UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

SISTEMA SERVOMECÁNICO PARA COMPENSACIÓN DE PARES GRAVITACIONALES EN UN SIMULADOR SATELITAL.

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO ELÉCTRICO - ELECTRÓNICO

PRESENTA:

DURÁN FLORENTINO LAURA ARELI

DIRECTOR DE TESIS:

Dr. en I. JORGE PRADO MOLINA

CD. UNIVERSITARIA MÉXICO D.F. JUNIO 2012

Índice

Capítulo

1

Introducción

En el desarrollo de cualquier proyecto de ingeniería espacial, siempre son necesarias las pruebas previas a la implementación en el ambiente real de operación de todo o parte del sistema realizado. Para las pruebas de componentes espaciales esto representa un gran desafío, dado que los costos de ensayos en el espacio son muy altos, además de inviables en muchos casos. Por esta razón, es necesario contar con un sistema capaz de emular las condiciones ambientales del espacio y que sea adecuado para probar de manera experimental, bajo las condiciones controladas de un laboratorio, el desempeño de componentes en desarrollo, como sensores, actuadores y algoritmos que serán incluidos posteriormente en equipos espaciales.

En el Laboratorio de Percepción Remota Alternativa y Tecnología Avanzada del Instituto de Geografía de la UNAM, se han realizado diversos proyectos con el fin de generar la infraestructura para llevar a cabo los procesos de detección de orientación y de control de estabilización de satélites pequeños (con masa menor a 100 kg). En particular se ha diseñado y construido un simulador satelital, llamado SIMUSAT, en diferentes versiones, conforme ha evolucionado con el tiempo. Este es un sistema conformado por una plataforma circular fabricada a base de fibra de carbono y resina epóxica, montada sobre un balero de aire esférico, el cual permite entre otras cosas, llevar a cabo ensayos experimentales de algunos de los conceptos fundamentales del control de orientación de naves espaciales en un medio con fricción prácticamente nula, condición similar a la que se presenta en el espacio exterior. La última versión de este tipo de simuladores es el SIMUSAT_3.0, que contempla un cambio fundamental en la reducción de sus componentes y en la forma de su funcionamiento, esto se hizo con el fin de lograr una plataforma que tenga una masa menor a 10 kg, esta condición es necesaria para pruebas en nanosatélites (Escobedo L. 2012).

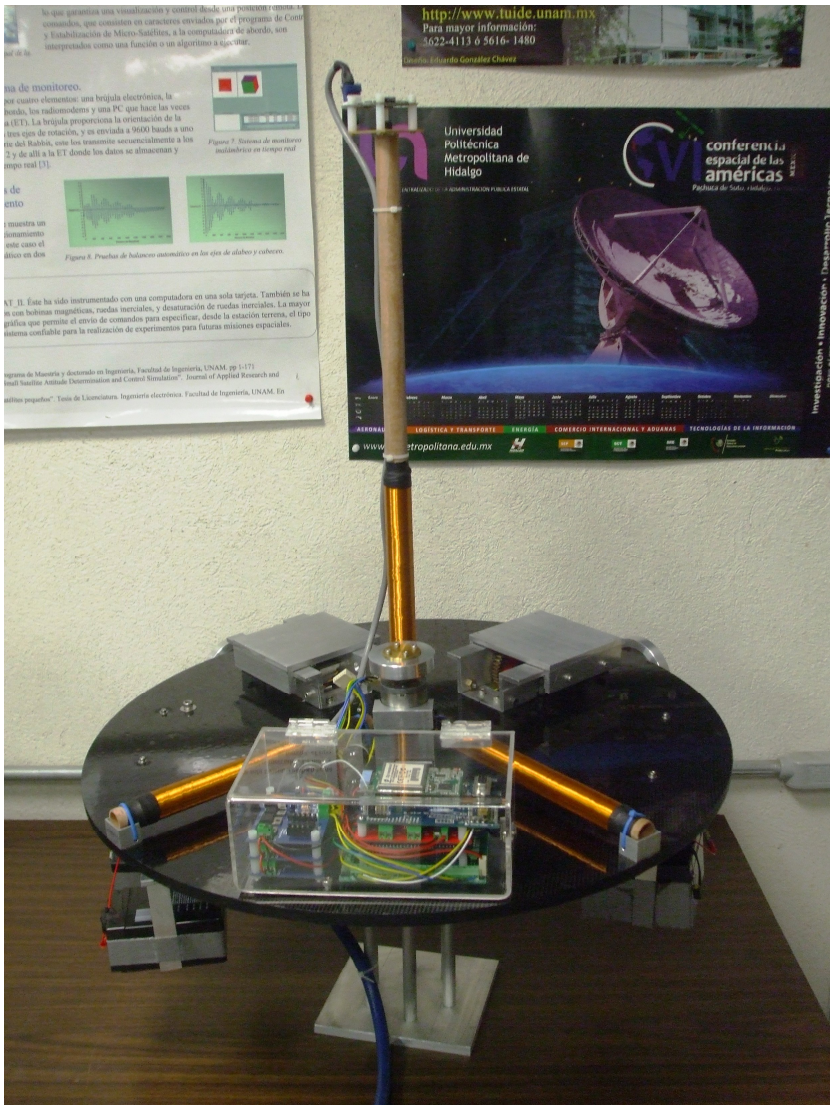


Figura 1.1. Sistema de simulación satelital SIMUSAT_3.0, especialmente adaptado para pruebas de sistemas de control de orientación de nanosatélites.

El movimiento libre sin fricción de la plataforma genera ciertos efectos no deseados al realizar las pruebas en tierra, bajo el efecto de fuerzas que no existirían de realizarse en el espacio exterior; uno de los más importantes es el causado por la atracción gravitacional terrestre. El sistema oscila intermitentemente mientras estas fuerzas externas y la inercia del movimiento impiden la estabilidad de la plataforma. Una parte operativa se ve limitada al ocurrir el fenómeno de oscilación antes mencionado, el cual no permite apreciar el movimiento que acontecería en el espacio exterior. La falta de fricción es una de las condiciones más importantes a emular desde el punto de vista de sistemas dinámicos, por tanto, es deseable poder contar con un algoritmo que minimice los efectos gravitacionales indeseables a los que se que se ve sometida la plataforma de simulación.

En este trabajo se propuso un método para minimizar el efecto que ejercen los pares gravitacionales externos sobre el simulador, mediante la implementación de un sistema de seguimiento servomecánico que compensa los movimientos de los actuadores de la plataforma, emulando el comportamiento de la plataforma, cual si ésta se encontrara en un ambiente de gravedad reducida.

1.1. Sistema de simulación.

Consiste de una plataforma circular suspendida sobre un balero de aire esférico, que es dónde se genera un medio sin fricción (ver figura 1.4). Este es un medio muy efectivo de integración de todos los componentes de los sistemas de detección de orientación y de control de estabilización. Además de incluir a la computadora de abordo y baterías, también cuenta con un sistema de transmisión de datos de manera inalámbrica, ya que el uso de cables rompería el delgado colchón de aire que sustenta la plataforma.

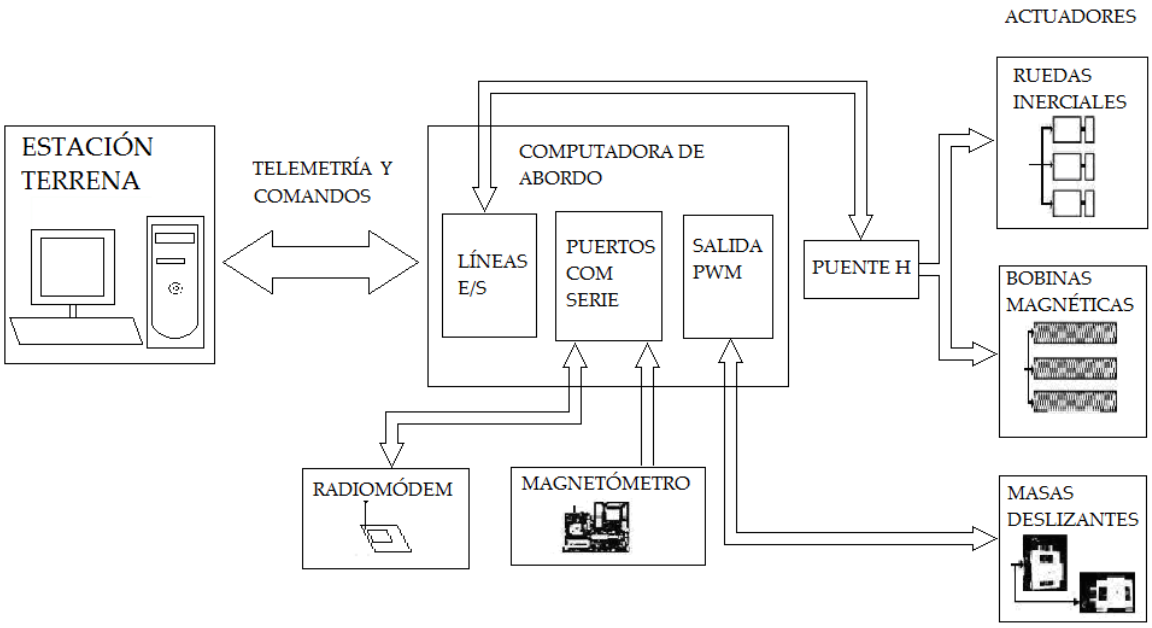


Figura 1.2 Diagrama a bloques del sistema de simulación.

Desde la estación terrena enviamos comandos a la plataforma de simulación y ésta nos regresa la telemetría, que en este caso son las señales de la brújula electrónica, es decir, las tres componentes de orientación (Roll, Pitch y Yaw). En un caso diferente, se pueden incluir otros sensores y por ejemplo, recibir temperaturas, velocidades angulares, señales de orientación de sensores de sol, de Tierra o de estrellas, etc. La gama de señales de telemetría cambia según las necesidades que se tengan de monitorear aquellas variables más importantes de la misión. En la

figura 1.3 se muestra del lado izquierdo, la plataforma de simulación y en el derecho, una PC de escritorio que hace las veces de estación terrena. El módem de comunicación bidireccional inalámbrica se encuentra sobre el CPU. Existe, por supuesto, otro módem en la plataforma de simulación, para conformar un sistema de comunicaciones bidireccional.

El sistema completo está integrado por: a) una plataforma circular suspendida sobre un balero de aire esférico, donde es generado un medio sin fricción, b) tres ruedas inerciales que constituyen el grupo actuadores para control de orientación de la plataforma, c) tres bobinas magnéticas, localizadas en ejes mutuamente perpendiculares, que de-saturan las ruedas inerciales en su función de momentum diferente a cero y proporcionan un sistema de control de respaldo, d) una compás electrónico, integrado por dos inclinómetros y un magnetómetro para determinar cualquier desviación angular en los tres ejes de la plataforma, y e) dos masas deslizantes para el balanceo automático. También cuenta con un sistema de monitoreo inalámbrico que transmite la orientación de los tres ejes, durante las pruebas.

Figura 1.3. Sistema de simulación completo. A la izquierda se encuentra la plataforma y a la derecha la estación terrena.

Los datos de orientación son desplegados en tiempo real, sin embargo, la parte más importante es el almacenamiento ya que esto permite su graficación, para sí evaluar el comportamiento de los dispositivos bajo prueba, en particular, sensores, actuadores y algoritmos de los sistemas de control de orientación (Méndez F. y Huante D., 2009).

1.1.1. Plataforma de simulación.

No es posible emular al mismo tiempo todas las condiciones ambientales del espacio exterior como son: vacío, microgravedad, falta de fricción, etc. Para poder llevar a cabo el desarrollo y las pruebas de funcionamiento de los sistemas de control de orientación de satélites de una manera realista y objetiva, es necesario contar con diferentes equipos para este propósito. La principal finalidad de este sistema de simulación es emular la falta de fricción, muy importante desde el punto de vista dinámico.

Se cuenta con un simulador que permite tener movimiento angular en tres ejes y un medio con fricción prácticamente nula. Consta de una plataforma móvil donde se colocan los componentes del sistema, suspendida sobre un cojinete neumático esférico. Los cojinetes neumáticos son dispositivos auxiliares muy utilizados para el desarrollo de la programación y la instrumentación de sistemas para determinar la orientación de naves espaciales. Aunque existen otras posibilidades de simular la falta de fricción, éstas presentan inconvenientes como, por ejemplo, el bloqueo que ocurre con los sistemas basculantes, además de no presentar fricción despreciable. (Prado Molina, Jorge, 2007)

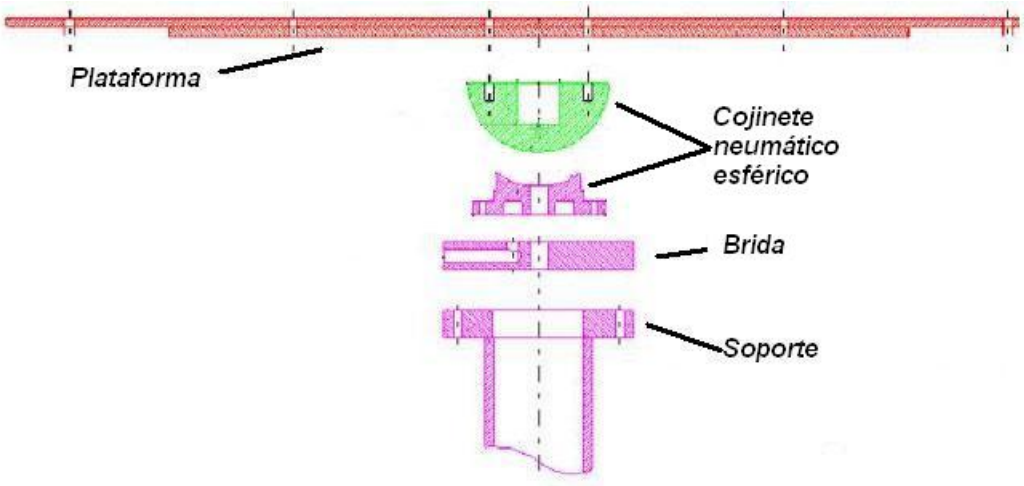


Figura 1.4 Diagrama general de la plataforma de simulación

En resumen, el simulador es totalmente autónomo, cuenta con su propio sistema de alimentación y comunicación vía inalámbrica. No depende de cables para su interacción con la estación terrena ya que cualquier peso, por ligero que sea, afectaría la estabilidad de la plataforma debido a la falta de fricción. Es ideal soporte para los actuadores, sensores, sistema de balanceo automático y demás componentes.

1.1.2. Medio sin fricción.

El cojinete de aire esférico está compuesto por una semiesfera y una copa de aluminio. La unión de estos dos elementos produce un colchón de aire que provee el medio sin fricción. El aire se introduce por la parte baja de la copa y sale por la periferia, es decir, por su parte superior. La semiesfera se sujeta firmemente a la plataforma y se eleva sobre el colchón de aire, generando así el medio sin fricción.

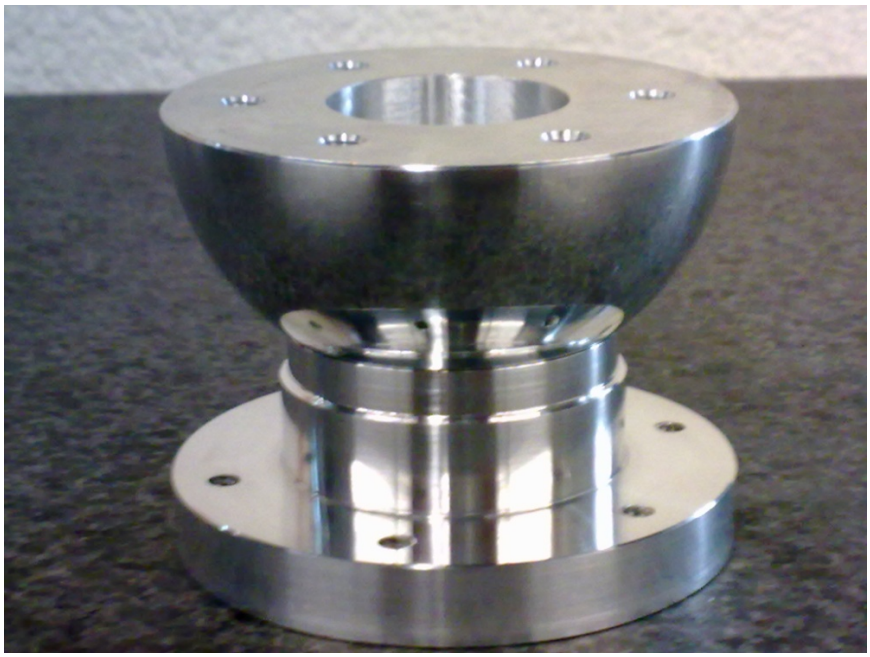


Figura 1.5 Semiesfera y copa de aluminio. El medio sin fricción se crea entre ambos componentes.

Una característica importante de los cojinetes neumáticos, es el hecho de necesitar pequeñas presiones y gastos de aire para soportar una carga dada, aunque es necesario colocar filtros para impedir el paso del agua y el aceite ya que éstos últimos pueden romper fácilmente el delgado colchón de aire. Es necesario mantener en condiciones adecuadas el medio sin fricción, es decir, tener una superficie de contacto limpia y bien pulida entre las partes que conforman el balero de aire. Anteriormente el cojinete neumático era elaborado en bronce; con el objeto

de reducir su masa se optó por rediseñarlo con base en el aluminio 6061-T6 y además reducir su tamaño (Escobedo L. 2012), sin embargo, se tienen problemas con este material, ya que es mucho más dúctil que el bronce y se raya fácilmente cuando el aire a presión es suspendido y ambas piezas entran en contacto.

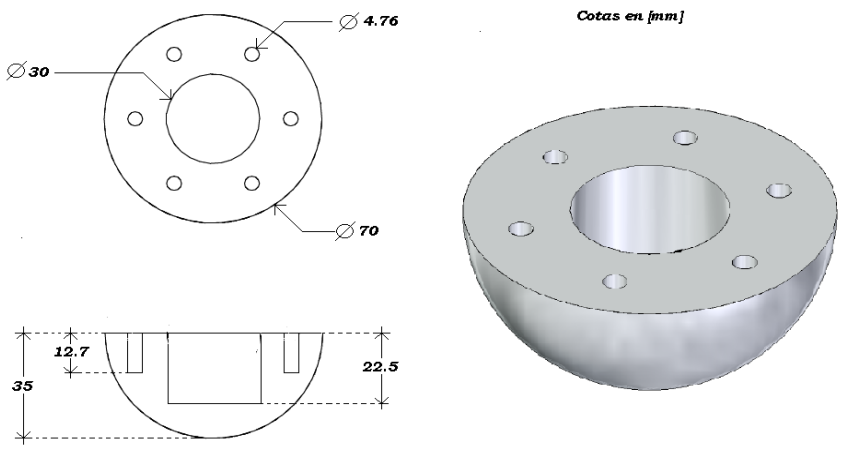


Figura 1.6 Dibujo de la semiesfera de aluminio.

1.1.3. Subsistema de balanceo en dos ejes.

Para realizar las pruebas de orientación y control se requiere que la plataforma de simulación se encuentre balanceada, así se minimizan los efectos causados por los pares gravitacionales y es posible valorar adecuadamente el funcionamiento de los dispositivos bajo ensayo. Depender de un sistema manual de balanceo es poco adecuado, dado que cualquier pequeña alteración cambia notablemente la estabilidad de la plataforma debido al medio sin fricción en que se encuentra.

Se ha implementado un sistema de balanceo automático en dos ejes, compuesto de dos masas deslizantes móviles y totalmente controlables que se ajustan automáticamente para proporcionar un acomodo del centro de masa de la plataforma en el centro geométrico del balero de aire esférico. Los dos inclinómetros del compás electrónico envían las señales con la posición de la plataforma con respecto a la horizontal, y por medio de un algoritmo ejecutado por la computadora de abordo, las masas se deslizan para ajustar el centro de masa (Juarez A., 2001).

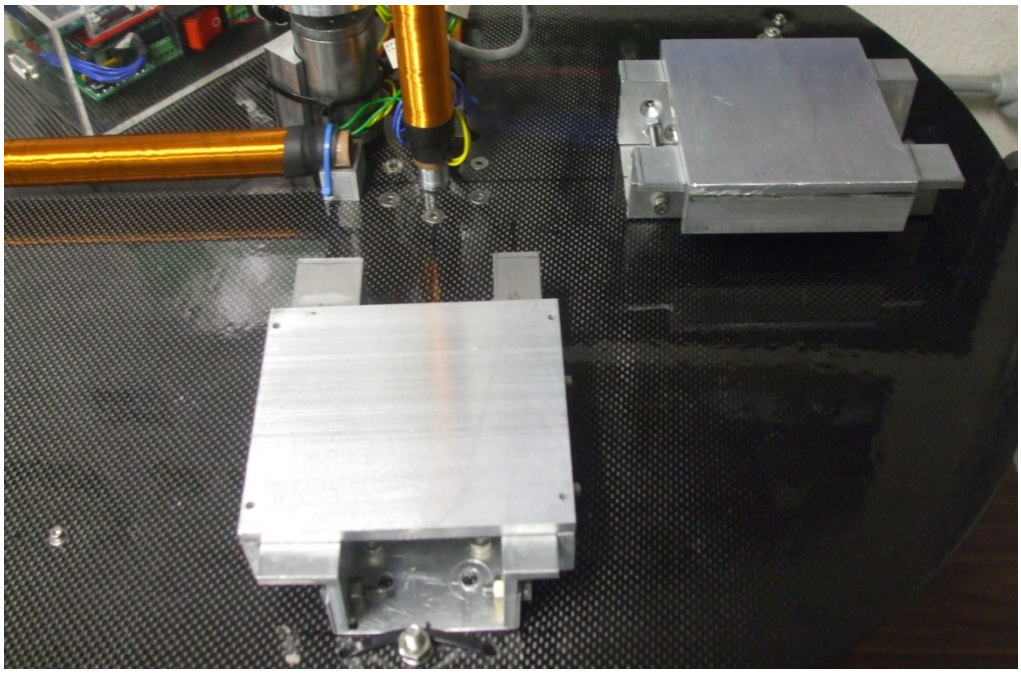


Figura 1.7. Sistema de masas deslizantes.

Los requisitos para el equilibrio estático son simplemente que la suma de todas las fuerzas en el sistema, sean iguales a cero. Esto equivale a equilibrar todos los pares existentes en la plataforma, manteniendo su centro de masa en el centro del cojinete neumático. El balanceo es exclusivamente estático debido a que no se ha considerado el uso de los sistemas estabilizados por giro en satélites. Además este ajuste sólo es necesario para pruebas realizadas en tierra, dónde los efectos de la atracción gravitacional son notables; para las condiciones de microgravedad del espacio exterior este sistema es totalmente inútil.

1.1.4. Actuadores del Subsistema de orientación

Las ruedas inerciales y las bobinas magnéticas constituyen el conjunto de actuadores de la plataforma que se encargan de orientarla de acuerdo al algoritmo programado en la computadora de abordo (microprocesador Rabbit 3000). El simulador es direccionado por las ruedas inerciales a través del intercambio de momentum, entre ellas y la propia plataforma; las bobinas magnéticas reciben un par externo al interactuar con el campo magnético terrestre. Esta condición las hace indispensables en el espacio, ya que permiten imprimir un par neto al satélite, cosa que no ocurre con las ruedas inerciales que solo intercambian momentum entre ellas y el cuerpo del satélite (Méndez F. y Huante D., 2009).

