch);
  float c3 = cos(yaw);
  float s3 = sin(yaw);

  // Euler angles, right-handed, intrinsic, XYZ convention
  // (which means: rotate around body axes Z, Y', X'')
  m[0][0] = c2 * c3;
  m[0][1] = c3 * s1 * s2 - c1 * s3;
  m[0][2] = s1 * s3 + c1 * c3 * s2;

  m[1][0] = c2 * s3;
  m[1][1] = c1 * c3 + s1 * s2 * s3;
  m[1][2] = c1 * s2 * s3 - c3 * s1;

  m[2][0] = -s2;
  m[2][1] = c2 * s1;
  m[2][2] = c1 * c2;
}

/* This file is part of the Razor AHRS Firmware */


void IMU::I2C_Init()
{
  this->Acelerometro = new Adxl345(module);
  this->Gyrometro = new Gyro(module);
  this->Magnetometro= new MAG(module);
}
```

```
// Reads x, y and z accelerometer registers

void IMU::Read_Accel()
{
  this->Acelerometro->readFullAcel();
  accel[0] = this->Acelerometro->getAcelX();  // X axis (internal sensor y axis)
  accel[1] = this->Acelerometro->getAcelY();  // Y axis (internal sensor x axis)
  accel[2] = this->Acelerometro->getAcelZ();  // Z axis (internal sensor z axis)

}

void IMU::Read_Gyro(){
  this->Gyrometro->readFullGyro();
  gyro[0] = this->Gyrometro->getGyroX() * 1.05;  // X axis (internal sensor y axis)
  gyro[1] = this->Gyrometro->getGyroY() * 1.05;  // Y axis (internal sensor x axis)
  gyro[2] = this->Gyrometro->getGyroZ() * 1.05;  // Z axis (internal sensor z axis)

}
void IMU::Read_Magn(){
  this->Magnetometro->readfullState();
  magnetom[0]=this->Magnetometro->getMagX();
  magnetom[1]=this->Magnetometro->getMagY();
  magnetom[2]=this->Magnetometro->getMagZ();
}

void IMU::read_sensors() {

  Read_Gyro(); // Read gyroscope
  Read_Accel(); // Read accelerometer
  Read_Magn(); // Read magnetometer
}

// Read every sensor and record a time stamp
// Init DCM with unfiltered orientation
// TODO re-init global vars?
void IMU::reset_sensor_fusion() {
  float temp1[3];
  float temp2[3];
  float xAxis[] = {1.0f, 0.0f, 0.0f};

  read_sensors();
  //timestamp = millis();

  // GET PITCH
  // Using y-z-plane-component/x-component of gravity vector
  float pitch_TMP = -atan2(accel[0], sqrt(accel[1] * accel[1] + accel[2] * accel[2]));
  //
  if (std::isfinite(pitch_TMP))
    pitch = pitch_TMP;

  // GET ROLL
  // Compensate pitch of gravity vector
  Vector_Cross_Product(temp1, accel, xAxis);
  Vector_Cross_Product(temp2, xAxis, temp1);
  // Normally using x-z-plane-component/y-component of compensated gravity vector
  // roll = atan2(temp2[1], sqrt(temp2[0] * temp2[0] + temp2[2] * temp2[2]));
  // Since we compensated for pitch, x-z-plane-component equals z-component:
  float roll_TMP = atan2(temp2[1], temp2[2]);
  //
  if (std::isfinite(roll_TMP))
    roll = roll_TMP;

  // GET YAW
  Compass_Heading();
  yaw = MAG_Heading;

  // Init rotation matrix
  init_rotation_matrix(DCM_Matrix, yaw, pitch, roll);
}

// Apply calibration to raw sensor readings
void IMU::compensate_sensor_errors() {
    // Compensate accelerometer error
    accel[0] = (accel[0] - ACCEL_X_OFFSET) * ACCEL_X_SCALE;
    accel[1] = (accel[1] - ACCEL_Y_OFFSET) * ACCEL_Y_SCALE;
    accel[2] = (accel[2] - ACCEL_Z_OFFSET) * ACCEL_Z_SCALE;

    // Compensate magnetometer error
#if CALIBRATION__MAGN_USE_EXTENDED == true
    for (short i = 0; i < 3; i++)
      magnetom_tmp[i] = magnetom[i] - magn_ellipsoid_center[i];
    Matrix_Vector_Multiply(magn_ellipsoid_transform, magnetom_tmp, magnetom);
```

```
#else
    magnetom[0] = (magnetom[0] - MAGN_X_OFFSET) * MAGN_X_SCALE;
    magnetom[1] = (magnetom[1] - MAGN_Y_OFFSET) * MAGN_Y_SCALE;
    magnetom[2] = (magnetom[2] - MAGN_Z_OFFSET) * MAGN_Z_SCALE;
#endif

    // Compensate gyroscope error
    gyro[0] -= GYRO_AVERAGE_OFFSET_X;
    gyro[1] -= GYRO_AVERAGE_OFFSET_Y;
    gyro[2] -= GYRO_AVERAGE_OFFSET_Z;
}