Existen dos modos básicos de funcionamiento de las ruedas inerciales: en modo de reacción y con momentum diferente de cero. En el primer caso tenemos una rueda con velocidad inicial cero y que reacciona girando en un sentido o en otro para compensar una desviación en su eje. En el segundo caso, tenemos una rueda girando a una determinada velocidad; misma que se incrementa o disminuye atendiendo a una perturbación externa. Para el sistema aquí descrito se utilizan las ruedas inerciales en modo de reacción.

Los sistemas de control magnético son relativamente sencillos, no requieren de partes móviles, de sensores sofisticados, ni de consumibles abordo de la nave. Esto hace que los pares magnéticos sean atractivos para aplicaciones espaciales, sin embargo, estos proporcionan pares poco significativos, lo que restringe la cantidad y la rapidez de las maniobras, ya que su operación depende del valor de las componentes vectoriales del campo magnético en el lugar de la órbita donde se realiza la maniobra y de las dimensiones de las bobinas y de la energía disponible a bordo (Juarez A., 2001)

En la figura 1.8 puede observarse la ubicación de ambos conjuntos de actuadores localizados sobre los ejes de referencia de la plataforma de simulación.

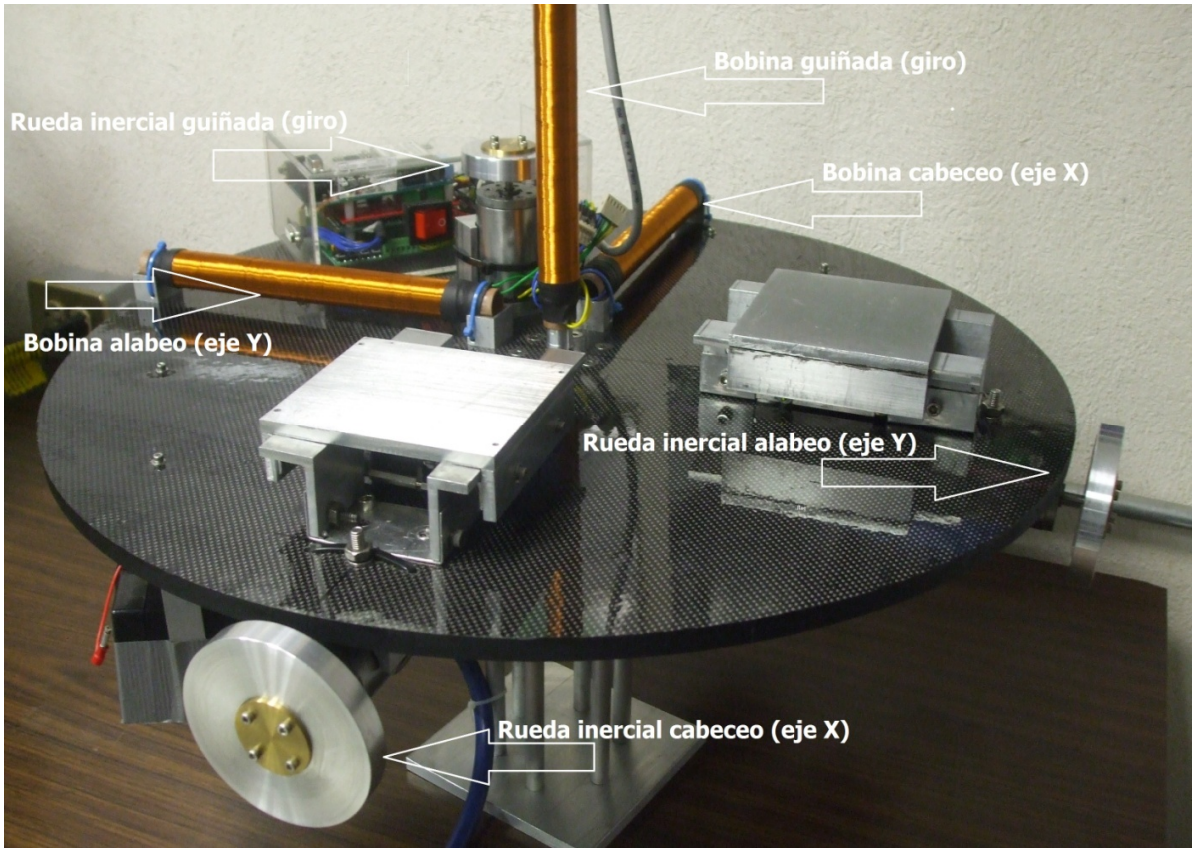


Figura 1.8. Descripción de los elementos actuadores en la plataforma, que son empleados para cambiar su orientación.

1.1.5. Computadora de abordo e interfaces de comunicación.

La computadora de abordo está constituida por un microprocesador Rabbit 3000 con su tarjeta de desarrollo LP3500. Se pueden consultar sus especificaciones técnicas en los manuales correspondientes (Rabbit semiconductor 2007). Una de las características más sobresalientes de este equipo es su capacidad de ser programado utilizando un lenguaje de alto nivel (lenguaje C), que permite un manejo más comprensible y lógico de las estructuras de control, además el tener librerías propias evita el tener que programar rutinas, como aquellas que incluyen el uso del punto flotante. Se tienen dos módem para radiocomunicación inalámbrica con configuración punto a punto full-duplex, es decir, bidireccional; a una velocidad de transmisión de 9600 baudios, seleccionado para poder sincronizar los datos de los subsistemas, en un tiempo considerablemente rápido. La comunicación entre todos los componentes del sistema de simulación, se lleva a cabo siguiendo el protocolo RS-232 que funciona de manera asíncrona.

El monitoreo de la plataforma se hace a través de un programa con una interfaz gráfica que ha sido desarrollada en Visual Basic y cuyo objetivo es permitir la ejecución de las diferentes rutinas en la plataforma, a través del control de los actuadores, suponiendo una estación terrena alejada del simulador. Este sistema de monitoreo se encarga de generar una base de datos que a su vez permite graficar y analizar el comportamiento de la plataforma durante las pruebas de balanceo automático, oscilación simple, pruebas con bobinas magnéticas y ruedas inerciales, etc.

1.2. Influencia de los pares gravitacionales externos en el simulador

El SIMUSAT 3.0 ha sido diseñado para realizar pruebas sin la necesidad de cambiar la ubicación de la plataforma o someterla a algún tipo de aceleración externa; por tanto, además de las fuerzas inducidas por los actuadores, sólo es susceptible a la influencia de la fuerza gravitacional terrestre. Ésta actúa modificando la velocidad de respuesta y reduciendo el efecto de los actuadores sobre la orientación de la plataforma. La mesa oscila intermitentemente al generarse un par opuesto al movimiento impulsado por las ruedas inerciales o las bobinas magnéticas. Durante el balanceo no es muy notorio, dado que el desequilibrio es mínimo y el mismo sistema lo compensa, pero al realizar pruebas de orientación, es causante de una gran oscilación en la plataforma.

Para compensar estas perturbaciones y así poder observar el efecto que tendrían los actuadores en un ambiente de gravedad reducida, se ha elegido sustituir el sistema de masas deslizantes que se venia usando regularmente, por un servomecanismo que pueda realizar ambas funciones, balanceo automático y compensación de pares gravitacionales en la plataforma de simulación, cuando así se desee.

1.3. Sistema de seguimiento servomecánico.

El rediseño del simulador ha permitido, entre varias otras mejoras, la inclusión de un nuevo sistema de masas deslizantes, optimizado para ejecutar el balanceo y para minimizar los efectos de la gravedad sobre la plataforma. En el SIMUSAT_2.1 (Méndez F. y Huante D., 2009), y en todas las versiones anteriores, se utilizaban motores a pasos y un sistema de helicoide para modificar la posición de las masas deslizantes; su movimiento era lineal, constante y con mínimas perturbaciones, aunque muy lento. Al considerar la forma de implementar el sistema compensador de pares gravitacionales en la plataforma, se eligió cambiar este módulo, optimizándolo a fin de que pudiera servir tanto para balanceo como para la compensación de fuerzas gravitacionales, reduciendo además los elementos de mecánicos necesarios para el control. Los motores a pasos fueron cambiados por servomotores y las masas deslizantes modificadas en su mecanismo de acción.

1.3.1. Masas deslizantes y servomotores.

Se han diseñado un nuevo par de masas deslizantes, a fin de incluir en ellas los elementos necesarios para obtener una respuesta de control más rápida y poder así tener el sistema de balanceo y el de compensación en un sólo elemento. El rediseño incluye la sustitución de los motores a pasos y el sistema de helicoide por un servomotor y un sistema engrane-cremallera para cada masa deslizante.

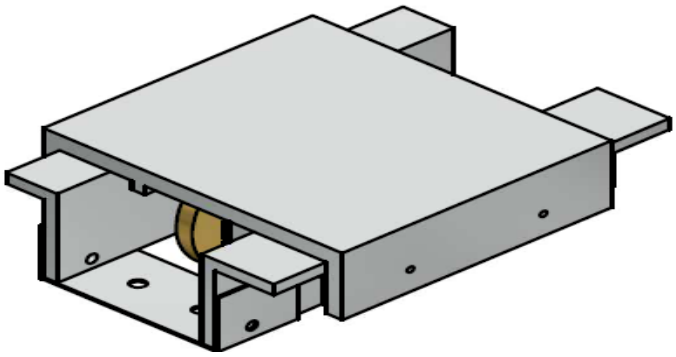


Figura 1.9. Dibujo de una de las mesas deslizantes.

Al igual que las masas deslizantes del sistema anterior, el servomotor también puede mover la platina a un lugar exacto y en desplazamientos pequeños; pero se tiene ahora la ventaja de que se puede realizar esta acción con un retardo más corto y así optimizar el tiempo de balanceo por masas deslizantes. (Escobedo L. 2012)

El servomotor utilizado para mover la platina de las mesas deslizantes es el modelo HS-645MG de la marca Hitec. A diferencia del motor a pasos que se utiliza en el

SIMUSAT 2.1, no necesita que se le envíen una secuencia de datos para que avance, sino una señal PWM, cambiando su posición según sea requerido. Puede mover la platina a un lugar exacto en desplazamientos pequeños y con un mínimo retardo. En el capítulo dos se analiza a detalle este nuevo sistema.

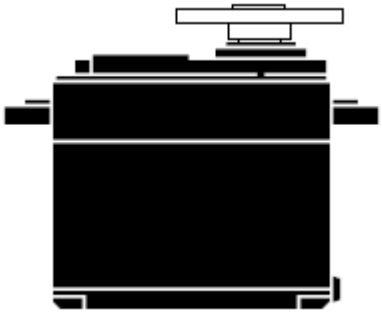


Figura 1.10. Servomotor HS-645MG HITEC

1.4. Programación de rutinas.

Han sido elaboradas las rutinas necesarias para poner en marcha el servomecanismo en una prueba básica de orientación, controlando la posición de las masas deslizantes hasta compensar la inestabilidad generada por las fuerzas existentes sobre la plataforma. Esas rutinas son llamadas por medio de la interfaz programada en Visual Basic para el SIMUSAT 2.1, la cual ha sido modificada, no solo para agregar la nueva instrucción, sino para hacerla más apegada a la realidad al tener ahora una representación gráfica más exacta de la plataforma. La nueva programación de las rutinas de despliegue de la posición en tiempo real de la plataforma se describe en el capítulo 4.

Capítulo

2

Modelado dinámico de la plataforma

Para llevar a cabo el control de orientación de la plataforma de simulación, es necesario hacer un modelado que nos describa su comportamiento dinámico. Para simplificar este proceso de modelado, es necesario hacer algunas consideraciones como el suponer que la plataforma se comporta como un cuerpo rígido, que los momentos de inercia principales pueden ser determinados de manera precisa y pueden ser posicionados de tal manera que los pares de control, puedan ser aplicados alrededor de cada uno de los ejes principales de inercia.

Existen diferentes maneras de representar la orientación de un cuerpo rígido, sin embargo en este capítulo solamente hablaremos de la matriz de cosenos directores, que es la parametrización más utilizada para este fin. Se desarrolla un modelo simplificado de las ecuaciones de movimiento cuando se tiene solamente rotación, para pruebas de funcionamiento en Tierra con el sistema de simulación utilizado en este trabajo de tesis.

2.1. Modelo de cuerpo rígido de la plataforma.

Cualquier vector **A** puede ser expresado como:

$$\mathbf{A} = A(\cos\alpha\,\mathbf{i} + \cos\beta\,\mathbf{j} + \cos\gamma\,\mathbf{k}) \qquad (2.1)$$

donde α, β, γ son los ángulos que forma **A** con los ejes **i** , **j** , **k** respectivamente y cos α, cos β, cos γ son conocidos como los cosenos directores de **A**

Para poder determinar la orientación en tres ejes de una nave, es necesario llevar a cabo una transformación para pasar de un sistema de coordenadas fijo al cuerpo del satélite a un sistema de coordenadas inercial. Consideremos que nuestro satélite es un cuerpo rígido, asumamos que existe una tercia de vectores unitarios $\vec{u},\vec{v},\vec{w}$ ortogonales, fijos al cuerpo de la nave, de tal manera que:

$$\vec{u} \times \vec{v} = \vec{w}$$

El problema básico es especificar la orientación de esta tercia, con respecto a algún marco de referencia fijo. Ver figura 2.1

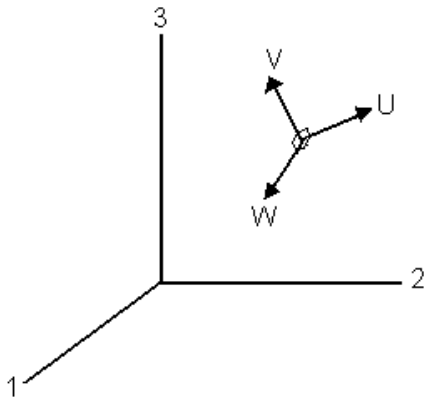


Figura 2.1 El problema fundamental de la parametrización de la orientación en tres ejes, consiste en especificar la orientación de los ejes de la nave $\vec{u},\vec{v},\vec{w}$ en el marco de referencia 1,2,3.

Si podemos relacionar los componentes de $\vec{u},\vec{v},\vec{w}$ a lo largo de los tres ejes del marco de coordenadas 1,2,3 tendremos entonces la orientación completamente definida. Para esto necesitamos nueve parámetros, que pueden ser vistos como los elementos de una matriz A de 3x3, llamada *matriz de orientación*:

$$A \equiv \begin{bmatrix} u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \\ w_1 & w_2 & w_3 \end{bmatrix} \qquad (2.2)$$

donde $\vec{u} = (u_1,u_2,u_3)^T, \vec{v} = (v_1,v_2,v_3)^T, \vec{w} = (w_1,w_2,w_3)^T$. Cada uno de estos elementos es el coseno del ángulo entre un vector unitario fijo al satélite y uno de los ejes de referencia; u_1 por ejemplo, es el coseno del ángulo formado entre $\vec{u}$ y el eje inercial 1. Es por esta razón que a la matriz A, se le llama matriz de cosenos directores. Sus elementos no son todos independientes, por ejemplo, el hecho de que $\vec{u}$ sea un vector unitario, implica que:

$$u_1^2 + u_2^2 + u_3^2 = 1 \tag{2.3}$$

y la ortogonalidad entre $\vec{u}$ y $\vec{v}$, significa que:

$$u_1 v_1 + u_2 v_2 + u_3 v_3 = 0 \tag{2.4}$$

Todas estas relaciones pueden resumirse con la siguiente igualdad: el producto de A por su transpuesta, es la matriz identidad.

$$A\, A^T = 1 \tag{2.5}$$

Esto quiere decir que A es una matriz ortogonal real. La definición del determinante de A es:

$$\det A = \vec{u} \cdot (\vec{v} \times \vec{w}) \tag{2.6}$$

El hecho de que los vectores $\vec{u}, \vec{v}, \vec{w}$ formen una tercia ortogonal (en el sentido positivo), quiere decir que *det* A=1. Entonces, A es una matriz ortogonal real propia. **La matriz de cosenos directores es una transformación de coordenadas que mapea vectores desde el marco de referencia inercial, hacia el marco fijo al cuerpo del satélite.** Esto quiere decir que si $\vec{a}$ es un vector con componentes a_1, a_2, a_3 localizado a lo largo de los ejes de referencia, entonces:

$$A\vec{a} = \begin{bmatrix} u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \\ w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} \vec{u} \cdot \vec{a} \\ \vec{v} \cdot \vec{a} \\ \vec{w} \cdot \vec{a} \end{bmatrix} \equiv \begin{bmatrix} a_u \\ a_v \\ a_w \end{bmatrix} \tag{2.7}$$

Los componentes de $A\vec{a}$ son los mismos del vector $\vec{a}$ a lo largo de la tercia $\vec{u}, \vec{v}, \vec{w}$ fija al satélite. Una transformación de una matriz ortogonal real propia, conserva las longitudes de los vectores y los ángulos entre ellos. Esto significa que puede representar una *rotación.* El producto de dos matrices ortogonales reales propias $A'' = A'A$ representa los resultados de rotaciones sucesivas, primero por A y luego por A', en ese orden. Debido a que la transpuesta y la inversa de una matriz ortogonal son idénticas, A^T mapea los vectores fijos al cuerpo del satélite, hacia el marco de referencia inercial. También existe el hecho de que una matriz de 3x3 ortogonal real propia, tiene por lo menos un vector característico, cuyo valor característico es la unidad. Esto quiere decir que existe un vector unitario $\vec{e}$, que permanece inalterado al ser premultiplicado por A:

$$A\vec{e} = \vec{e} \qquad (2.8)$$

El vector $\vec{e}$ tiene las mismas componentes a lo largo de los ejes del satélite y a lo largo de los ejes de referencia. Entonces $\vec{e}$ es un vector que se encuentra a lo largo del eje de rotación. La existencia de $\vec{e}$ demuestra la validez del teorema de Euler: *La rotación de un cuerpo rígido alrededor de un punto fijo, es equivalente a una rotación alrededor de un eje que pasa a través de dicho punto.*

Nosotros vemos a la matriz de cosenos directores como una relación fundamental que nos especifica la orientación de un cuerpo rígido. Sin embargo, algunas otras parametrizaciones, que se resumen en la tabla 2.1, pueden ser más convenientes dependiendo de la aplicación (Wang P., Yee J. and Hadaegh F.). En todos los casos, siempre relacionaremos los parámetros con la matriz de cosenos directores.

Tabla 2.1 Representaciones alternativas de la orientación en tres ejes.

Parametri-zación	**Notación**	**Ventajas**	**Desventajas**	**Aplicaciones más comunes.**
Matriz de Cosenos directores	$A=[A_{ij}]$	No existen singularidades ni funciones trigonométricas. La regla de productos es adecuada para representar rotaciones sucesivas.	Seis parámetros redundantes.	En el análisis para transformar vectores de un marco de referencia a otro.
Ejes de Euler y ángulo de Euler	e, Φ	Clara representación física.	Un parámetro redundante Un eje indefinido cuando Φ =0.	Comando de maniobras de orientación.
Parámetros simétricos de Euler (quaterniones) .	q_1,q_2,q_3,q_4 (q)	No tiene singularidades ni funciones trigonométricas. La regla de productos es adecuada para representar rotaciones sucesivas.	Un parámetro redundante. No hay una interpretación Física obvia.	Navegación inercial a bordo.
Vector de Gibbs	**g**	No existen singularidades ni funciones trigonométricas. La regla de productos es adecuada para representar rotaciones sucesivas.	Valor infinito para una rotación de 180^0	Estudios analíticos.
Ángulos de Euler.	ϕ,θ,ψ	No tiene parámetros redundantes. La interpretación física es clara en algunos casos.	Con algunos valores de θ presenta singularidades. La regla de productos no es adecuada para representar rotaciones sucesivas.	Estudios analíticos. Entrada/ Salida. Control de orientación a bordo en naves estabilizadas en tres ejes.

2.1.1 Ecuaciones de Euler.

Es muy común utilizar las ecuaciones de Euler para la determinación de la orientación de un cuerpo rígido. Éstas pueden ser escritas de la siguiente manera (Peterson G):

$$T_1 = I_{xx}\dot{\omega}_x + (I_{zz} - I_{yy})\omega_y\omega_z$$

$$T_2 = I_{yy}\dot{\omega}_y + (I_{xx} - I_{zz})\omega_x\omega_z \quad (2.9)$$

$$T_3 = I_{zz}\dot{\omega}_z + (I_{yy} - I_{xx})\omega_x\omega_y$$

Donde T_1, T_2, T_3 son los momentos externos alrededor de los ejes principales; I_{xx}, I_{yy}, I_{zz} son los momentos de inercia principales, y $\omega_x, \omega_y, \omega_z$ son las velocidades angulares alrededor de los mismos ejes principales.

Las ecuaciones y los ángulos de Euler constituyen un método clásico para describir la posición de un cuerpo rígido en el espacio, con respecto a un sistema de coordenadas inercialmente fijo (Hughes P. C). Las ecuaciones de Euler están basadas en ejes fijos al cuerpo. Para poder visualizar el movimiento descrito por estas ecuaciones, en un sistema de coordenadas inercial, un juego de ángulos de rotación o de orientación debe aplicarse a estas ecuaciones. Los ángulos de Euler se utilizan de manera común en la determinación de la orientación de naves espaciales. Algunas secuencias presentan singularidades en la posición horizontal, que es precisamente la orientación más probable de un satélite en su apuntamiento continuo hacia la Tierra, esta es una razón muy importante para no emplearlas. Usar una secuencia diferente de ángulos de Euler, significa trasladar la singularidad a otra posición en la que no cause problemas.

2.2. Corrimiento del centro de masa de la plataforma de simulación

El método de balanceo de la plataforma de simulación es estático y únicamente se lleva a cabo en dos ejes (rotación y cabeceo) (Prado J., Bisiacchi G., Mesinas M., Ruiz D.). Esto se efectúa, por medio de las señales de dos inclinómetros localizados dentro de la brújula electrónica y que determinan la orientación de la plataforma con respecto a la horizontal local y como complemento de estos sensores, se utilizan dos masas deslizantes para reposicionar el centro de masa. Este balanceo es exclusivamente estático y su justificación está basada en el hecho de que los satélites estabilizados por giro, no se encuentran contemplados en nuestras aplicaciones, por tanto, no es necesario el balanceo dinámico.

El balanceo estático es aplicable al tipo de maniobras que se pretenden efectuar en el espacio, y que consisten en probar los subsistemas de orientación y control para un satélite estabilizado por gradiente gravitacional y que emplea bobinas magnéticas y ruedas inerciales para realizar maniobras de apuntamiento. Este tipo de estabilización

implica que el satélite le dará una vuelta a la Tierra cada 90 o 100 minutos, con una de sus caras apuntando continuamente hacia nadir, por lo que se considera que para todo fin práctico no se encuentra girando. A partir de esta situación, se ha determinado que la plataforma únicamente será balanceada estáticamente. Los requisitos para el equilibrio estático son simplemente que la suma de todas las fuerzas en el sistema sean iguales a cero.

$$\Sigma F - ma = 0 \quad (2.10)$$

$$\Sigma F = 0 \text{ ; suma de fuerzas igual a cero.} \quad (2.11)$$

$$\Sigma M = 0 \text{ ; suma de momentos igual a cero.} \quad (2.12)$$

El equilibrio estático se logra cuando la plataforma se encuentra perfectamente horizontal, ya que esto significa que no existe ninguna fuerza actuando fuera del centro de masa del conjunto (Prado J., Bisiacchi G., Mesinas M., Ruiz D.). Con base en esta idea, se desarrolló un sistema de control que permite equilibrar de manera automática la plataforma de simulación, solamente en los ejes de alabeo (X) y cabeceo (Y), dentro de varios intervalos: $\pm$ 3, $\pm$ 2, $\pm$ 1, $\pm$ 0.5 y $\pm$ 0.25 grados, con respecto a la propia horizontal (Escobedo L. 2012).

2.3. Sistema de masas deslizantes para balanceo automático.

Las masas deslizantes tienen la capacidad de mover el centro de masa de la plataforma. Su operación está basada en el movimiento de una platina, a lo largo de una cremallera, por medio de un servomotor. Los sistemas de masas deslizantes son los componentes más importantes en el balanceo de la plataforma, ya que de ellos depende que se logre un par residual de pequeñas proporciones (Juarez A., 2001). Esto significa tener un valor de unas pocas decenas de gramos-cm de desbalanceo y mucho depende de la manera en que se lleve a cabo este proceso. En este caso en particular, de la resolución que se obtiene en el movimiento de la platina por el servomecanismo y la precisión en la orientación que entrega la brújula electrónica.

El sistema de masas deslizantes para el balanceo automático tiene como principal objetivo hacer el centro de masa de todo el conjunto, coincidente con el centro de rotación de cojinete neumático, para así reducir al máximo la influencia de los pares gravitacionales (AeroAstro 2006). Sin embargo, hay que hacer notar que este proceso tradicional de balanceo, será alterado al momento en que las masas deslizantes sean utilizadas para compensar los pares gravitacionales, ya que éstas serán las que

cancelen todos los pares actuantes sobre la plataforma y que ésta quede en una posición semejante a la que tendría en un ambiente de gravedad reducida.

2.3.1. Balanceo manual o acomodo inicial de componentes.

El balanceo manual de la plataforma, es un procedimiento que debe seguirse independientemente del método de balanceo automático que pueda llevarse a cabo posteriormente. La plataforma de simulación se encuentra inicialmente balanceada cuando no tiene ningún otro componente, debido a que la masa se encuentra distribuida de manera homogénea en toda su superficie. El procedimiento que seguiremos consiste básicamente en mantener balanceada la plataforma; añadiendo un par de componentes cada vez, contrarrestando cada uno el efecto de desbalanceo causado por el otro. La importancia de esta estrategia radica en que una vez alcanzado el equilibrio, es posible añadir más componentes por pares, manteniendo sin alteraciones el balanceo (Prado J., Bisiacchi G., Mesinas M., Ruiz D.).

Para clarificar un poco este procedimiento, vamos a presentar un ejemplo donde incluiremos todos los componentes del sistema SIMUSAT 3.0. Lo primero que debemos hacer es identificar a cada componente, tener su masa, como se observa en la tabla 2.2. Todos los pasos subsecuentes de fijación de elementos deben de mantener el centro de masa sobre el centro del cojinete neumático esférico, determinando cual debe ser la localización de cada uno de ellos, siguiendo un cálculo sencillo.

Es indispensable optimizar el acomodo de los componentes, primero se seleccionan aquellos que tienen necesidades específicas de localización; como los sensores y los actuadores y se fijan en un determinado lugar, a continuación, y siguiendo un eje imaginario que une el centro de masa del dispositivo en cuestión y que pasa por el centro de la plataforma, se coloca otro componente o un lastre, de tamaño conveniente, a una determinada distancia del centro de la plataforma para que el sistema quede balanceado (necesariamente en la parte baja de la plataforma para que el centro de masa se mantenga sobre el plano de la plataforma).

Tabla 2.2 Características de los componentes que integran la plataforma de simulación.

Elemento	Masa [kg]	Subtotal por elementos [kg]	Forma
Balero de aire	0.200	0.200	Semiesfera
Plataforma	1.173	1.173	Disco
Mesa deslizante completa	0.930	1.860	Paralelepípedo
Platina deslizante	0.545	1.090	Placa
Bobinas magnéticas	0.289	0.867	Tubular

(3 unidades)			
Motores (3 unidades)	0.348	1.044	Cilíndrica
Ruedas inerciales Ejes X,Y.	0.145	0.290	Disco
Rueda inercial eje Z	0.090	0.090	Disco
Circuito de control	0.749	0.749	Paralelepípedo
Baterías (2@6 volts)	0.753	1.506	Paralelepípedo
Batería @12 volts	1.700	1.700	Paralelepípedo
Masa equilibrante	0.466	0.466	Cilindro
		Masa total 10.945	

El arreglo de los componentes queda como se muestra en la figura 2.3. Se colocó la bobina del eje Z y se ubicó a la brújula en su parte superior, es decir, en el centro de la plataforma alineando los ejes de este sensor de posición, con los ejes de la plataforma X, Y y Z y lograr que las mediciones se realicen con respecto a los mismos ejes de referencia. En segundo lugar se fijaron las dos mesas deslizantes sobre los ejes X,Y, respectivamente, debido a que son los ejes que van a ser controlados. Como contraparte a ambas, se colocaron las 3 baterías. Posteriormente se fijó el circuito de control y las bobinas de los ejes X,Y y para contrarrestar estos componentes, se colocaron los motores en los ejes X,Y con sus respectivas ruedas. Se colocó finalmente el motor del eje Z y para balancear todo el sistema se colocaron dos masas cilíndricas.

De esta manera se hizo el acomodo inicial y ahora corresponde al sistema de balanceo automático colocar el centro de masa en el centro geométrico del balero de aire.

Cuando el sistema no está balanceado, existe una diferencia entre el centro de masa de la plataforma de simulación y el centro geométrico del cojinete neumático esférico. Es labor del sistema de balanceo automático llevar a cabo este ajuste, aunque como lo muestran las ecuaciones de movimiento del simulador, los ejes están acoplados y el movimiento en uno de ellos genera una perturbación en los otros, finalmente todas las perturbaciones son compensadas automáticamente en el proceso de balanceo. En este caso el proceso fue llevado a cabo como se mencionó en la sección 2.2.

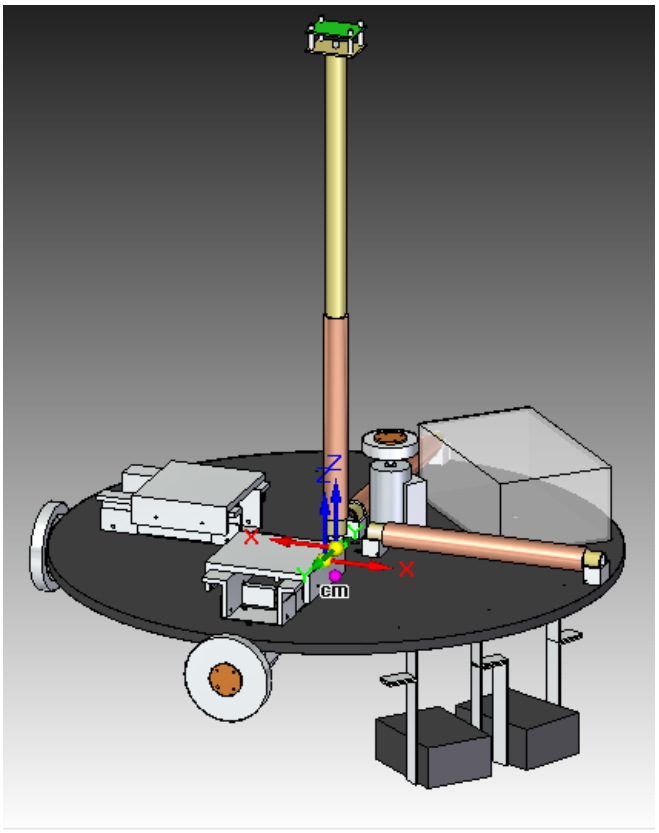


Figura 2.3. Distribución de los componentes en la plataforma de simulación.

Capítulo

3

Sistema de seguimiento servomecánico

En el desarrollo del más reciente sistema de simulación llevado a cabo en el instituto de Geografía de la UNAM, SIMUSAT 3.0 (Escobedo L. 2012), se ha planteado dar solución a uno de los problemas que se presentan durante la ejecución de pruebas de control de orientación: la oscilación causada por los efectos de la atracción gravitacional terrestre sobre la plataforma. El subsistema de orientación no puede ser probado adecuadamente debido a que cualquier movimiento impulsado por acción de los actuadores, provoca un par restaurador -causado por el propio peso de los elementos del simulador-, derivando en un movimiento oscilatorio de la plataforma. Esto significa que una vez llevada la plataforma a determinada posición, el algoritmo debe ser interrumpido en ese momento, ya que los sensores comienzan a enviar información sobre una posición, que está afectada por la oscilación mencionada, y que provoca un reinicio de la ejecución del algoritmo de control. Esto es particularmente notable en los ejes de Rotación y Cabeceo, no así en el de Guiñada que permanece invariante ante esta situación.

Esta oscilación intermitente impide hacer un análisis adecuado de resultados, por esta razón se ha propuesto añadir para el SIMUSAT 3.0, un sistema independiente de compensación que equilibre la acción de estos pares externos, un elemento mecánico capaz de aplicar fuerzas iguales y opuestas a las que existen por causa de la gravedad terrestre sobre la plataforma. Debido a las restricciones en la masa máxima del simulador (10 kg), fue necesario cambiar el concepto y los componentes mecánicos de balanceo automático manejados en el simulador anterior, ya que ahora se tiene el objetivo de compensar los pares gravitacionales externos. Para compensar estas perturbaciones y así poder observar el efecto que producirían los actuadores en un ambiente de gravedad reducida, se ha elegido sustituir el sistema de masas deslizantes que se venia usando regularmente, por un servomecanismo.

Además, si el sistema de simulación quiere usarse de la manera tradicional, es decir, haciendo un balanceo estático en dos ejes y luego efectuar pruebas de control de orientación, se puede hacer, ya que los servomecanismos pueden efectuar también la tarea de balanceo (Escobedo L. 2012).

En este capítulo se describe el rediseño de las masas deslizantes, así como los elementos utilizados para su desarrollo y su acoplamiento a la plataforma de simulación.

3.1 Seguidor servomecánico.

La idea de incluir un sistema de seguimiento servomecánico surge por la necesidad de una respuesta rápida y precisa en el control de la posición del centro de masa del simulador, a fin de estabilizar las perturbaciones con mayor celeridad. Al modificar el mecanismo de acción del sistema de masas deslizantes para convertirlo en balanceo-compensador, se requería una respuesta más rápida de la que podían brindar los motores a pasos y el sistema de tornillo con que se contaba. Allí se planteó la opción de sustituir los motores a pasos por servomotores y diseñar un mecanismo piñón-cremallera para cada masa deslizante.

Una de las ventajas de implementar un mecanismo de este tipo, fue el reducir el número de salidas utilizadas del microcontrolador (Rabbit Semiconductor 2007), necesarias para manejarlo. En el SIMUSAT 2.1 se requería un controlador adicional PIC con todos sus elementos de potencia para controlar y polarizar los motores a pasos; al cambiarlos por servomotores sólo se utilizan las salidas PWM de la computadora de abordo y la alimentación que provee la misma tarjeta que polariza al microcontrolador principal.

3.1.1 Generalidades.

En la actualidad, los sistemas de control llevan a cabo una importante función en el desarrollo de la tecnología moderna. Prácticamente todos los aparatos electrónicos de uso común poseen algún tipo de sistema de control, como por ejemplo el calentador de agua, o el horno de microondas. El tipo de control depende de las necesidades de cada sistema, la precisión requerida y las condiciones de las variables que lo integran. El esquema básico de un sistema de control es el siguiente:

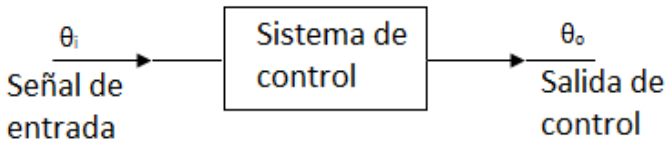


Figura 3.1. Esquema básico de control

La salida θ_o es controlada por la entrada θ_i mediante los elementos que forman el sistema de control.

3.1.1.1 Sistemas de lazo abierto o no realimentados.

En un sistema de lazo abierto, la aplicación de una señal de entrada produce una salida de potencia, pues la señal de entrada suele ser amplificada, pero su reacción depende del criterio humano, dando la salida correspondiente sin corrección alguna (Bucley, V.). La salida no es comparada con el valor de la señal de entrada o referencia. Un ejemplo de este tipo de sistema de control, sería un calentador de agua sin termostato.

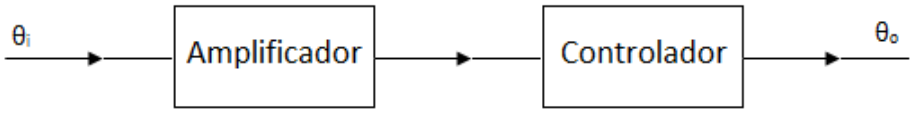


Figura 3.2. Diagrama de bloques de un sistema de control de lazo abierto

3.1.1.2 Sistemas de lazo cerrado o realimentados.

A fin de obtener un control más preciso en la salida, se utiliza el concepto de realimentación. Esta consiste en reintroducir la salida nuevamente al sistema, para compararla con la señal deseada establecida por la señal de mando de entrada, y cualquier diferencia entre ambas es conducida a través del sistema, haciéndola actuar para corregir el error.

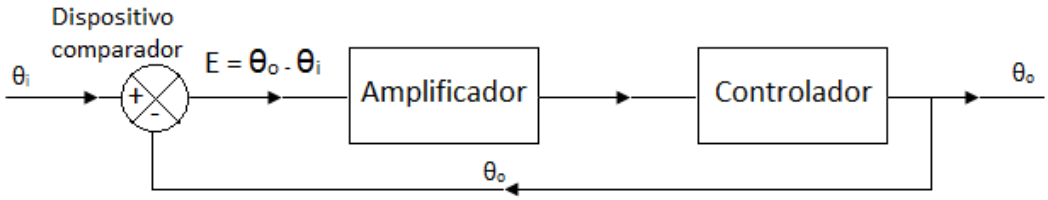


Figura 3.3. Diagrama de bloques de un sistema de control de lazo cerrado o realimentado

Un sistema de lazo cerrado ofrece una mayor precisión de control que un sistema no realimentado, pero puede resultar inestable. Por ejemplo, si la ganancia es demasiado elevada puede dar lugar a una acción correctora excesiva y hacer que la salida oscile continuamente (Manual Servomotores HITEC).

3.1.1.3 Servomecanismo

Un servomecanismo es un tipo particular de control con realimentación en el que una o más señales representan posiciones mecánicas o movimientos. Poseen las siguientes propiedades:

a) Son accionados por una magnitud proporcional a una desviación o error.

b) Tienen amplificación de potencia.

c) Contienen un mecanismo (partes móviles).

d) Operan automáticamente.

Los servomecanismos son sistemas seguidores y son utilizados en múltiples aplicaciones, como son: buscadores de radar, control de tiro, gobierno de barcos, brazos robóticos, mecanismo de frenado de automotores, etc.; de aquí que sus señales de entrada cambien continuamente y se requiere un control de posición. Se trata de dispositivos capaces de captar información del medio y de modificar sus estados en función de las circunstancias y regular su actividad de cara a la consecución de una meta (Stockdale, L.A).

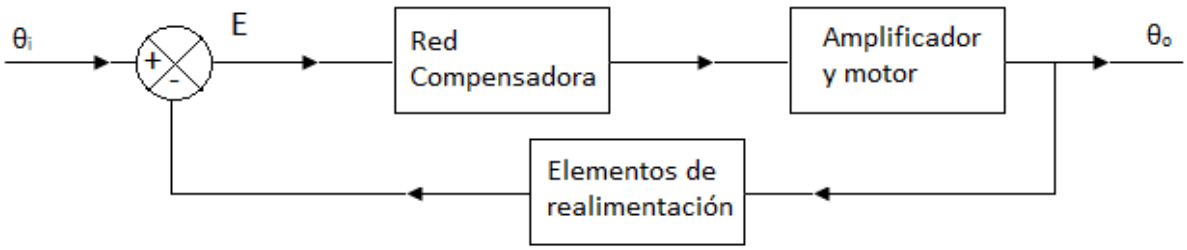


Figura 3.4. Elementos básicos de un servomecanismo.

3.1.1.4 Tipos de servomecanismos.

Los servomecanismos pueden clasificarse de acuerdo a varios criterios, con respecto a la forma en que actúa la señal a la salida, se dividen en:

1) *Servomecanismo de señales continúas.* Aquellos en los que las señales en distintas partes del sistema son todas funciones continúas en el tiempo. Ejemplo: La regulación de tensión en un generador.

2) *Servomecanismos del tipo encendido apagado (ON-OFF).* Son en los que la función de la señal de error abre o cierra el paso de potencia, sin estados

intermedios. Una vez establecidos no hay control posterior, sólo se pasa bruscamente de un estado a otro. Ejemplo: el control de calefacción mediante termostato.

3) *Servomecanismos muestreados.* Son aquellos en los que la señal de control es muestreada a intervalos de tiempo, es decir, la señal aparece de modo intermitente. Ejemplo: Sistema de seguimiento por radar, dónde la información se obtienen por impulsos periódicos.

Un servomecanismo es, esencialmente, un sistema seguidor o reproductor en el que la posición de salida sigue o reproduce a la señal de entrada (referencia).

3.2 Diseño del sistema de masas deslizantes.

El diseño del sistema de masas deslizantes tiene algo en común con el sistema de balanceo del SIMUSAT 2.1. Al igual que éste, las mesas de aluminio pretenden modificar el centro de masa de la plataforma para moverlo hacia el centro geométrico del balero de aire esférico, logrando así la estabilidad de la mesa por medio de un sistema de balanceo en dos ejes (Juarez A., 2001). El modelo anterior realiza un ajuste fino y preciso del centro de masa por medio de un sistema de tornillo y motores a pasos, aunque una desventaja de este sistema es su tardanza en este proceso, un ajuste a un nivel de error de medio grado puede tardar más de 25 minutos en terminar. Es más que evidente que un sistema con estas características no podía ser usado para los fines aquí propuestos. Además, el cambio de posición en los motores de pasos es muy lento, no pudiendo iniciar algún tipo de control desde otra posición arbitraria.

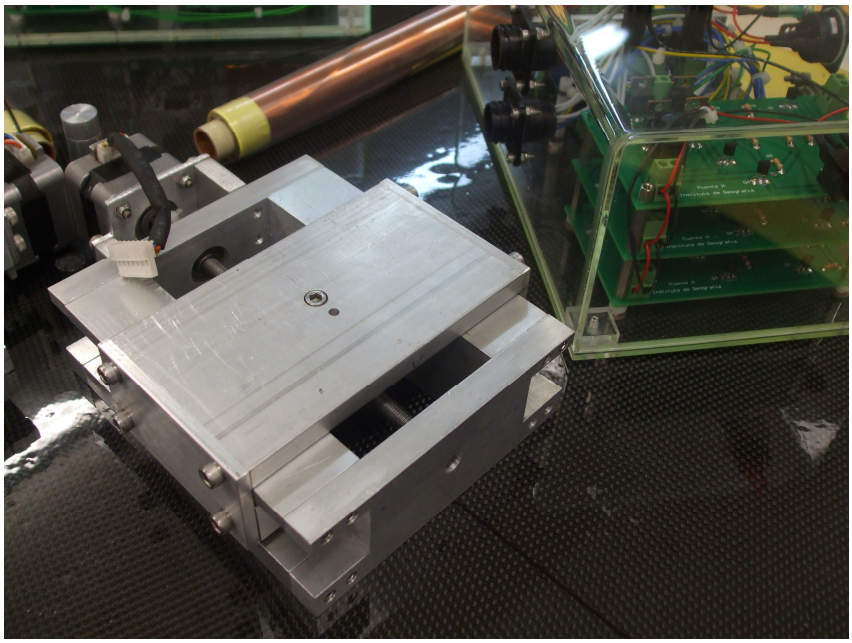


Figura 3.4. Mesa del SIMUSAT 2.1, con sistema de tornillo y motor a pasos (Juarez A., 2001).

El nuevo diseño pretende agilizar ese proceso utilizando servomotores, que a diferencia de los motores a pasos, pueden tomar una posición inicial diferente en menos de un segundo, abarcando 180° de giro sobre su propio eje. Esta consideración se pensó no únicamente para el balanceo, sino para poder aplicar distintas operaciones de control automático en el mismo sistema, permitiendo incorporar los algoritmos de compensación gravitacional necesarios en las pruebas de orientación.

3.2.1 Consideraciones de diseño y manufactura de las masas deslizantes.

Los sistemas de masas deslizantes son los componentes más importantes en el balanceo de la plataforma, ya que de ellos depende que se logre un par residual de pequeñas proporciones (Juárez A. 2001). Las masas deslizantes para nuestro nuevo simulador, cuentan con una masa de 700 g, incluyendo todos sus componentes, como los tornillos para sujetarla a la plataforma. Sus componentes principales son los siguientes: base; que es donde se deslizará la platina y donde se montará el servomotor, platina deslizante; es la pieza que se moverá según se necesite en la etapa de balanceo y por último, el servomotor; que es el elemento que mueve a la platina por medio de un engrane. Y mediante una de las salidas de la computadora de abordo, se envía una señal PWM para que nuestro servo se mueva según lo necesite la masa.

La distancia máxima de desplazamiento de la parte deslizable de la masa es de 8.8 [cm]; esto nos ayuda para hacer ajustes grandes, aún con balanceo manual inadecuado. En la figura 3.5. se muestra el nuevo diseño de las masas deslizantes

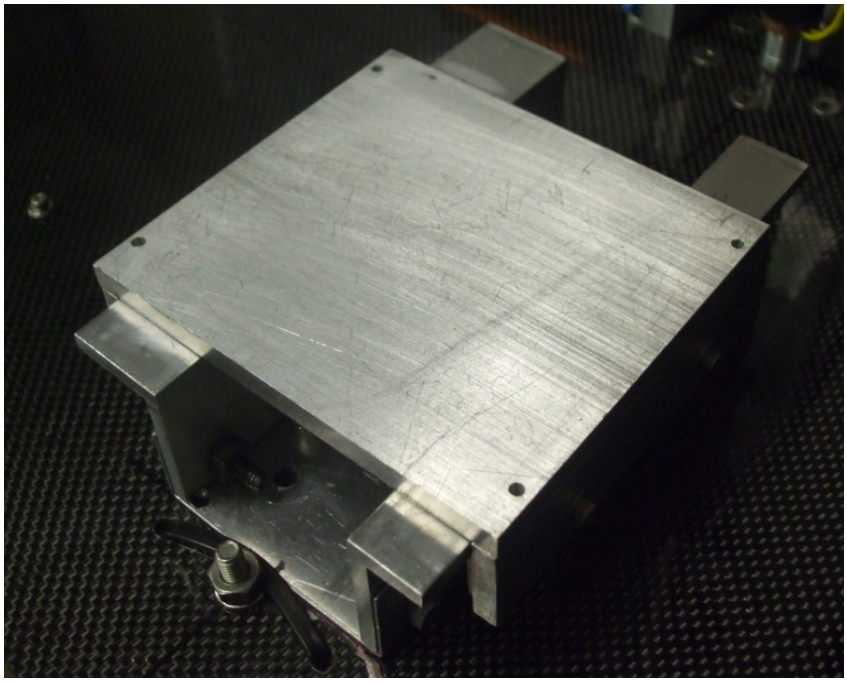


Figura 3.5. Mesa del SIMUSAT 3.0, con sistema de engrane y servomotor.