// Reset calibration session if reset_calibration_session_flag is set
void IMU::check_reset_calibration_session()
{
  // Raw sensor values have to be read already, but no error compensation applied

  // Reset this calibration session?
  if (!reset_calibration_session_flag) return;

  // Reset acc and mag calibration variables
  for (short i = 0; i < 3; i++) {
    accel_min[i] = accel_max[i] = accel[i];
    magnetom_min[i] = magnetom_max[i] = magnetom[i];
  }

  // Reset gyro calibration variables
  gyro_num_samples = 0;  // Reset gyro calibration averaging
  gyro_average[0] = gyro_average[1] = gyro_average[2] = 0.0f;

  reset_calibration_session_flag = false;
}

void IMU::setup()
{
  I2C_Init();
  usleep(20000);  // Give sensors enough time to collect data
  reset_sensor_fusion();
}


// Main loop
void IMU::loop()
{
  G_Dt = .020;
  read_sensors();

  // Apply sensor calibration
  compensate_sensor_errors();

  // Run DCM algorithm
  Compass_Heading(); // Calculate magnetic heading
  Matrix_update();
  Normalize();
  Drift_correction();
  Euler_angles();

}
```

C.2.11 Generación de señales PWM en el microcontrolador

Creado para un microncontrolador Stellaris usando Energia, plataforma similar a Arduino. Sólo se reciben las cadenas de control de los motores, con una suma de verificación contra errores en la trama.

```
#include "Servo.h"

unsigned long milisLast = 0;
unsigned long milisLastMsg = 0;
unsigned long millisTock = 0;
unsigned long timescont = 0;
bool timedOut = false;

Servo servo_13;
```

```
Servo servo_14;

volatile int s13_out = 0;  // m1
volatile int s14_out = 0;  // m2

void setup() {

  Serial.begin(115200);
  Serial5.begin(115200);

  servo_13.attach(13);
  servo_14.attach(14);

  servo_13.writeMicroseconds( 1500 ); // m1
  servo_14.writeMicroseconds( 1500 ); // m2

  timedOut = true;

  pinMode(RED_LED, OUTPUT);
  pinMode(GREEN_LED, OUTPUT);

  pinMode(13, OUTPUT);
  pinMode(14, OUTPUT);

  millisTock = millis();
}

uint16_t word1 = 0;
int debug = 0;
int lec1 = 0;
int lec2 = 0;

void commloop() {

  if (Serial.available()) {
    debug = Serial.read();
  }

  // PROT FF AL HL HL CHKSUM = 10 bytes
  if (Serial5.available() >= 10){

    // 0x7530 = 30000
    lec1 = Serial5.read();
    word1 = (word1 << 8) | lec1;
    if (word1 != 0x7530) {
      while(Serial5.available()) {
        lec1 = Serial5.read();
        word1 = (word1 << 8) | lec1;
        if (word1 == 0x7530) {
          if (Serial5.available() < 8) {
           delay(10); // espera por la trama de ocmunicaciones completa
           if (Serial5.available() < 8) { // no se consiguio la trama
             return;  // stop
           }
          }
          break;
        }
      }
    }

    // read alive status
    lec1 = Serial5.read();
    lec2 = Serial5.read();
    uint16_t aldata = (lec1 << 8) | lec2;
    // read the servos data, in microseconds
    lec1 = Serial5.read();
    lec2 = Serial5.read();
    int16_t s1data = (lec1 << 8) | lec2;
    lec1 = Serial5.read();
    lec2 = Serial5.read();
    int16_t s2data = (lec1 << 8) | lec2;
    // verify the checksum
    lec1 = Serial5.read();
    lec2 = Serial5.read();
    uint16_t checksumH = lec1;
    uint16_t checksumL = lec2;
    //
    uint16_t localchecksumH = (((long)aldata + (long)s1data + (long)s2data) >> 8) &
        255;
    uint16_t localchecksumL = (((long)aldata + (long)s1data + (long)s2data) >> 0) &
        255;
    //
```

```
    uint16_t checksum = (checksumH << 8) | checksumL;
    uint16_t localchecksum = (localchecksumH << 8) | localchecksumL;

    if (debug == '1') {
     Serial.print("␣w␣");
     Serial.print((word1));
     Serial.print("␣a␣");
     Serial.print((aldata));
     Serial.print("␣s1␣");
     Serial.print((s1data));
     Serial.print("␣s2␣");
     Serial.print((s2data));
     Serial.print("␣c␣");
     Serial.print(checksum);;
     Serial.print("␣l␣");
     Serial.print(localchecksum);
     Serial.print("␣b␣");
     Serial.print(Serial5.available());
     Serial.println("");
    }