Dentro de la masa deslizante, para que se pueda efectuar el movimiento de la platina, se acopló un servomotor HS-645MG de la marca HITEC con un engrane de 2 [in] de diámetro, que cuenta con 40 dientes; más que suficiente para que se tengan desplazamientos hasta de un cuarto del desplazamiento que le corresponde a cada diente de nuestro engrane, lo que sería 4.5 [°]. Éste está acoplado con la platina para que no tenga ningún problema al moverse o no se disloque la platina de su base al inclinarse la plataforma.

La platina cuenta con un trozo de banda dentada de hule para que trabaje en conjunto con el engrane y así garantizar la movilidad sin ningún tipo de juego entre las dos piezas. Se eligió dicha banda porque es la que se acoplaba a nuestro engrane y proporciona una solución adecuada, pero en caso que se necesiten desplazamientos de mayor precisión, se le puede maquinar a la platina una sección dentada que concuerde con el engrane o también aumentarle el número de dientes, de acuerdo a lo que demande la exactitud del movimiento.

El sistema engrane cremallera se muestra en la Figura 3.6. (Mecanismo piñón-cremallera).

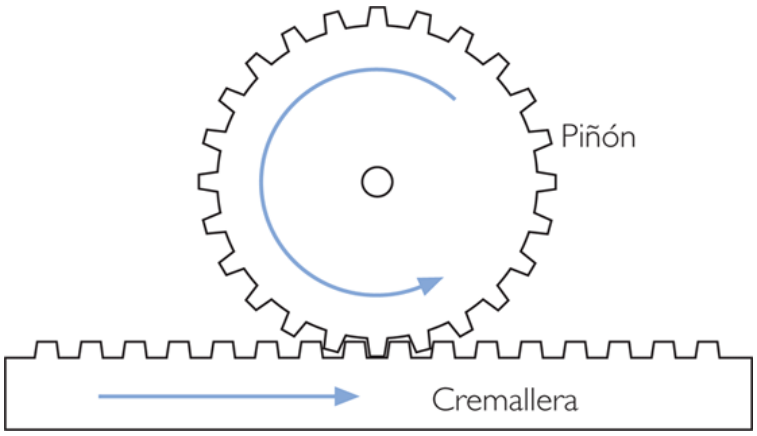


Figura 3.6. Sistema engrane cremallera implementado en la masa deslizante.

3.2.2 Dibujos de las mesas deslizantes.

En la figura 3.7. se muestran los planos de un conjunto completo de la mesa deslizante con su platina.

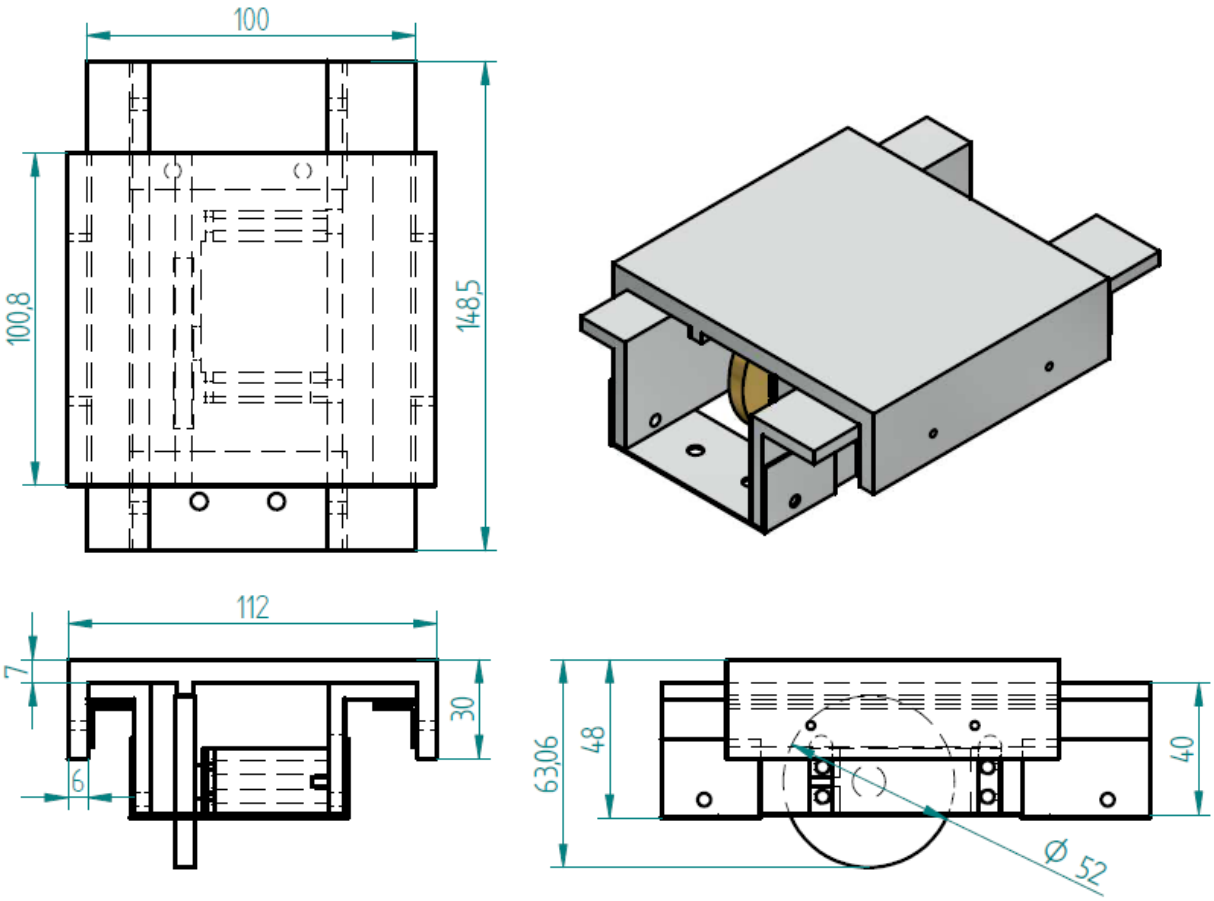


Figura 3.7. Dibujos de las masas deslizantes.

La elaboración de todos los elementos se efectuó en el Laboratorio de Percepción Remota Alternativa y Tecnología Avanzada del Instituto de Geografía de la UNAM; al ser un prototipo no fue necesario acudir a ningún taller de manufactura especializado. En la siguiente generación, cuando se establezcan las mejoras del sistema actual, la fabricación será por medio de máquinas de control numérico.

3.3 Servomotores.

La necesidad de una respuesta más rápida y de un control de corrimiento más preciso derivó en el uso de los servomotores para el sistema de masas deslizantes. Un servomotor es un dispositivo similar a un motor de corriente continua que tiene la capacidad de ubicarse en cualquier lugar dentro de su intervalo de operación y mantenerse estable en esa posición (Irving L.). Estos dispositivos están conformados por un motor, una caja reductora y un circuito de control (ver figura 3.8).

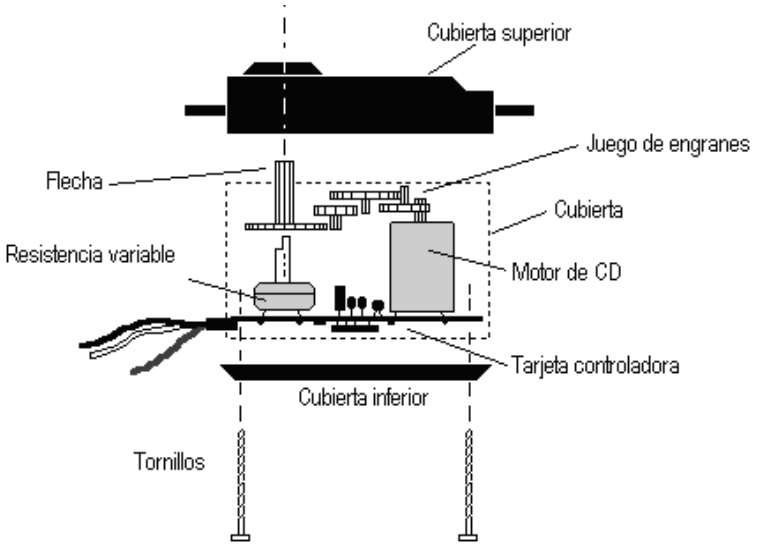


Figura 3.8. Diagrama esquemático de un Servomotor.

Los engranes reductores se encargan de convertir gran parte de la velocidad de giro del motor de corriente continua, en torque; y la circuitería se encarga del control de la posición del motor. En el último engrane acoplado al eje de salida se obtiene una velocidad notablemente reducida, a pesar de que dentro del sistema el motor está girando a altas velocidades. El servomotor recibe una señal PWM a la entrada y ubica al motor en su nueva posición, dependiendo del valor del ancho de pulso recibido, esto se explicará con más detalle un poco más adelante. El potenciómetro conectado al eje central del motor, entrega indirectamente el ángulo actual del servo motor. Si el eje está en el ángulo correcto, entonces el motor estará apagado.

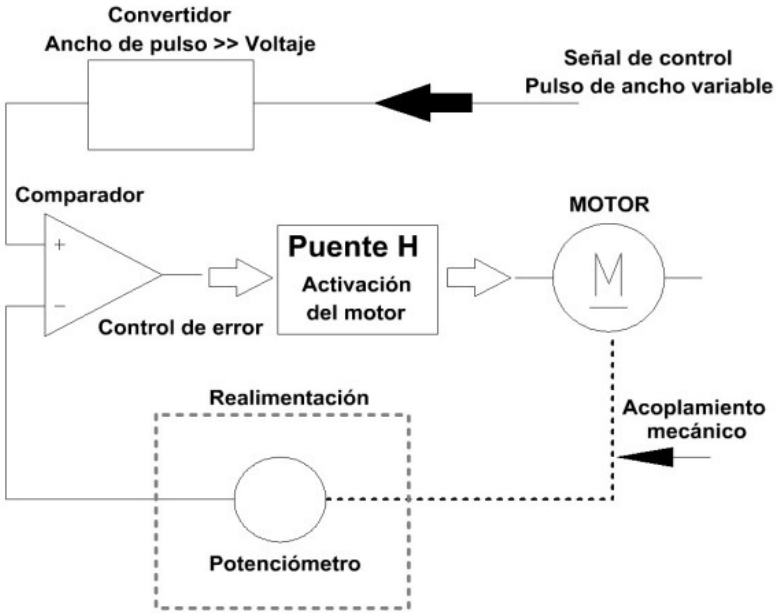


Figura 3.9. Diagrama de bloques de la estructura interna de un servomotor

Si el circuito verifica que el ángulo no es el correcto, el motor girará en la dirección necesaria hasta alcanzarlo, dentro del intervalo de 0° a 180°. Para mantener el servomotor en esa posición es necesario enviarle continuamente la señal PWM que corresponde, así el sistema de control seguirá operando y conservará su colocación resistiendo a las fuerzas externas que intenten moverlo. Si la señal sale de los márgenes de operación, el servomotor perderá fuerza de torque y estabilidad. Cada servomotor, dependiendo de la marca y modelo utilizados, tiene sus propios márgenes de operación.

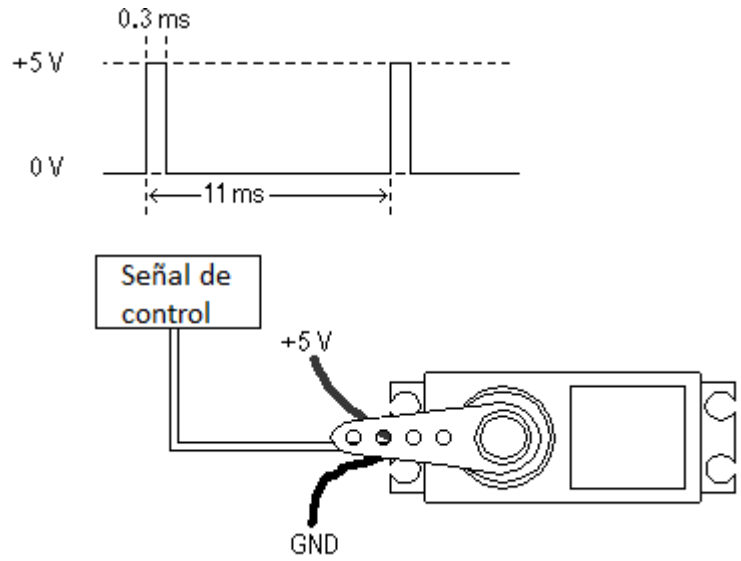


Figura 3.10. Tipo de señal de control requerida por un servomotor convencional.

La tensión de alimentación puede estar comprendida entre 4 y 8 V. Lo común es que sea de 6 V, pues aunque el motor soporta mayores voltajes de trabajo, el circuito de control no lo hace.

3.3.1 Características del servomotor utilizado

A fin de contar con la fuerza suficiente para mover el sistema de masas sin dificultad alguna, se eligió el servomotor HITEC HS-645MG, capaz de brindar un torque mayor al de servomotores convencionales. Cuenta con engranaje de metal y puede brindar un torque máximo de 9.6 kg·cm. Su velocidad máxima es de 0.24 s/60°, esto significa que puede rotar su eje 60 grados cada 240 [ms], sin carga. La posición de su eje depende del valor de ancho de pulso de la señal de PWM aplicada, y se encuentra dentro de los rangos mostrados en la figura 3.11.

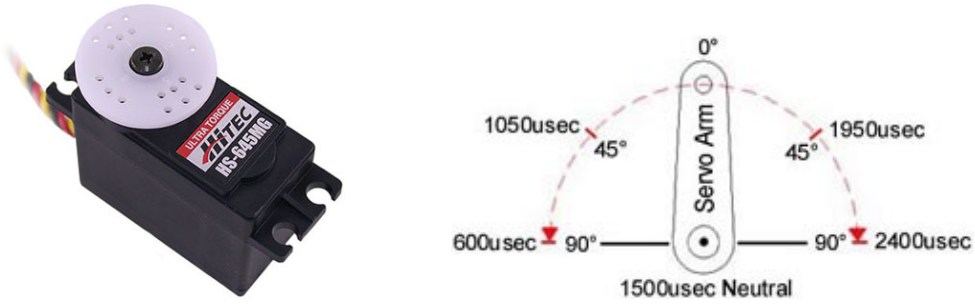


Figura 3.11. Servomotor HITEC HS-645MG. Rango de operación.

Tabla 3.1 Especificaciones del servomotor HITEC HS-645MG

Sistema de control	Control de ancho de pulso. 1500μs neutral
Pulso requerido	3 – 5 V pico a pico señal cuadrada
Voltaje de operación	4.8 – 6V
Temperatura de operación	-20 a 60 °C
Velocidad de operación (@4.0V)	0.24 seg/60° sin carga
Velocidad de operación (@6.0V)	0.20 seg/60° sin carga
Torque (4.8 V)	7.7 kg.cm
Torque (6.0V)	9.6 kg.cm
Ángulo de operación	45° por un pulso de 400μs
Consumo de corriente (4.8V)	88.8mA/en espera y 350mA en operación sin carga
Consumo de corriente (6.0V)	91.1mA/en espera y 450mA en operación sin carga

En pruebas con carga, es decir, la platina de la masa deslizante, trabajando a 6V, el voltaje a emplear en el sistema, el servomotor arrojó los siguientes resultados para consumo de corriente y velocidad de operación:

Tabla 3.2. Resultados de las pruebas con carga para el servomotor.

Velocidad de operación (6V)	1 seg/90°
Consumo de corriente (6V)	510 mA

3.4 Manejo de la posición de las masas con los servomotores

El control de los servomotores se realiza a través del microcontrolador Rabbit 3000 de la computadora de abordo y su tarjeta de desarrollo LP3500. Las salidas de los puertos PWM se encargan de enviar el ancho de pulso requerido de acuerdo a la posición que se desea tener en la masa deslizante, como veremos a continuación.

3.4.1 Sistema piñón-cremallera

El mecanismo piñón-cremallera permite transformar un movimiento circular a uno lineal, utilizando una rueda llamada piñón y una pieza rectilínea llamada cremallera.

Figura 3.12. Mecanismo piñón-cremallera

El piñón es una rueda dentada que gira. La cremallera es una pieza alargada con dientes entre los cuales se encajan los del piñón. Se desplaza linealmente a medida que la rueda gira. La velocidad de la cremallera dependerá del radio del piñón y de su velocidad de giro. Cuanto mayor sea el piñón y más deprisa gire, más rápido se desplazará la cremallera.

$$v = \frac{(N * z)}{n} [cm/minuto]$$

Dónde:
v es la velocidad lineal de la cremallera en [cm/minuto].
N es la velocidad angular del piñón en revoluciones por minuto [r.p.m.]
z es el número de dientes de la rueda dentada.
n es el número de dientes por cm de la cremallera.

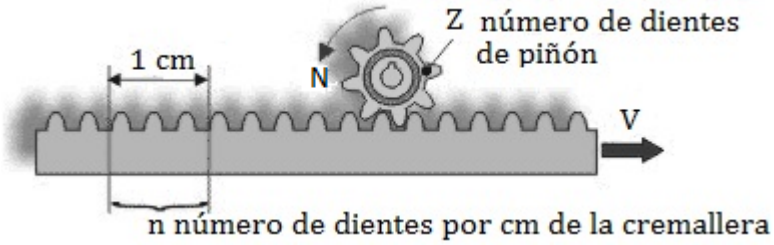


Figura 3.12. Descripción de variables en mecanismo piñón-cremallera

Se decidió colocar un mecanismo de este tipo dentro de cada masa deslizante, en substitución del sistema de tornillo usado en el SIMUSAT 2.0. Frente a otras opciones que también transforman el movimiento circular en lineal –como un sistema manivela biela u otros- se eligió el piñón-cremallera porque es posible introducir todo el mecanismo dentro de la mesa, así no sobresale ningún elemento y sólo se desplaza la platina como ocurre con el sistema anterior. La combinación elegida de piñón–cremallera permite un avance lineal de la platina de 0.72 mm por cada dos grados de giro del servomotor.

3.4.2 Control de las masas por medio del microprocesador

Para funcionar adecuadamente, el servomotor requiere del envío de una señal PWM que cumpla con ciertas características predeterminadas por el fabricante. En este caso, HITEC especifica que sus servomotores requieren de una señal PWM con amplitud de 4.8 a 6 V pico a pico, con una duración de pulso entre 0.6 y 2.5 [ms] y una frecuencia de 50 Hz (20 ms). En este caso se utilizaron de 6 [V], con tres cables, el rojo es para la alimentación, el negro para la conexión a tierra (GND) y el amarillo para la señal de control. El menor ancho de pulso (0.6 ms) marca la posición de 0° y aumenta hasta llegar a la posición de 180° (2.5 ms). Siguiendo esta secuencia de aumento en el ancho de pulso, la dirección de giro es en el sentido de las manecillas del reloj (Manual Servomotores HITEC).

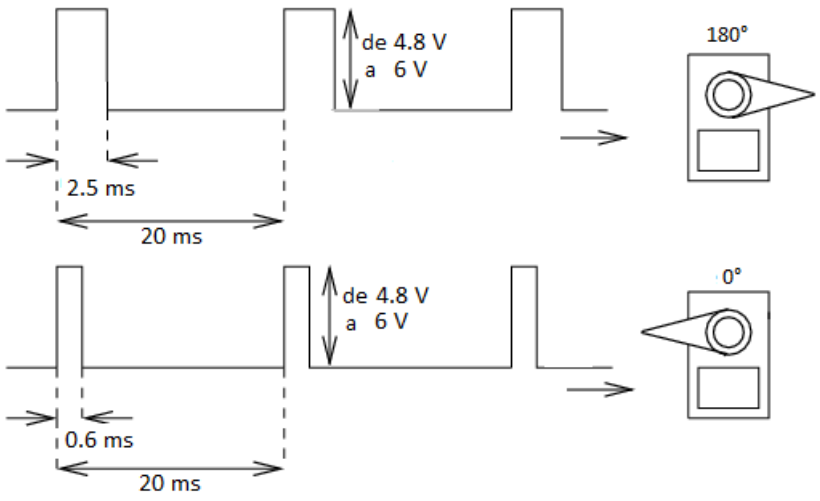


Figura 3.13. Señal requerida por el servo.

3.4.2.1. Generación de la señal PWM en el Rabbit

El microprocesador Rabbit3000 cuenta con tres salidas de señal PWM, controladas por funciones específicas incluidas en sus librerías de programación. En particular se cuenta con las instrucciones “pwmOutConfig”, “pwm_init” y “pwmOut”, que son las más utilizadas para este tipo de aplicaciones. La función “pwmOutConfig” se utiliza conjuntamente con “pwm_init” para fijar la frecuencia de la señal. La manera de declarar una frecuencia de 50Hz es la siguiente:

```
freq = pwmOutConfig(50ul);
pwm_init(freq);
```

La función “pwmOut” permite elegir el puerto PWM que se usará, así como la duración del pulso de la señal. Su estructura es la siguiente:

pwmOut(*Número de salida PWM, Duración del ancho de pulso*);

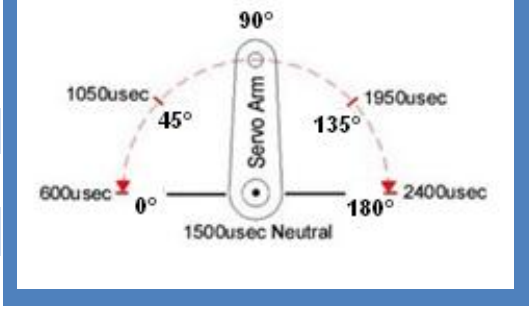
La duración del ancho de pulso se determina de acuerdo a una proporción del valor total del periodo completo de la señal. En este caso, el periodo de la señal es de 20 [ms], lo que equivale al 100%, y en la función pwmOut, al valor de 1. En las especificaciones del servomotor se marca un intervalo de operación de 0.6 a 2.4 ms con 1.5 ms como centro, esto es, del 3 al 12% de la señal PWM con 7.5% como centro. Así, para colocar el servomotor en una posición de 90° se declararía de la siguiente manera:

pwmOut(*Número de salida PWM, 0.075*);

El valor de 0.075 equivale a una duración de ancho de pulso de 1.5 ms, el 7.5% de la duración máxima de la señal. En la siguiente tabla se definen los valores que abarca el movimiento del servomotor de 0 a 180°.

Tabla 3.3. Valores correspondientes a la posición del servomotor.

Ángulo de giro servomotor [°]	Valor para la función pwmOut	Duración de ancho de pulso [us]
0	0.03	600
45	0.0525	1050
90	0.075	1500
135	0.0975	1950
180	0.12	2400



Los incrementos en el valor para la función y la duración del ancho de pulso son proporcionales, con un valor de 10 [μs] por cada grado (0.0005 en incremento para la función del Rabbit). Como se ha mencionado, un giro de 2° en el servomotor, equivale a un desplazamiento de 0.72 [mm] en la posición de la masa deslizante.

Recordando que uno de los objetivos principales de este trabajo de tesis, es el poder compensar los pares gravitacionales presentes en la plataforma de simulación, es necesario llevar a cabo la caracterización del sistema plataforma-masas deslizantes para poder implementar un sistema de seguimiento que compense los movimientos de la plataforma, de tal manera que se eviten las oscilaciones y se esté emulando lo que pasa en condiciones de microgravedad.

Para encontrar las curvas de respuesta, es decir, para saber cual es el comportamiento de la plataforma cuando ocurre un determinado movimiento de las masas deslizantes y que relación tiene este desplazamiento con el ángulo de inclinación de la plataforma, se desarrolló un pequeño programa en C. Esta rutina lleva a cabo incrementos periódicos de 6 grados cada 30 segundos. En el siguiente capítulo se explica este y todos los procesos desarrollados para esta aplicación, los diagramas de flujo y los códigos fuente de las rutinas.

Capítulo

4

Algoritmos de compensación y programación.

En este capítulo se describen los procedimientos desarrollados para efectuar el balanceo automático y para contrarrestar los efectos causados por los pares gravitacionales, sobre la plataforma de simulación. Se pone particular énfasis en el proceso por medio del cual las masas deslizantes ajustan su posición hasta llegar a un punto dónde los pares externos son disminuidos hasta un determinado umbral.

Finalmente se discute el nuevo proceso de graficación implementado para mejorar el despliegue de la orientación del simulador satelital en la pantalla de la estación terrena.

4.1. Balanceo automático

Para llevar el centro de masa de la plataforma de simulación hacia el centro geométrico del balero de aire, se utiliza un esquema como el mostrado en la figura 4.1. Con el nuevo programa de balanceo automático desarrollado de manera específica para manejar las nuevas masas deslizantes (Escobedo L. 2012), se comprobó el correcto funcionamiento del sistema de simulación.

La orientación de los tres ejes de la plataforma son proporcionados por la brújula electrónica y recibidos por el programa de monitoreo inalámbrico y se grafican para evaluar el comportamiento a lo largo del tiempo del ensayo que en ella se lleva a cabo. Los datos enviados por la brújula son guardados en la PC y cada vez que se realiza una prueba, y ésta finaliza de manera exitosa, permanecerán almacenados ahí hasta que se realice otra. Por tanto es necesario salvar los archivos, antes de iniciar otro ensayo. Con dichos datos (manejados en Excel) podemos obtener las gráficas de comportamiento del sistema para cada prueba.

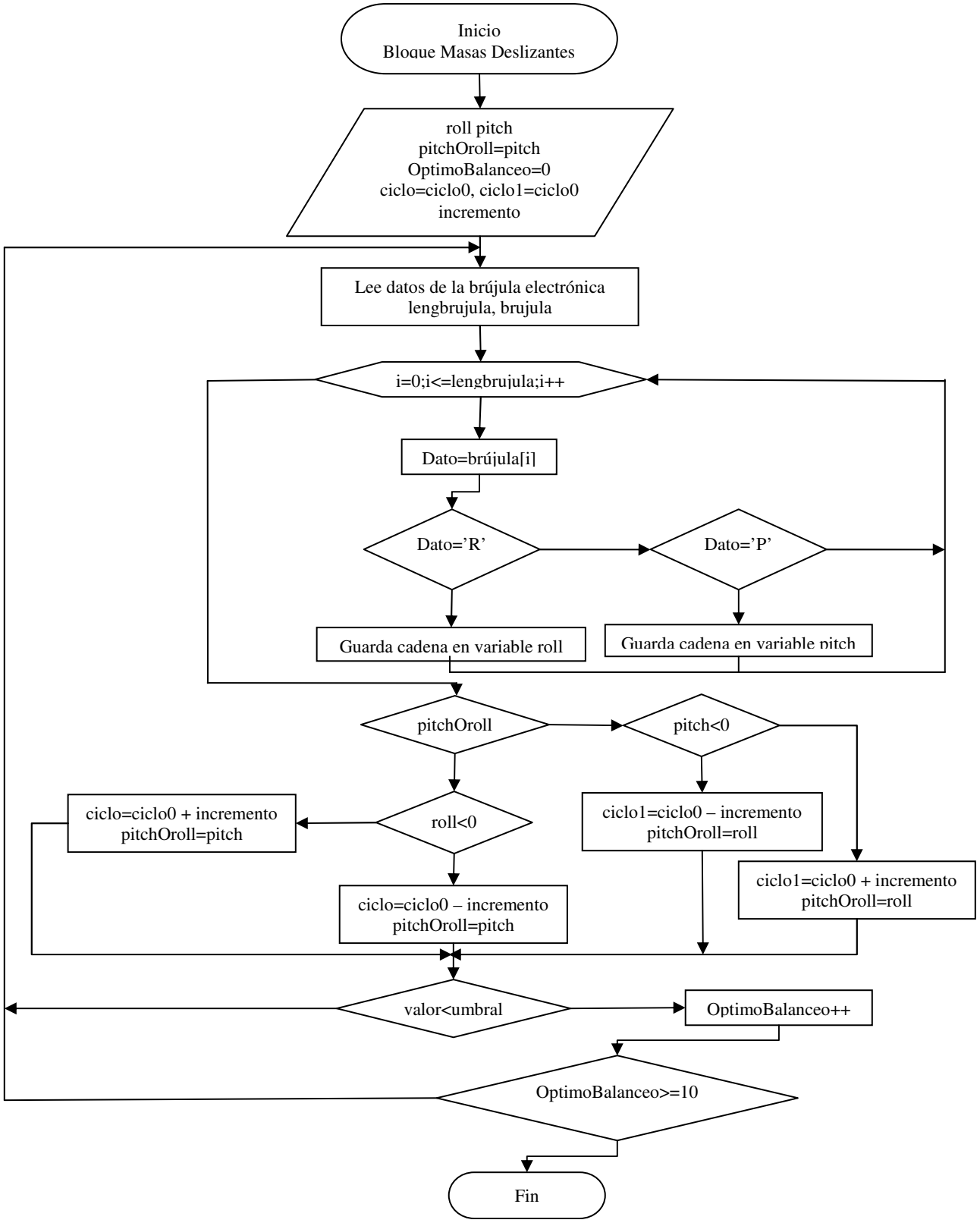


Figura 4.1. Diagrama de flujo utilizado para balancear automáticamente la plataforma.

4.2. Sistema de seguimiento de la posición.

En la figura 4.2 se muestra una gráfica del comportamiento de la plataforma de simulación cuando es ejecutado un algoritmo de orientación en el eje de alabeo (X). La rueda inercial del eje correspondiente es impulsada por el circuito de potencia que a su vez es controlado por el microprocesador Rabbit 3000. Como se puede observar, la orientación inicial es de -10º y después de 22 segundos, aproximadamente, la plataforma reacciona ante la operación del actuador y es llevada hasta la posición de referencia establecida, que en este caso es 0º. El umbral de la señal de control es de ± 1º, lo que se observa en la gráfica es que una vez estando la plataforma dentro del umbral el algoritmo deja de ejecutarse, sin embargo, por la acción de los pares gravitacionales, la posición de la plataforma tiende a regresa a su posición original, generando una oscilación.

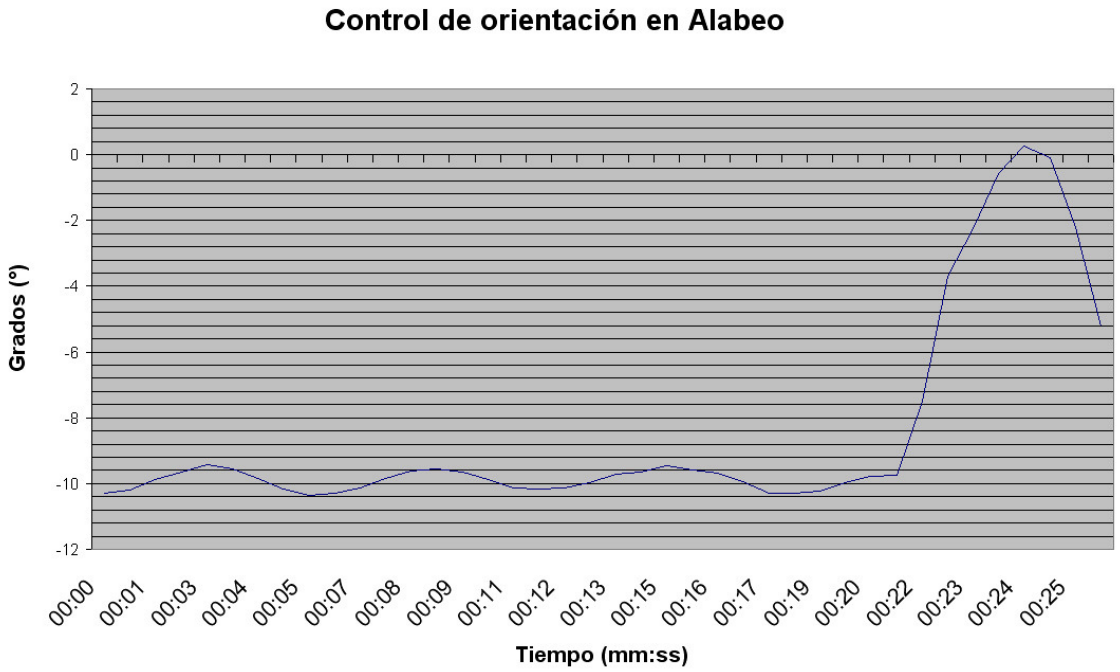


Figura 4.2. Control de orientación con una rueda inercial actuando sobre el eje de alabeo.

En la figura 4.3. se presenta otro ejemplo típico del comportamiento de la plataforma de simulación ante la acción de la rueda inercial colocada sobre el eje de cabeceo (Y). Nuevamente, una vez que se alcanza la posición deseada y la plataforma se encuentra dentro del umbral preestablecido, el algoritmo da por terminada su función y envía un mensaje de que el experimento ha terminado. No obstante, los pares gravitacionales regresan la plataforma a su posición original.

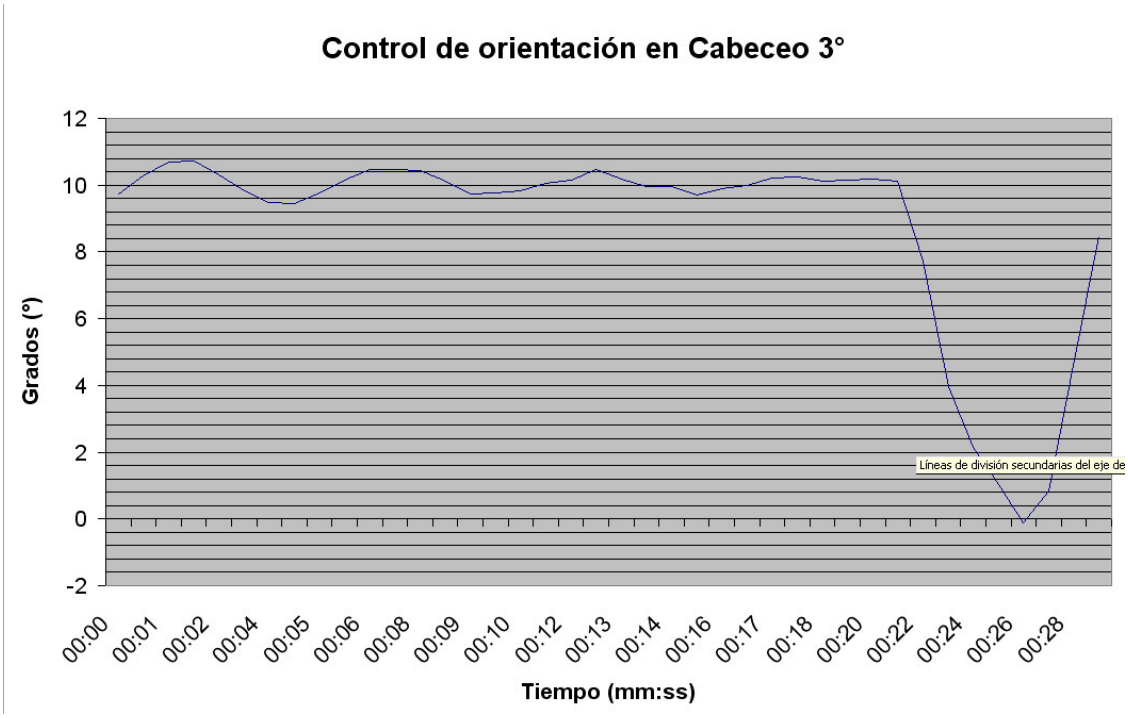


Figura 4.3. Control de orientación con una rueda inercial actuando sobre el eje de cabeceo.

Con la finalidad de contrarrestar estos efectos, se implementó un algoritmo que permite mantener la posición de la plataforma una vez alcanzada la posición de apuntamiento deseada. Todo el proceso se describe en las siguientes secciones.

4.3. Sistema de seguimiento de la posición.

En esta sección se explica detalladamente todo el proceso de calibración de la plataforma, para determinar cuales son los desplazamientos necesarios de las masas deslizantes para disminuir los efectos de los pares gravitacionales.

4.3.1. Curvas de respuesta. Caracterización PWM % vs. Desplazamiento de la platina.

Previo al proceso de obtención de las curvas de porcentaje de PWM vs. Ángulo de inclinación de la plataforma, se llevó a cabo un paso previo, para asegurar que el movimiento de las platinas era el adecuado, tomando en cuenta los desplazamientos de la flecha del servomecanismo. Esta prueba fue “estática”, es decir, sin introducir aire a presión en el cojinete neumático esférico. De esta manera se obtuvieron las curvas que se presentan en las figuras 4.5 y 4.6. En este punto, las platinas se encontraban en el centro geométrico de cada masa deslizante, es decir, a la mitad de su desplazamiento, con los servomotores en la posición de 90° (50% de valor de señal PWM). Se realizaron dos pruebas para cada masa, considerando un movimiento como

incremento cuando se desplaza desde su posición inicial, hacia el centro de la plataforma, y decremento cuando se aleja de ésta. En la figura 4.4 se muestra la localización de las masas deslizantes en los ejes X, Y, sus posiciones iniciales y los sentidos de movimiento considerados.

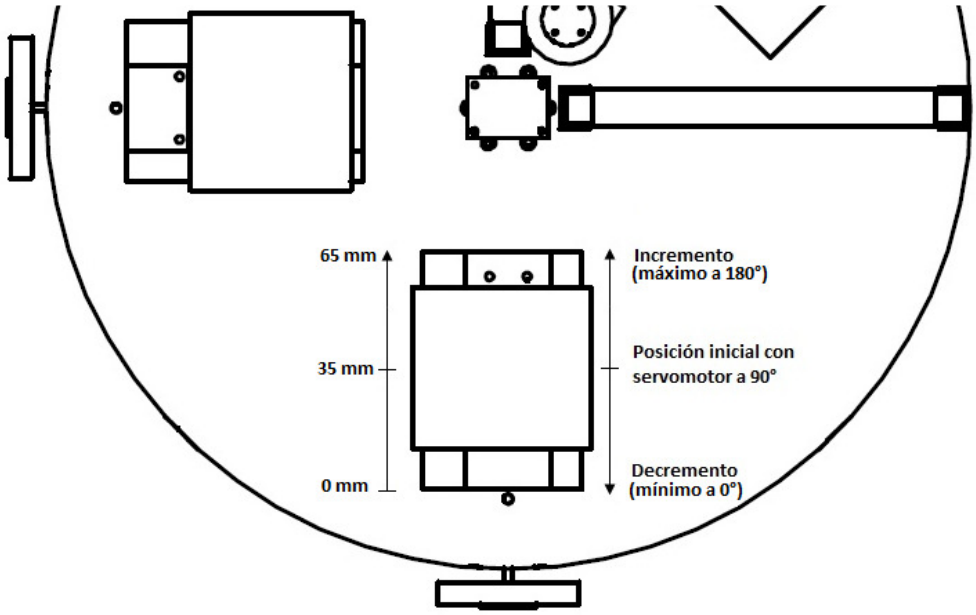


Figura 4.4. Posiciones consideradas para caracterizar el comportamiento las masas deslizantes.

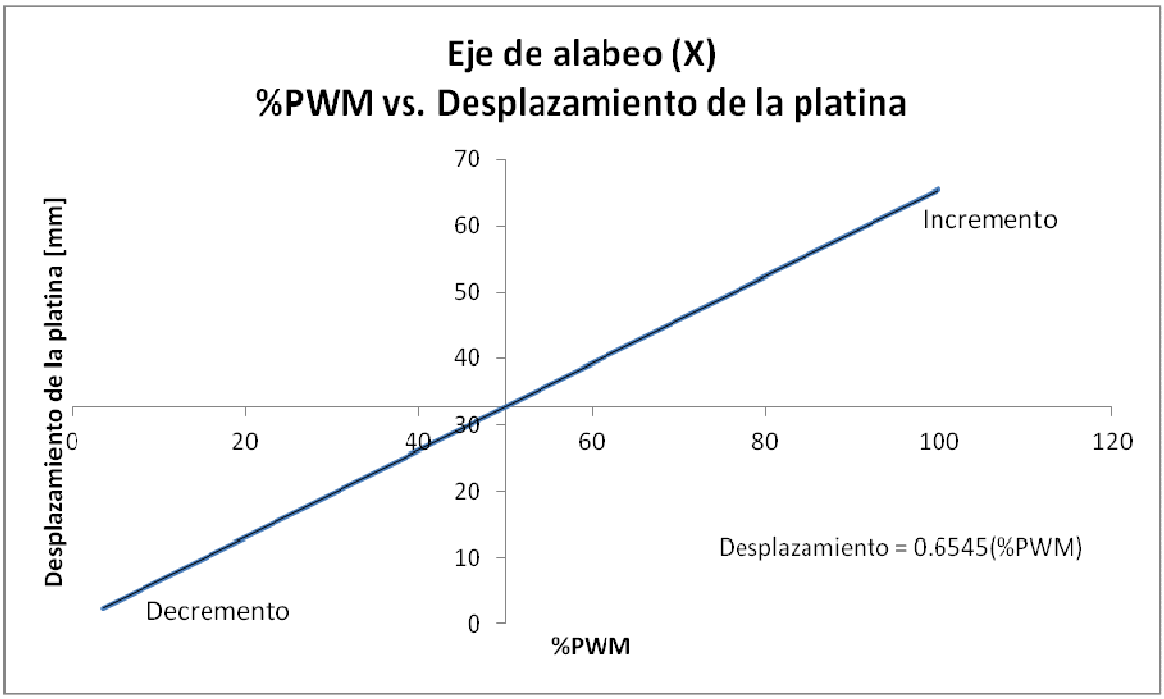


Figura 4.4. Prueba de desplazamiento de la platina, al cambiar el porcentaje de PWM enviado al servomotor. Eje X.

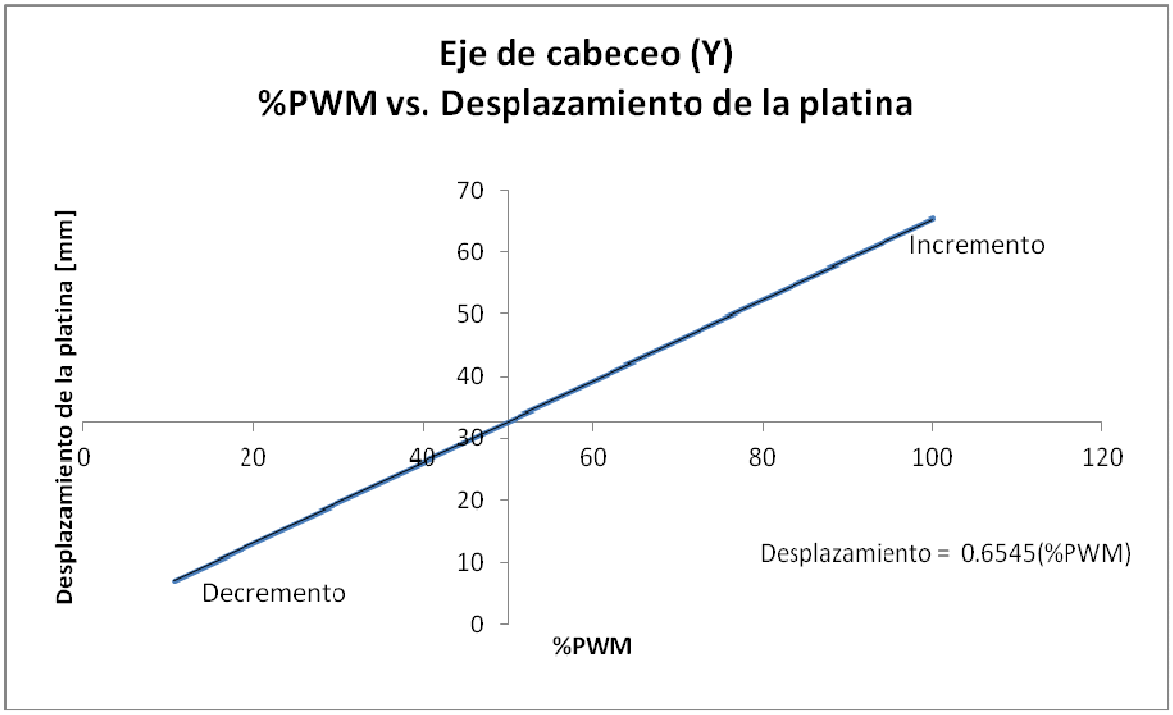


Figura 4.5. Prueba de desplazamiento de la platina, al cambiar el porcentaje de PWM enviado al servomotor. Eje Y.

En estas curvas se ve claramente que la respuesta de los desplazamientos de las platinas presentan un comportamiento lineal, con respecto a las señales de PWM enviadas hacia los servomotores. Con esto también se comprueba que las platinas se desplazan toda la carrera prevista, logrando un ángulo de corrección máximo, pero sin salirse de sus posiciones, evitando así un problema con el funcionamiento del sistema.

4.3.2. Curvas de respuesta. Desplazamiento de la platina vs. Ángulo de la plataforma.

Una vez que las masas fueron probadas y caracterizado su movimiento, con la relación mostrada en las gráficas de la sección anterior, se procedió a obtener las curvas de desplazamiento de la platina vs el ángulo de inclinación de la plataforma. Las mediciones fueron hechas ahora si con la plataforma flotando sobre el balero de aire, en un medio con fricción despreciable, desde un equilibrio con la plataforma balanceada en los dos ejes (X, Y) con un umbral de 1° con respecto a la horizontal. En este punto del experimento, las platinas se encontraban también en el centro geométrico de cada masa deslizante, es decir, a la mitad de su desplazamiento y con los servomotores en 90°.

La importancia de esta prueba radica en que a partir de este resultado, se podrá saber qué desplazamiento de la masa deslizante corresponde a qué ángulo de inclinación de la plataforma y así poder compensar los efectos oscilantes causados por los pares gravitacionales externos.

Básicamente se usó el mismo programa para obtener las curvas de la sección anterior, solamente que en este caso, al efectuar los incrementos en el porcentaje de PWM, un retraso aplicado de 30 segundos, entre cada paso, fue indispensable para que el sistema dejara de oscilar y se estabilizara, permitiendo tomar un dato más cercano a la realidad. En las figuras 4.6 a y b, se muestra el diagrama de flujo de este proceso , en el apéndice A se encuentra el programa fuente.