    // TODO add more aldata rules
    if (checksum == localchecksum) {
      // show status
      digitalWrite(GREEN_LED, HIGH);
      milisLastMsg = millis();
      timedOut = false;

      if (aldata == 1) {
        digitalWrite(RED_LED, LOW);
        s13_out = s1data;
        s14_out = s2data;
        // record msg time
        timescont++;
      }
      if (aldata == 0) {
        digitalWrite(RED_LED, HIGH);
        s13_out = 0;
        s14_out = 0;
      }
    }
  }
}

void loop() {

  commloop();

  if (millisTock <= millis()) {

    millisTock += 50;

    if ((millis() - milisLastMsg) > 500) {
      // shut down all
      digitalWrite(RED_LED, HIGH);
      digitalWrite(GREEN_LED, LOW);
      s13_out = 0;  // m1
      s14_out = 0;  // m2
      timedOut = true;
      timescont = 0;
    }

    servo_13.writeMicroseconds( s13_out / 20 + 1500); // m1 // 100 points would be a 1
        percent, 5 of 500 us, min of 0.2 percent
    servo_14.writeMicroseconds( s14_out / 20 + 1500); // m2 // 100 points would be a 1
        percent, 5 of 500 us, min of 0.2 percent
  }
}
```

Bibliografía

[1] Crossley, Vincent A. *A Literature Review on the Design of Spherical Rolling Robots*. Carnegie Mellon University.

[2] Chase, R., Pandya, A. *A Review of Active Mechanical Driving Principles of Spherical Robots*. Wayne State University, 2012.

[3] Antol, J., Calhoun, P., Flick, J., Hajos, G., Kolacinski, R., Minton, D., Owens, R., Parker, J. *Low Cost Mars Surface Exploration: The Mars Tumbleweed*. NASA Langley Research Center, 2003.

[4] Unkubo, T., Osawa, M., Maekawa, S. *Development of a Spherical Rolling Robot Equipped with a Gyro*. Proceedings of the 2012 International Conference on Mechatronics and Automation, 2012.

[5] Shapiro, Joel A. *Classical Mechanics*. 2003

[6] Tong, D. *Classical Dynamics*. University of Cambridge Part II Mathematical Tripos, 2005

[7] Hoifodt, H. *Dynamic Modeling and Simulation of Robot Manipulators, the Newton-Euler Formulation*. Norwegian University of Science and Technology, 2011

[8] Bicchi, A., Balluchi, A., Prattichizzo, D., Gorelli, A. *Introducing the "SPHERICLE": and Experimental Testbed for Research and Teaching in Nonholonomy*. Proceedings of the 1997 IEEE International Conference on Robotics and Automation, 1997.

[9] Brown, H., Xu, Y. *A Single-wheel, Gyroscopically Stabilized Robot*. Proceedings of the 1996 IEEE International Conference on Robotics and Automation, 1996.

[10] Halme, A., Suomela, J., Schönberg, T., Wang, Y. *A Spherical Mobile Microrobot for Scientific Applications*. Helsinki University of Technology, 1996.

[11] Han, B., Chen, Y., Li, H., Yang, L. *Discrete Model Reference Adaptive Control for Gimbal Servosystem of Control Moment Gyro with Harmonic Drive*. Mathematical Problems in Engineering, 2013.

[12] Hibbeler, R. C. *Engineering Mechanics: Dynamics*. 13a. edición. Prentice Hall, 2012.

[13] Javadi, A., Mojabi, P. *Introducing August: a Novel Strategy for an Omnidirectional Spherical Rolling Robot*. Proceedings of the 2002 IEEE International Conference on Robotics and Automation, 2002.

[14] Joshi, V., Banavar, R., Hippalgaonkar, R. *Design, modeling and controllability of a Spherical Mobile Robot*. 13th National Conference on Mechanisms and Machines, India, 2007.

[15] J. Lux *Alternative Way of Shifting Mass to Move a Spherical Robot*. NASA Jet Propulsion Laboratory, 2005.

[16] Mukherjee, R., Minor, M., Pukrushpan, J., *Simple Motion Planning Strategies for Spherobot: a Spherical Mobile Robot*. Proceedings of the 38th Conference on Decision and Control, 1999,

[17] Rotundus. *GroundBot*. Disponible en línea en: `http://www.rotundus.se/` (consultada el 20 de marzo de 2015).

[18] Schroll, G. *Design of a Spherical Vehicle with Flywheel Momentum Storage for High Torque Capabilities*. Massachusetts Institute of Technology, 2008.

[19] Schroll, G. *Dynamic Model of a Spherical Robot from First Principles*. Colorado State University, 2010.

[20] Wait, K., Jackson, P., Smoot, L. *Self Locomotion of a Spherical Rolling Robot Using a Novel Deformable Pneumatic Method*. Proceedings of the 2010 IEEE International Conference on Robotics and Automation, 2010.

[21] Yilkorpi, T., Suomela, J. *Ball-shaped Robots*. Helsinki University of Technology.
En: Zhang, H. *Climbing and Walking Robots: Towards New Applications*. I-Tech Education and Publishing, 2007, pp. 235-256.

[22] Yue, M., Deng, Z., Yu, X., Yu, W. *Introducing HIT Spherical Robot: Dynamic Modeling and Analysis Based on Decoupled Subsystem.* Proceedings of 2006 IEEE International Conference on Robotics and Biomimetics, 2006.