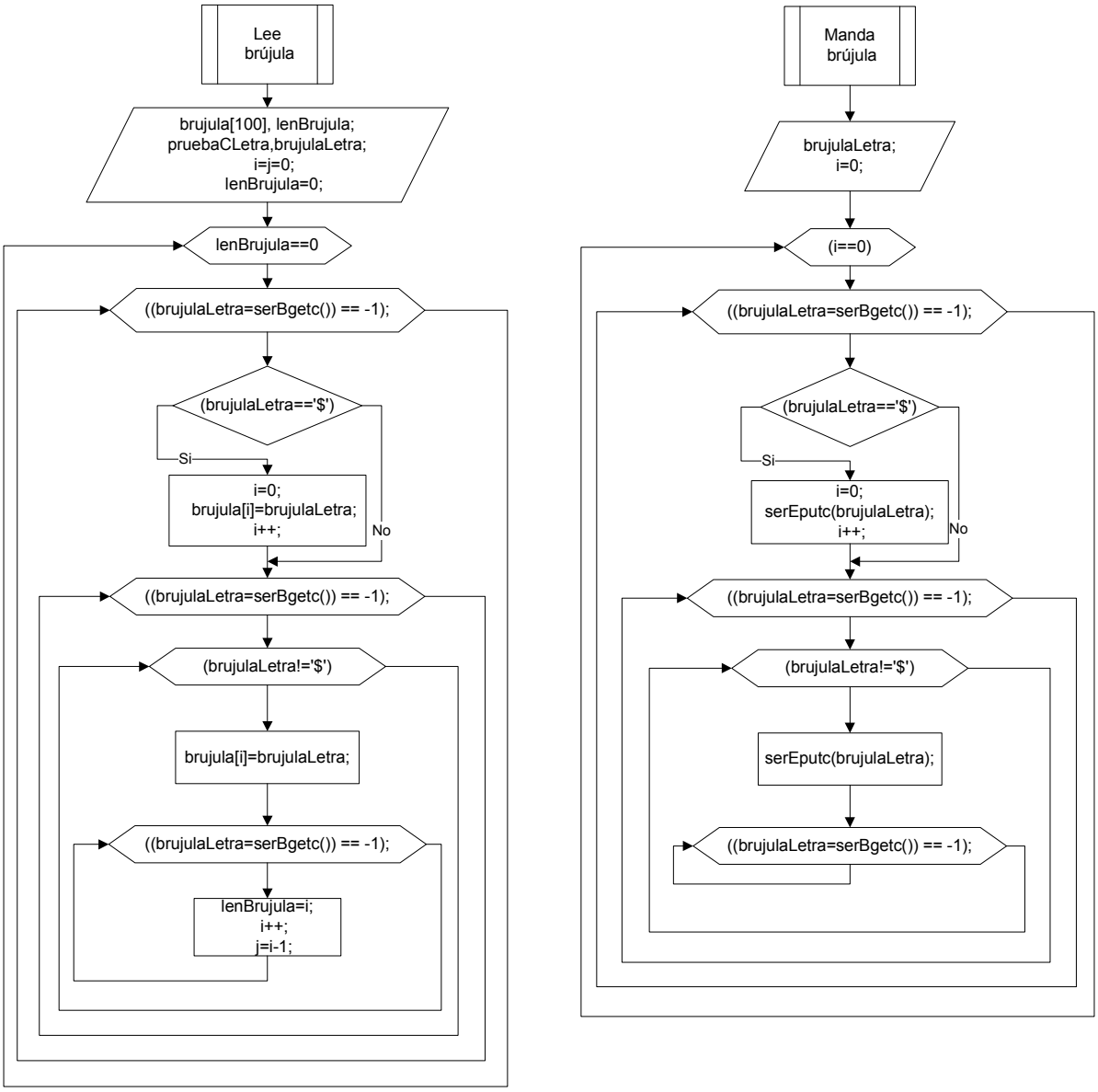


Figura 4.6a. Subrutinas “Lee brújula” y “Manda brújula” utilizadas en la rutina principal. Rutinas previamente elaboradas [Huante] para leer datos desde el compás electrónico y enviarlos a la computadora de abordo. Empleadas en cualquier proceso que requiera adquisición de datos desde el compás electrónico

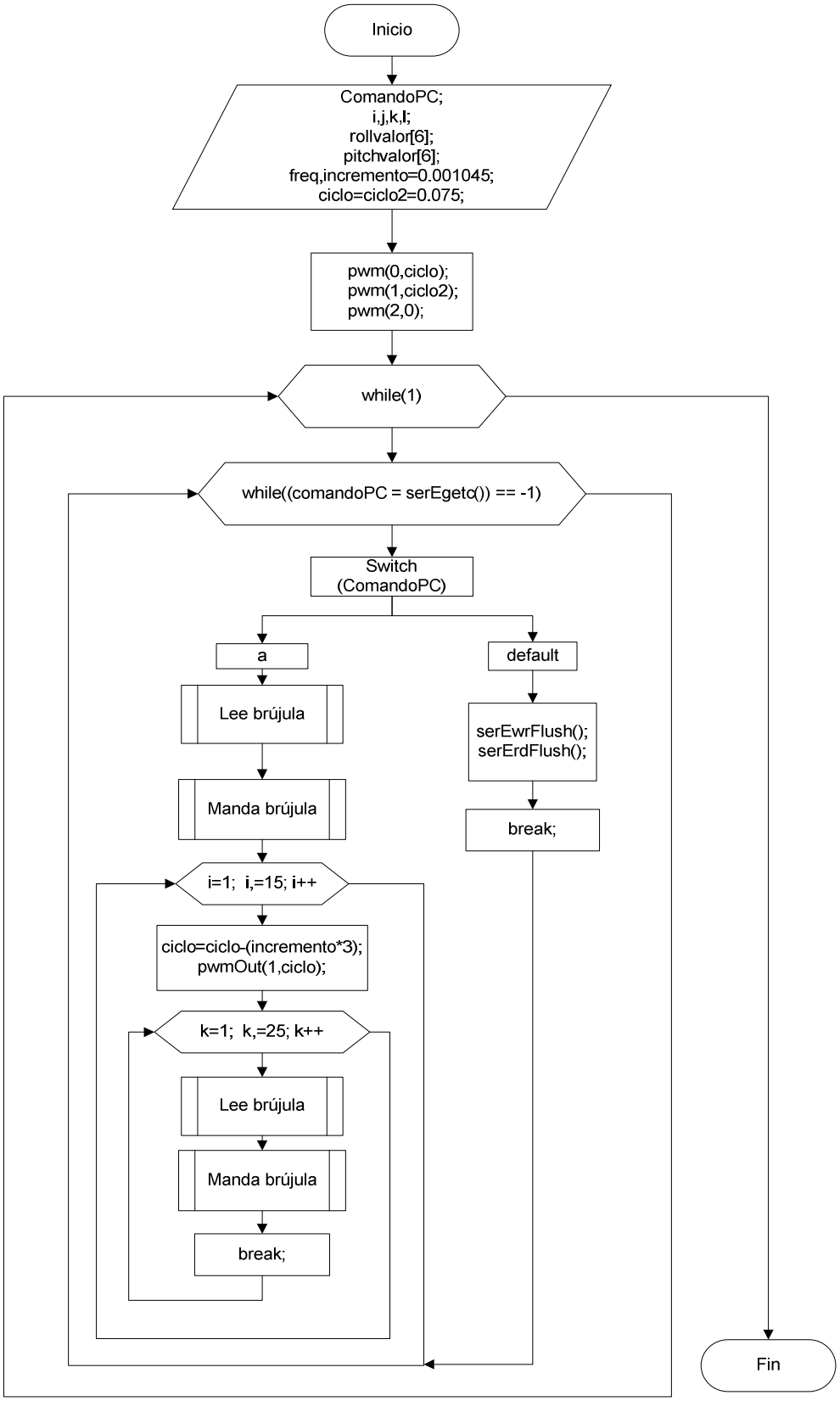


Figura 4.6b. Diagrama de flujo del proceso de calibración de la plataforma. Con esta rutina se encuentra la relación entre el porcentaje de PWM de los servos y el ángulo de inclinación de la plataforma.

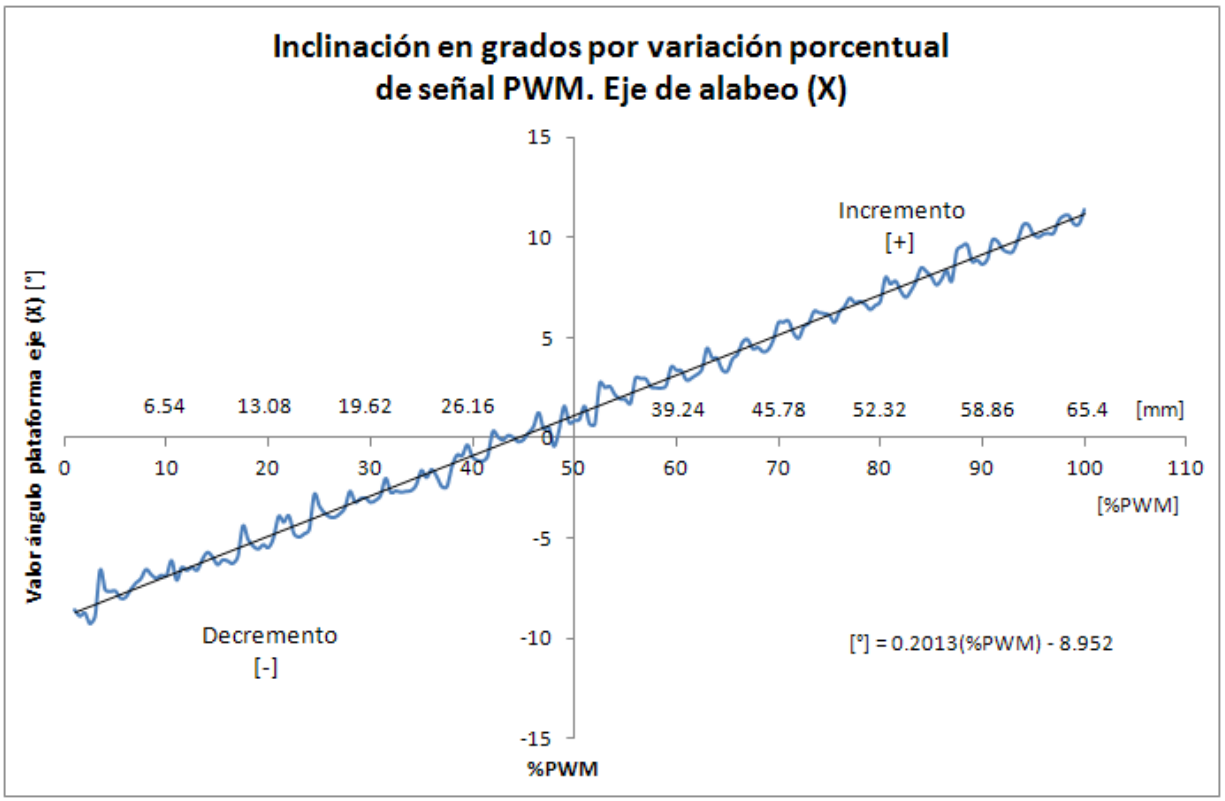


Figura 4.6. Curvas de respuesta % de PWM vs. Ángulo de inclinación de la plataforma en el eje X.

En la figura 4.7 se muestra en azul la curva original obtenida durante la prueba de calibración y en negro, un ajuste a una recta. Esto se hizo, dada la naturaleza oscilatoria de la plataforma y a que su tendencia lineal es muy clara. Este mismo procedimiento se aplicó en la prueba de calibración del eje Y. En ambos casos se obtuvo la ecuación de la recta, que servirá para el algoritmo de compensación, como se explica más adelante.

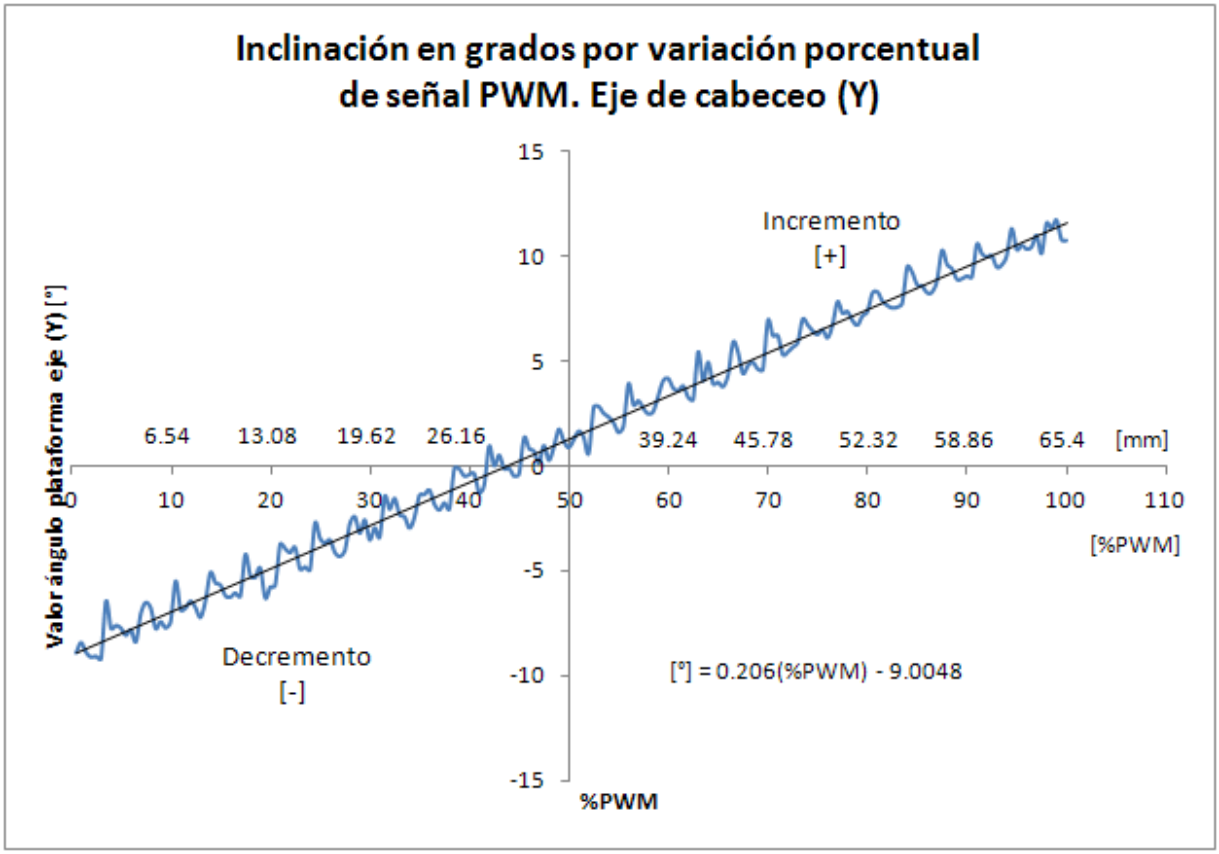


Figura 4.7. Curvas de respuesta % de PWM vs. Ángulo de inclinación de la plataforma en el eje Y.

Lo que se puede observar en las dos gráficas de %PWM vs. ángulo de la plataforma, es que tienen casi la misma pendiente y una diferencia pequeña en el valor de la ordenada al origen. Se demuestra que existe una tendencia lineal en el movimiento, alterado por las oscilaciones de la plataforma que, en menor escala, son generadas por los mismos pares gravitacionales que se pretenden anular.

4.3.3. Compensación de los pares gravitacionales en la plataforma de simulación.

Lo que se presenta enseguida son las pruebas de compensación de los pares gravitacionales actuando sobre la plataforma de simulación. En las figuras 4.9 a,b,y c, se muestra el diagrama de flujo de este procedimiento. La gráfica de la figura 4.10 representa un movimiento de la plataforma, realizado en forma manual, dónde se le cambia de posición 10 grados en el eje X y luego se suelta, lo que se observa es el movimiento oscilatorio natural del sistema, esta curva tiene la etiqueta de: *sin compensar*. La otra curva mostrada en esta misma gráfica, es el efecto de la compensación de la plataforma con el movimiento de la masa deslizante ubicado sobre este mismo eje. Se observa que una vez que se cambia la posición y la plataforma se

suelta, esto ocurre aproximadamente a los 2 segundos, el sistema compensa esta desviación y mantiene la plataforma en su posición.

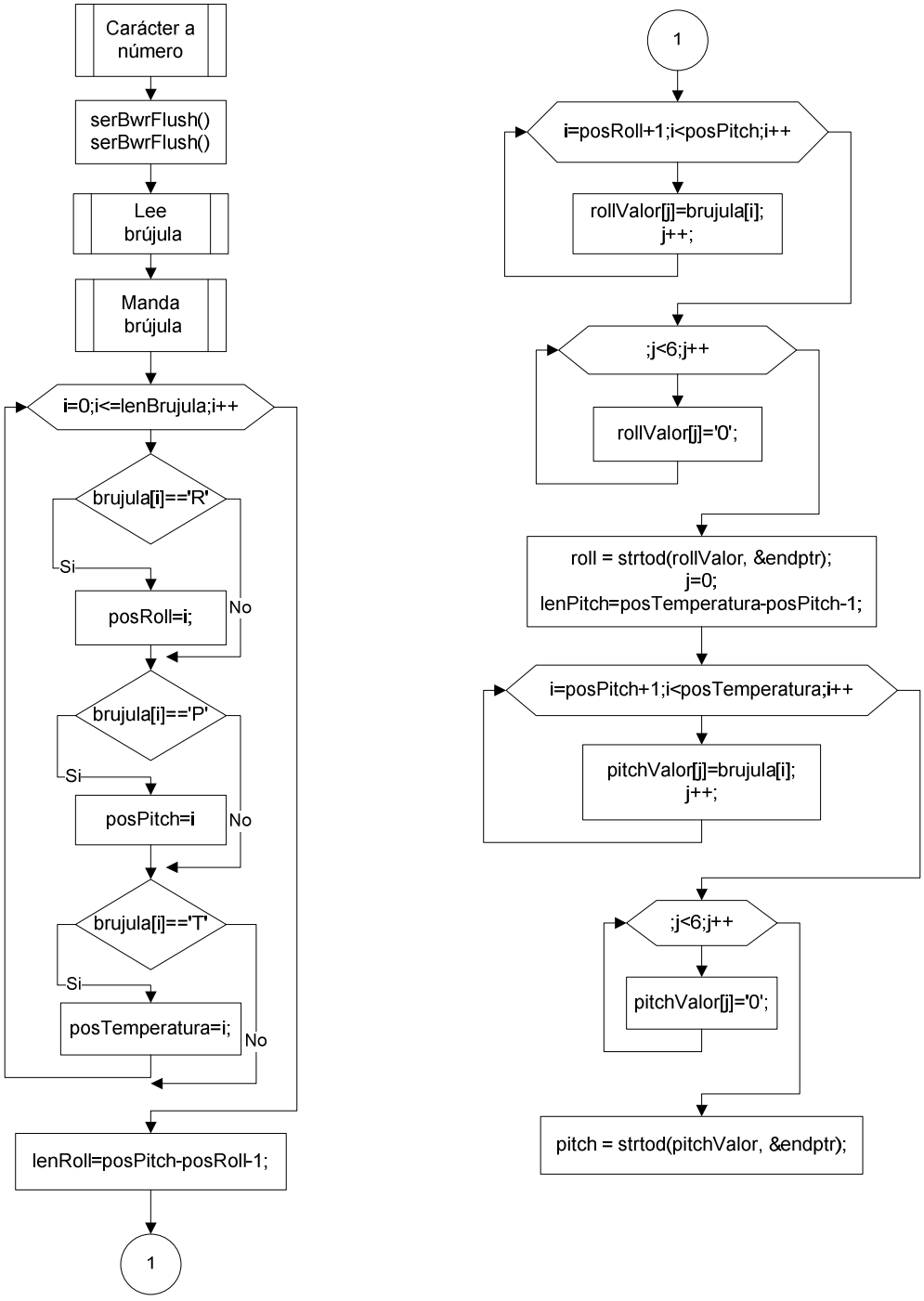


Figura 4.9a. Subrutina "Convierte carácter a número" utilizada en la rutina principal. Rutina previamente elaborada [Huante] para leer datos desde el compás electrónico y enviarlos a la computadora de abordo. Empleadas en los programas de seguimiento de la posición y el nuevo balanceo de la plataforma.

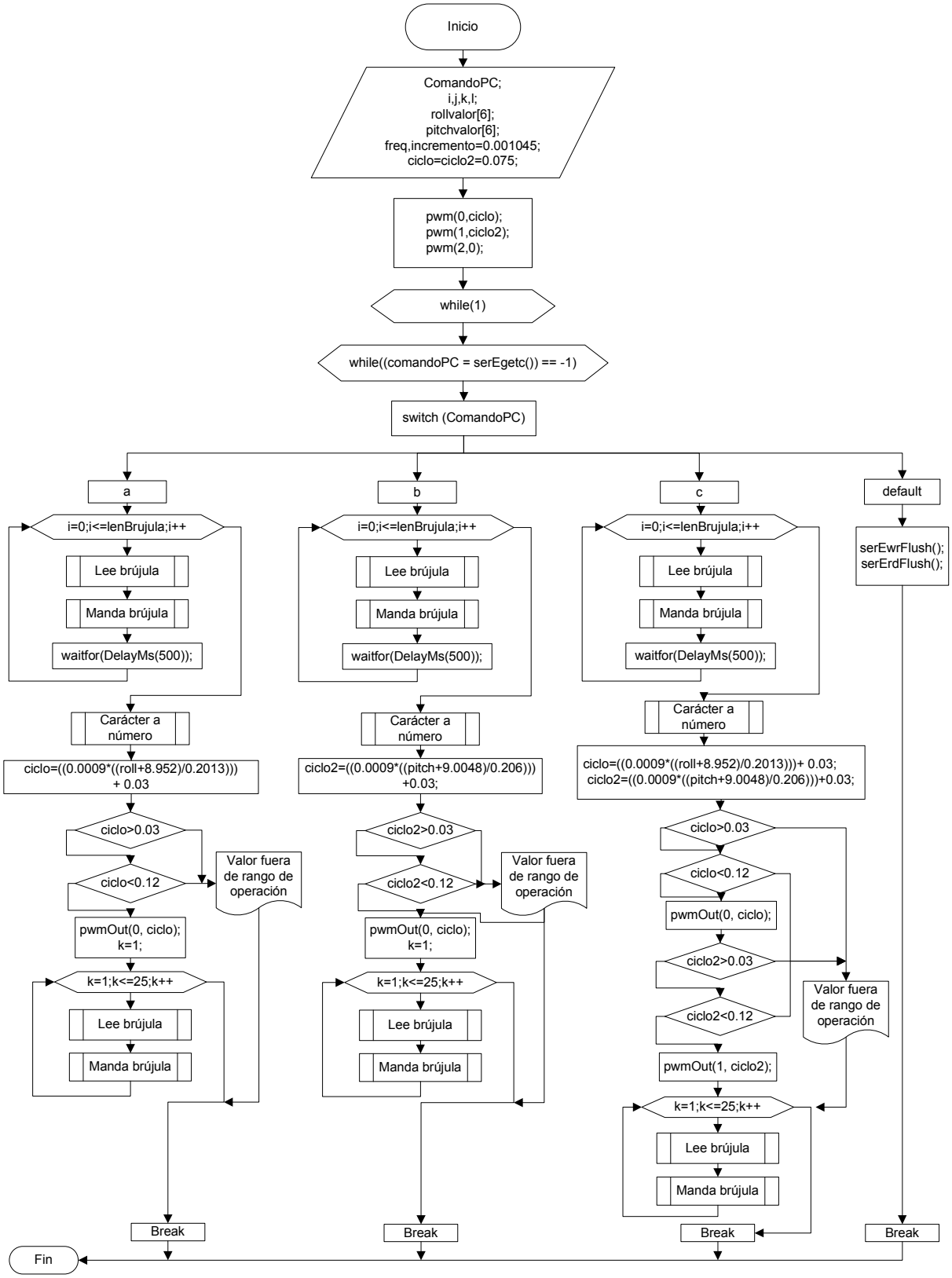


Figura 4.9b. Diagrama de flujo de la rutina que efectúa el seguimiento servomecánico del movimiento de la plataforma de simulación.

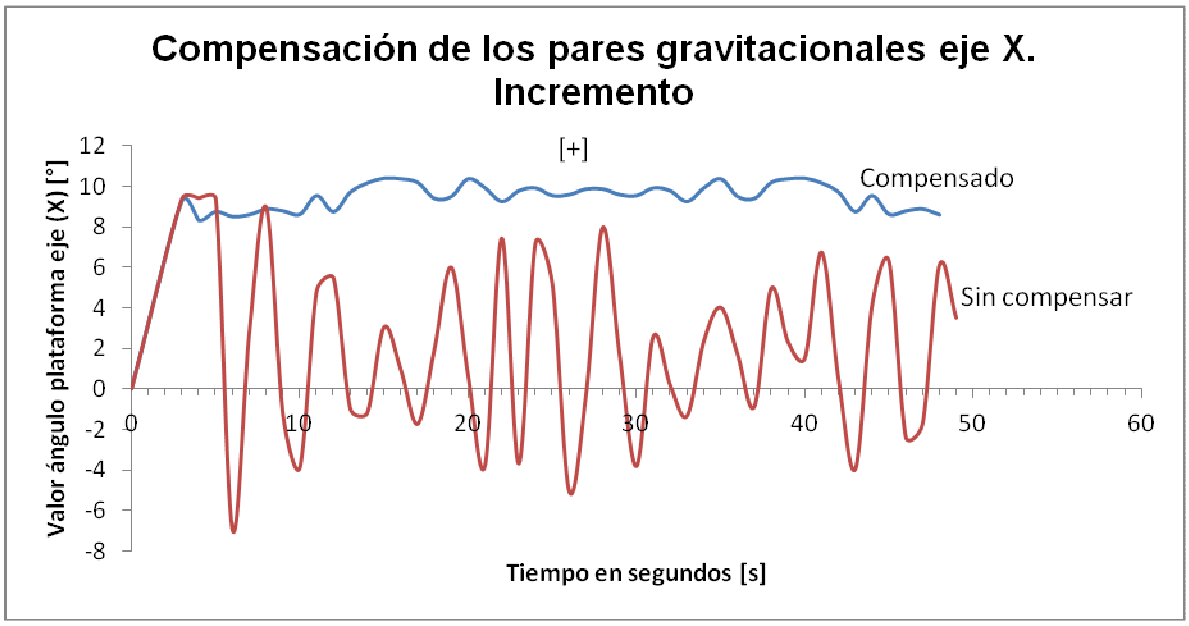


Figura 4.10. Compensación de los pares gravitacionales en el eje X, en el sentido positivo.

La gráfica de la figura 4.11, nos muestra el comportamiento de la plataforma ahora causando una desviación de 10 grados en el sentido negativo del eje X. El resultado es muy similar al obtenido en la parte positiva del mismo eje.

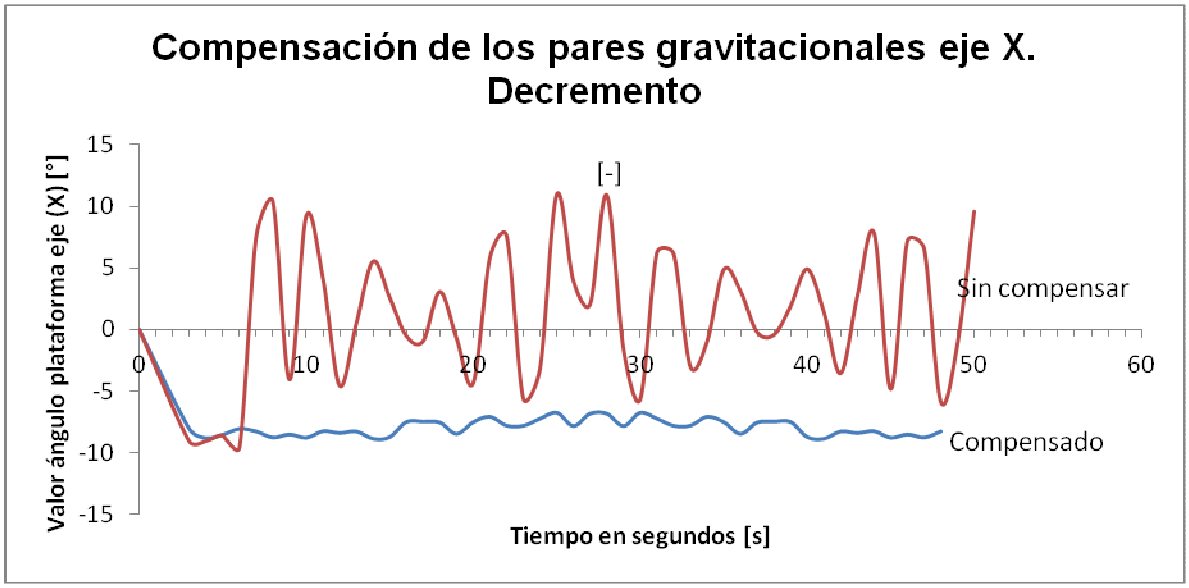


Figura 4.11. Compensación de los pares gravitacionales en el eje X, en el sentido negativo.

Estos mismos experimentos se efectuaron para el eje Y, con resultados similares, como lo muestran las gráficas de las figuras 4.12 y 4.13.

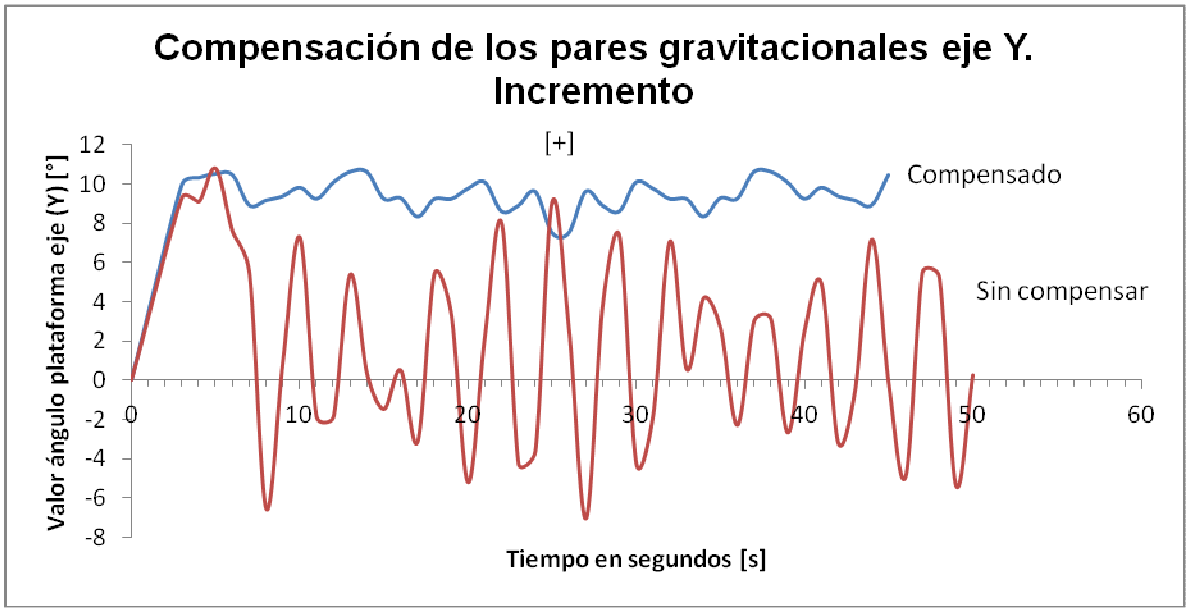


Figura 4.12. Compensación de los pares gravitacionales en el eje Y, en el sentido positivo.

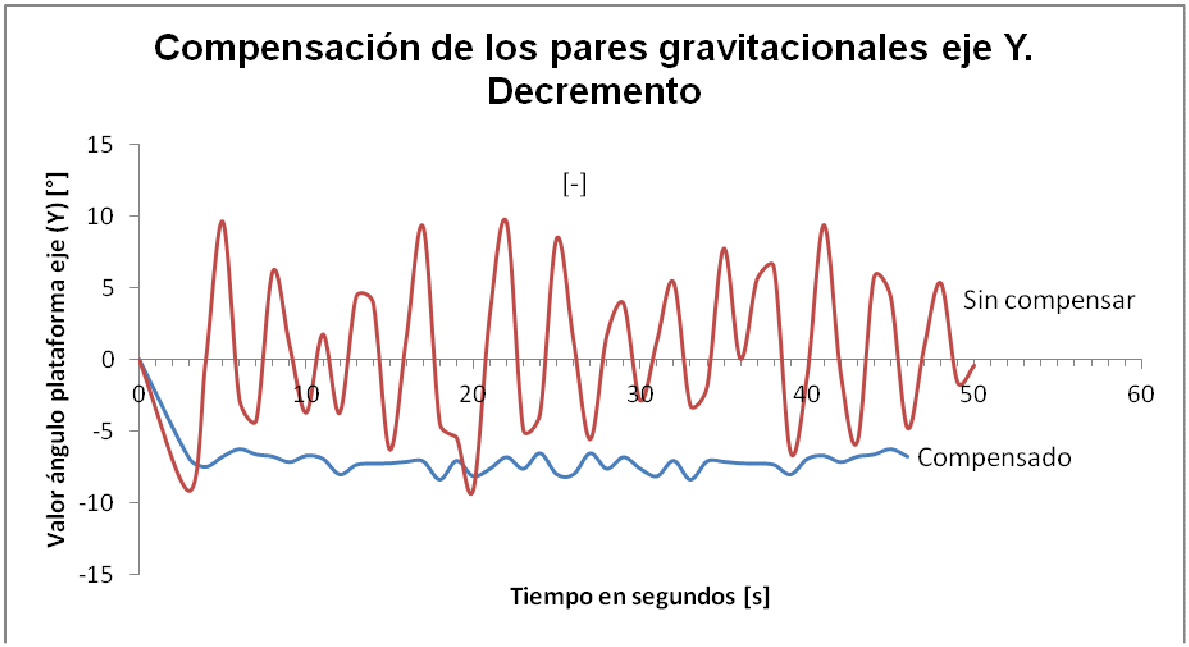


Figura 4.13. Compensación de los pares gravitacionales en el eje Y, en el sentido negativo.

También se llevaron a cabo pruebas de compensación en los dos ejes de manera simultánea, cambiando la posición de la plataforma, sacándola de su posición de equilibrio con respecto a la horizontal. El resultado puede verse en la figura 4.14, dónde se muestran dos ensayos efectuados con este sistema, demostrando que la compensación se efectúa adecuadamente en valores positivos y negativos de rotación de los dos ejes.

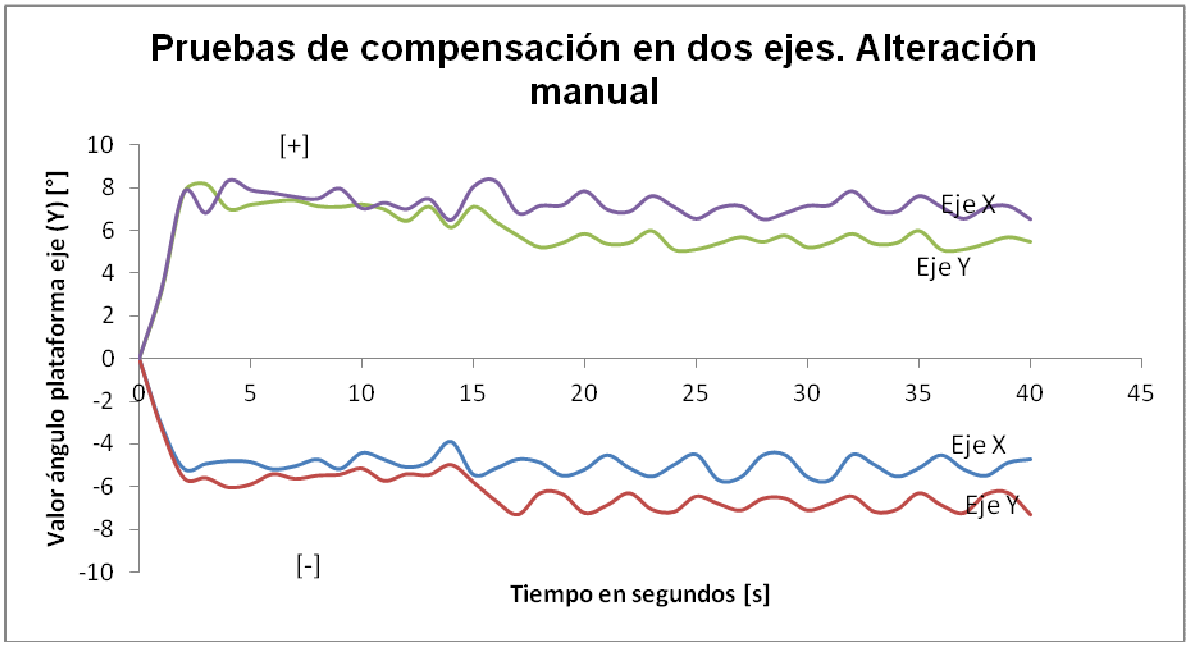


Figura 4.12. Pruebas de compensación en dos ejes, efectuando una alteración de la orientación en forma manual.

Estas son las primeras pruebas de funcionamiento de este prototipo, los algoritmos de compensación han demostardo su utilidad para reducir significativamente el efecto de la gravedad en la plataforma de simulación.

Con esto se ha corroborado que la propuesta es funcional y ha sido implementada en la plataforma sin problemas.

4.3.4. Nuevo método de balanceo automático.

Una vez que la plataforma ha sido calibrada, se ha implementado una nueva manera de balancearla. Ahora como ya se conoce la relación entre el ángulo de desviación que presenta y el porcentaje de PWM del servo, entonces se lee a través de la brújula electrónica dicha desviación y se envían el comando para compensarla, pero por pequeños incrementos, de manera pausada, para evitar que oscile demasiado. El

diagrama de flujo de este procedimiento se presenta en las figuras 4.15 a y b, mientras que el programa fuente aparece en el apéndice A.

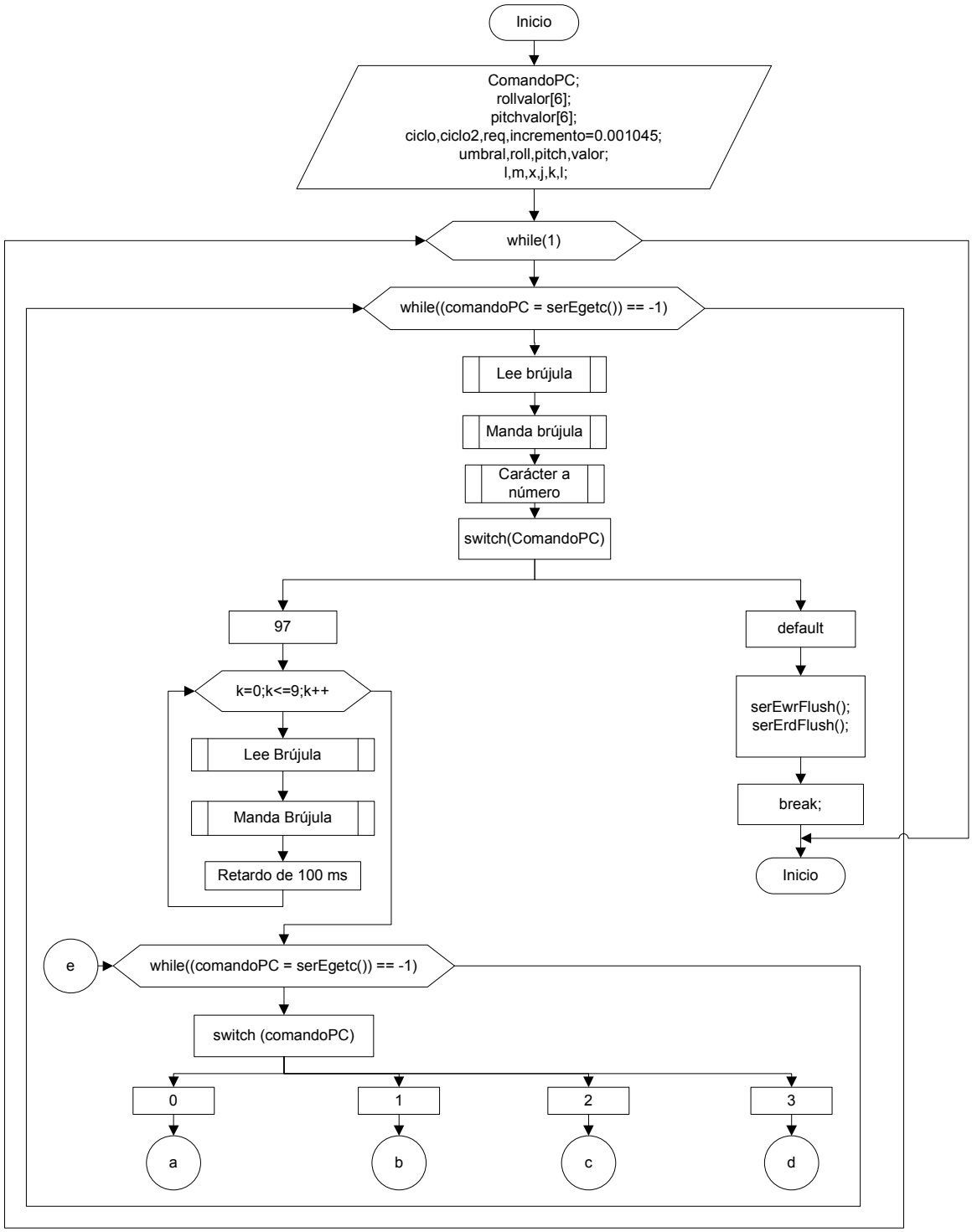


Figura 4.15a. Diagrama de flujo del procedimiento de balanceo automático implementado a partir de que se tienen los datos de calibración de la plataforma. Primera parte del algoritmo implementado.

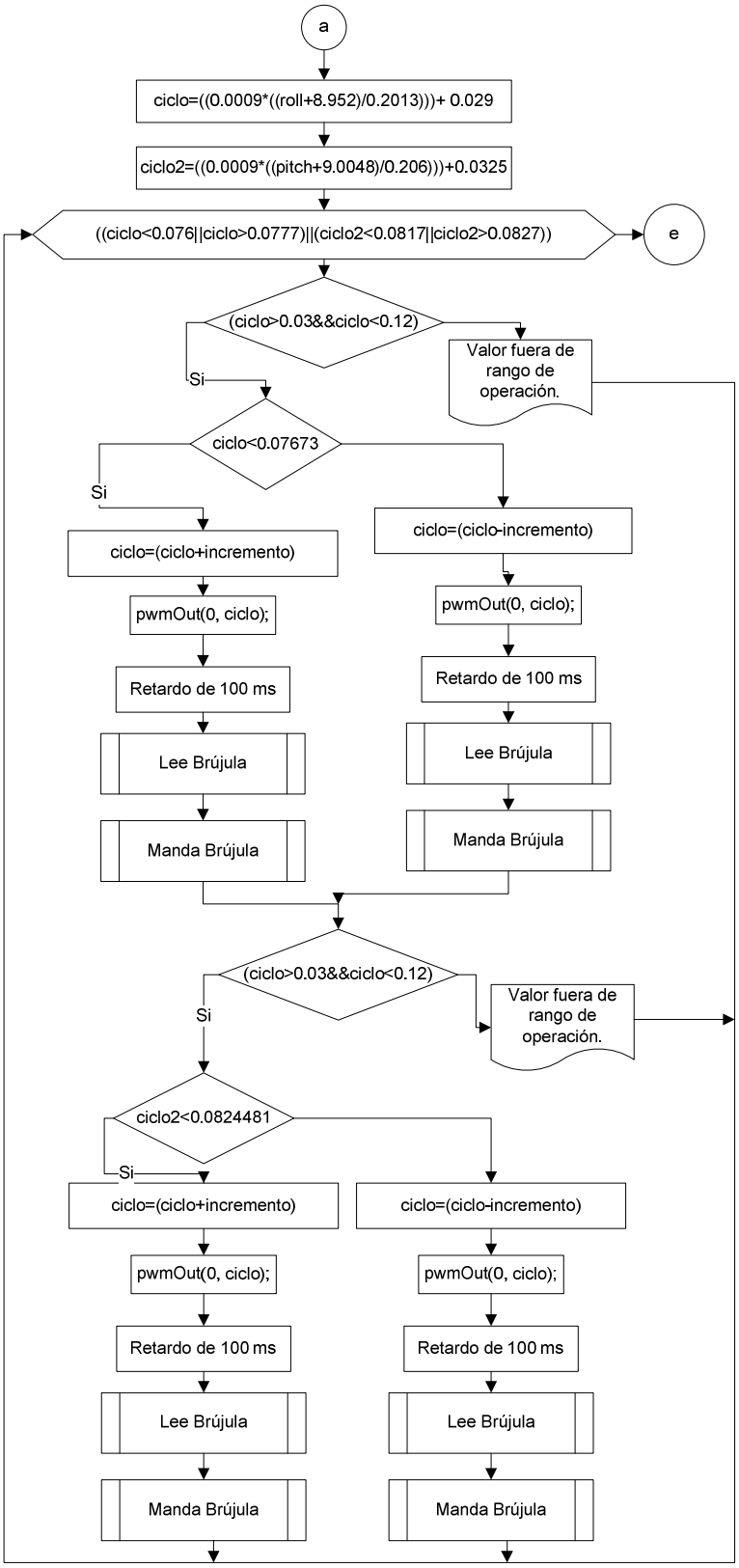


Figura 4.15b. Segunda parte del diagrama de flujo del procedimiento de balanceo automático implementado.

Se omitieron los casos b, c y d en el diagrama de flujo porque son muy parecidos al caso a, sólo cambian los valores de calibración para las variables “ciclo” y “ciclo2”.

En la figuras 4.16, 4.17, 4.18 y 4.19 se muestran las pruebas balanceo realizadas para el nuevo método de balanceo.

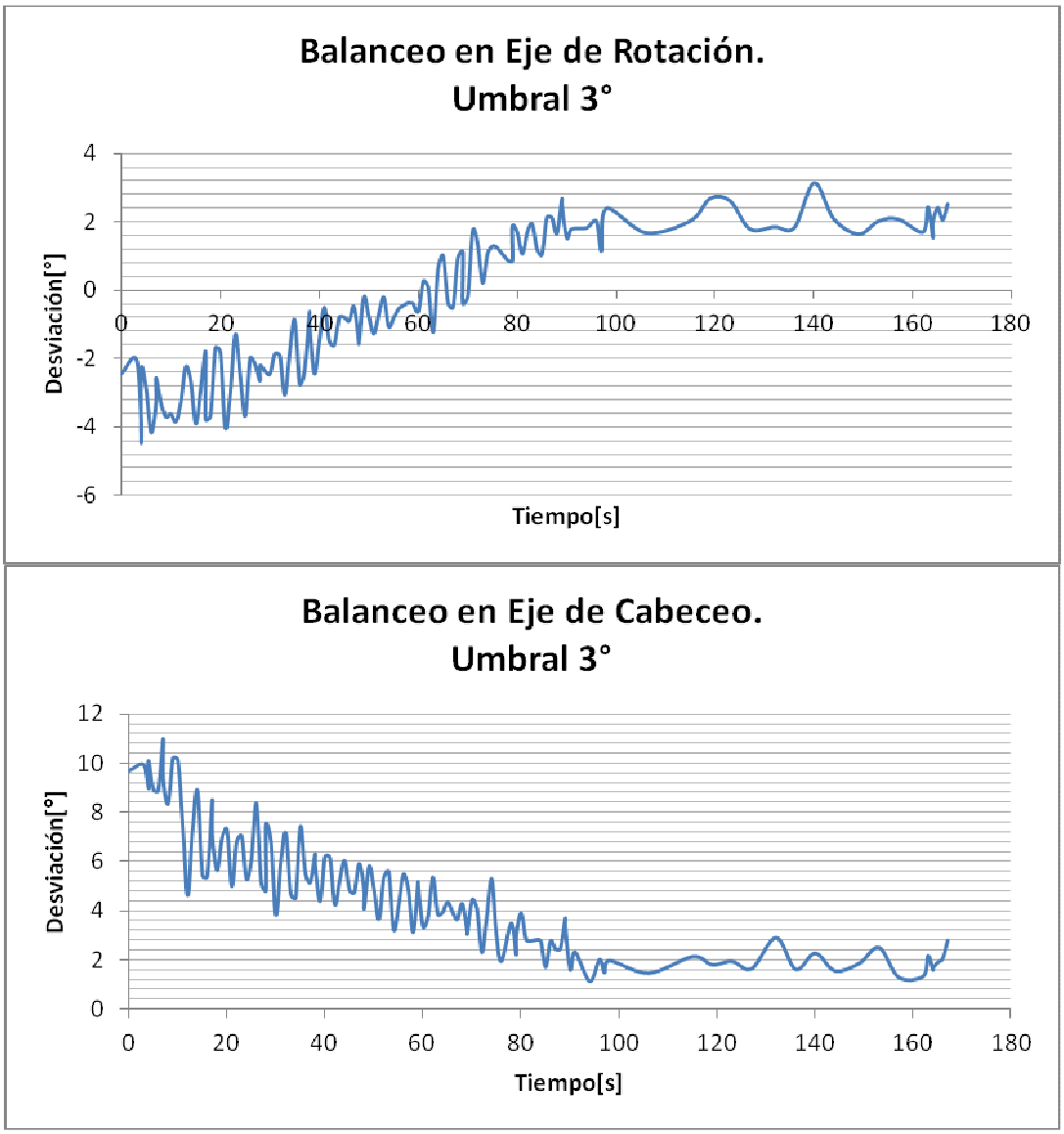


Figura 4.16. Pruebas de balanceo automático con un umbral de 3°

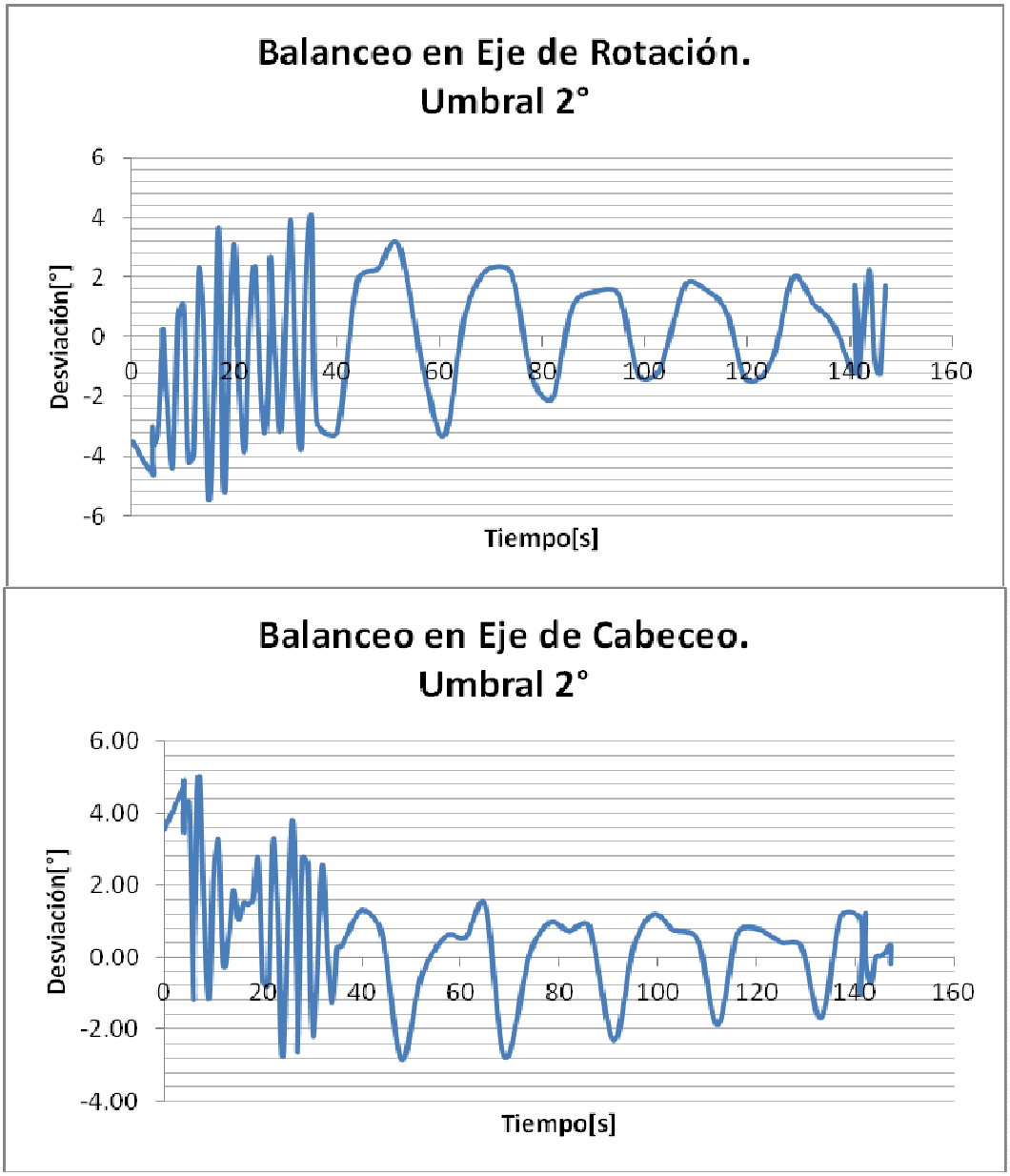


Figura 4.17. Pruebas de balanceo automático con un umbral de 2°

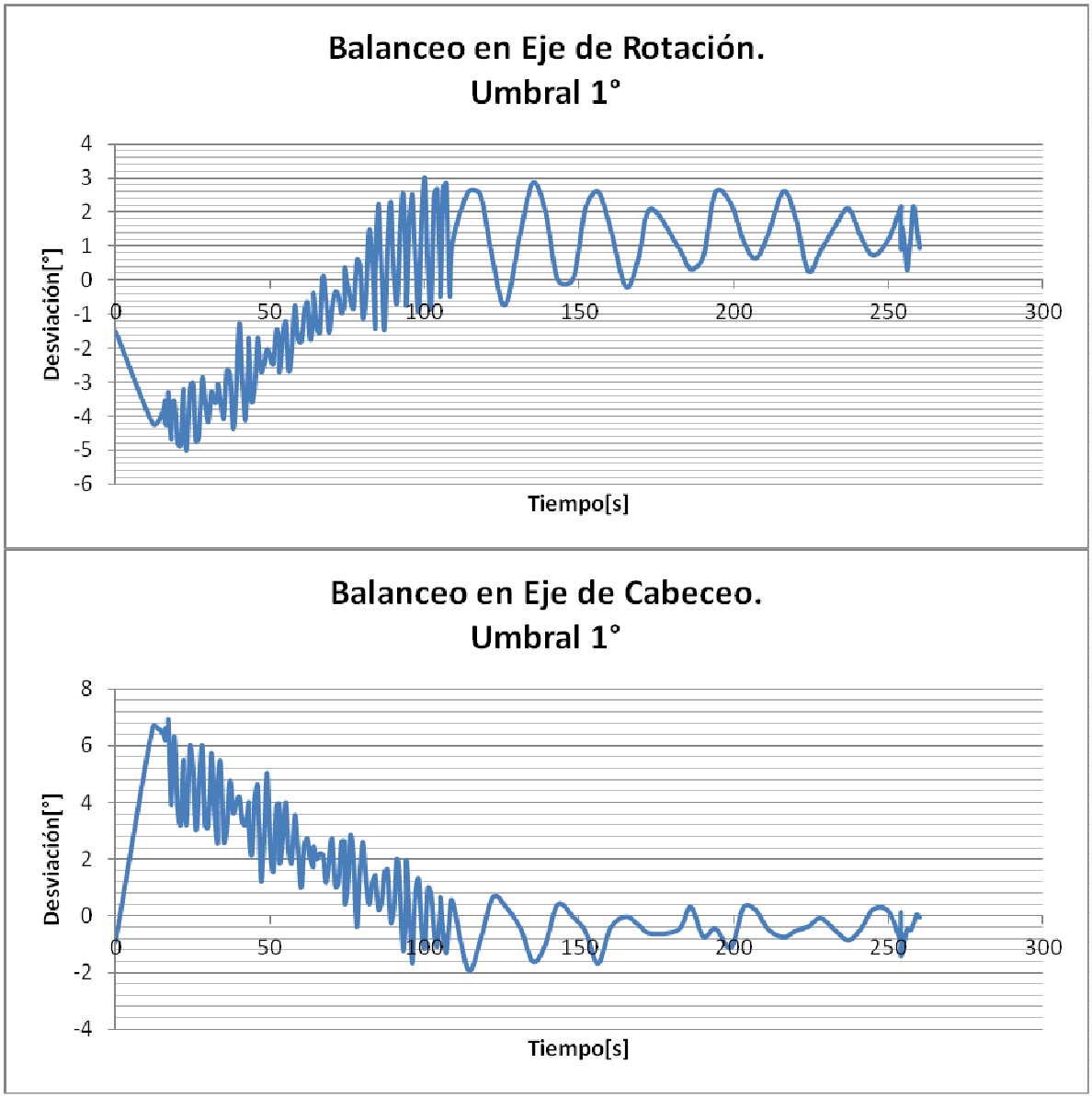


Figura 4.18. Pruebas de balanceo automático con un umbral de 1°

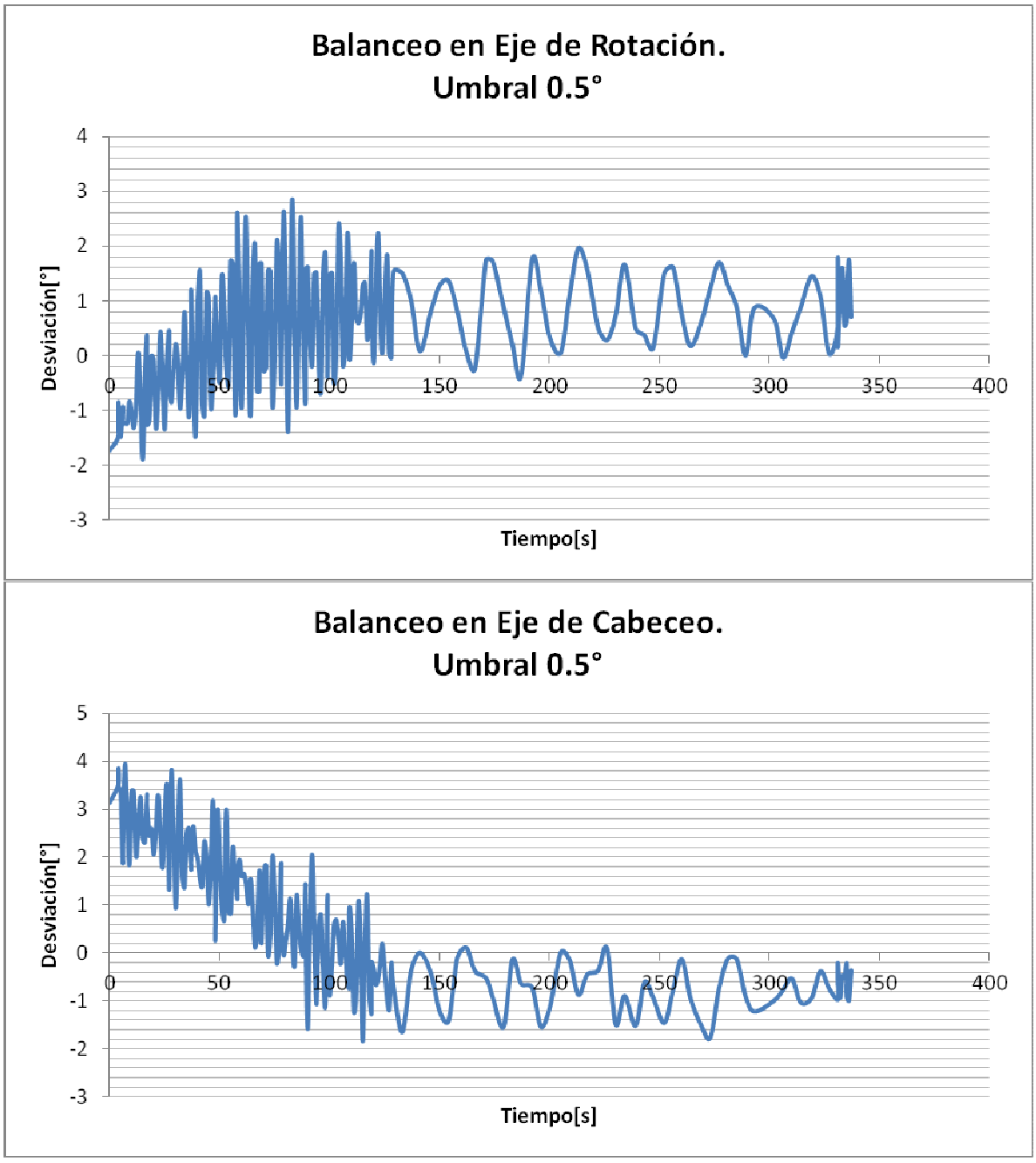


Figura 4.19. Pruebas de balanceo automático con un umbral de 0.5°

4.4. Despliegue en tiempo real y almacenamiento de datos de orientación de la plataforma

4.4.1. Despliegue gráfico en tiempo real.

Una de las mejoras que se implementaron en el nuevo sistema SIMUSAT_3.0, tiene que ver con el despliegue gráfico que proporciona un apoyo visual de la orientación de la plataforma. El programa de monitoreo ha sido cambiado para ofrecer ahora una nueva interpretación de despliegue un poco más realista que las versiones anteriores y aunque todavía no se ha conseguido el objetivo final de desplegar en 3D un modelo en CAD de la plataforma, esta versión se acerca un poco más a este punto.

En la figura 4.20 se muestra una de las versiones de despliegue implementadas para el sistema SIMUSAT_1.2. Un cubo con sus caras de distinto color, servía como referencia gráfica para observar el movimiento de la plataforma en tiempo real (Contreras, Fabiola).

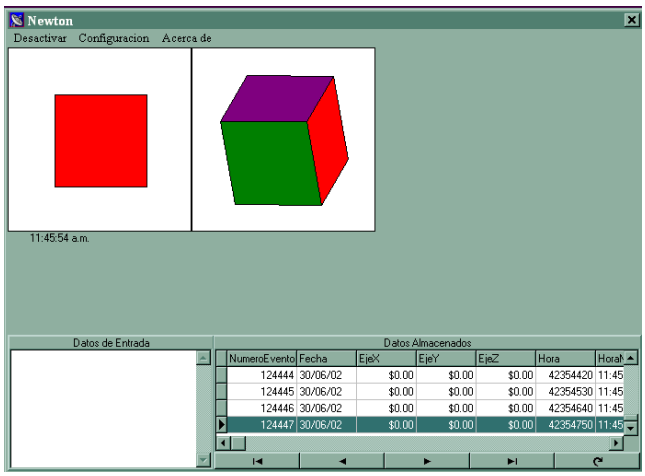


Figura 4.20. Despliegue de la orientación de la plataforma en el sistema SIMUSAT_1.2

En la figura 4.21 se muestra un ejemplo del despliegue de la orientación, utilizado en el sistema SIMUSAT_2.1. Esta fue una adaptación de una imagen de Saturno que se asemejaba de alguna manera a la plataforma de simulación.

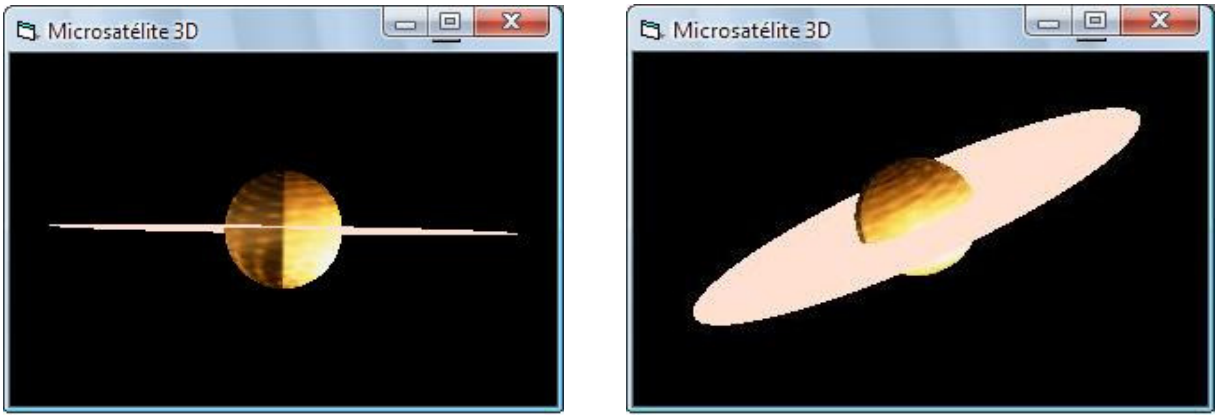


Figura 4.21. Despliegue de la orientación en el sistema simusat_2.1

4.4.2. Nueva versión de despliegue de datos de orientación.

En las siguientes figuras (4.22 y 4.23) se muestra un ejemplo del nuevo programa de graficación implementado en la nueva versión de simulador satelital.

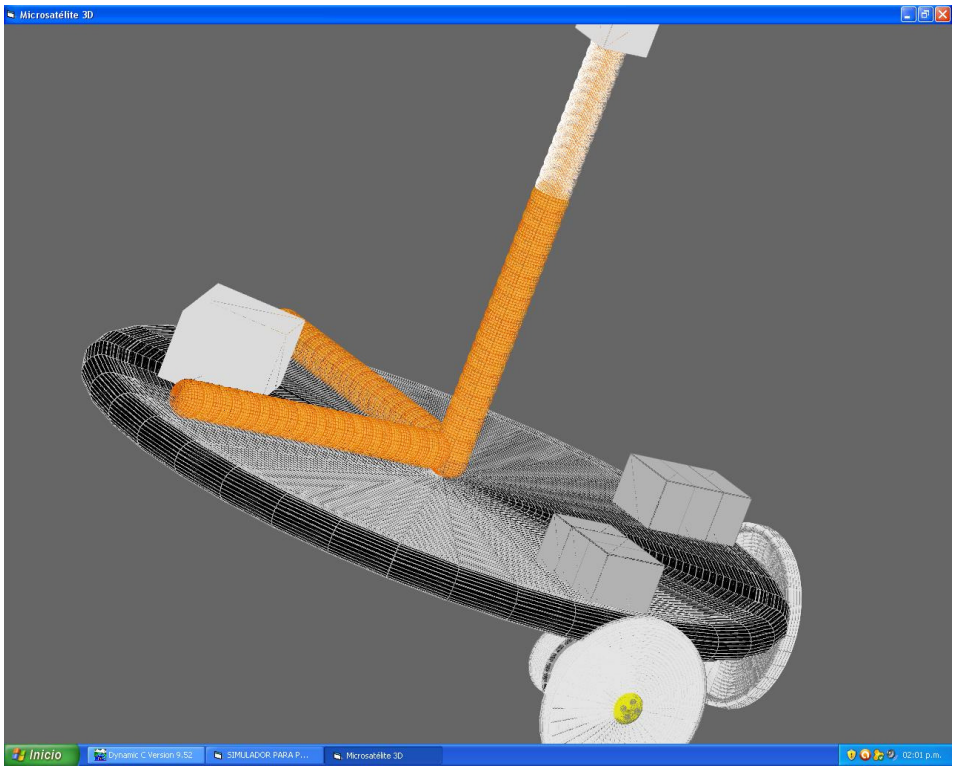
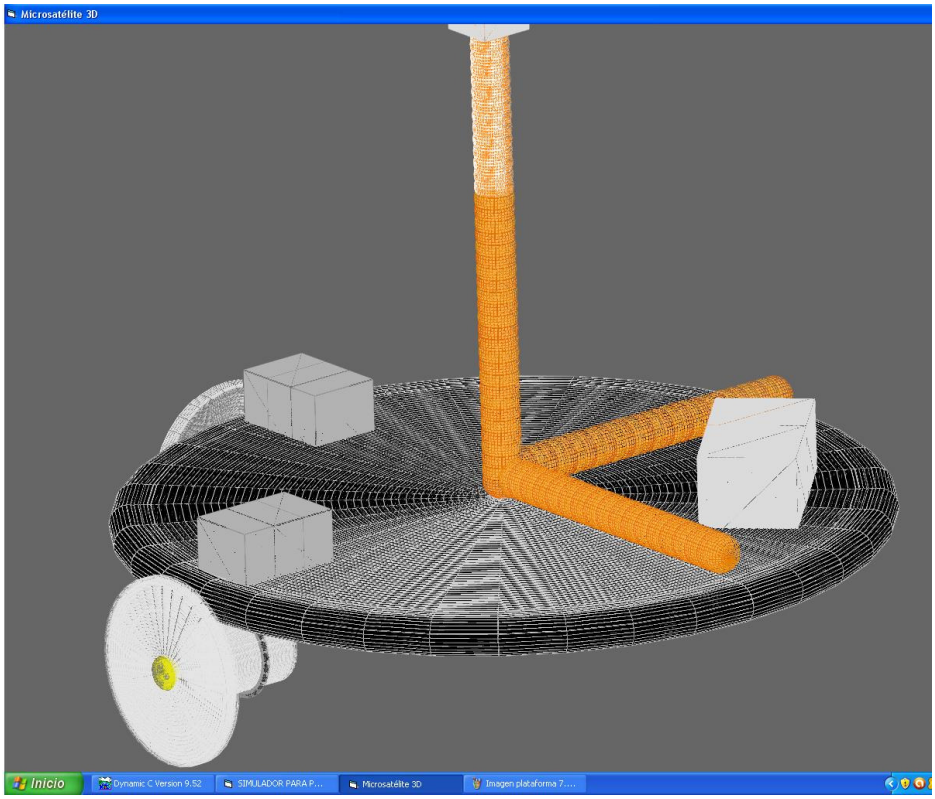


Figura 4.22. Despliegue de la orientación de la plataforma en el sistema SIMUSAT_3.0



4.23. Despliegue de la orientación en tiempo real.

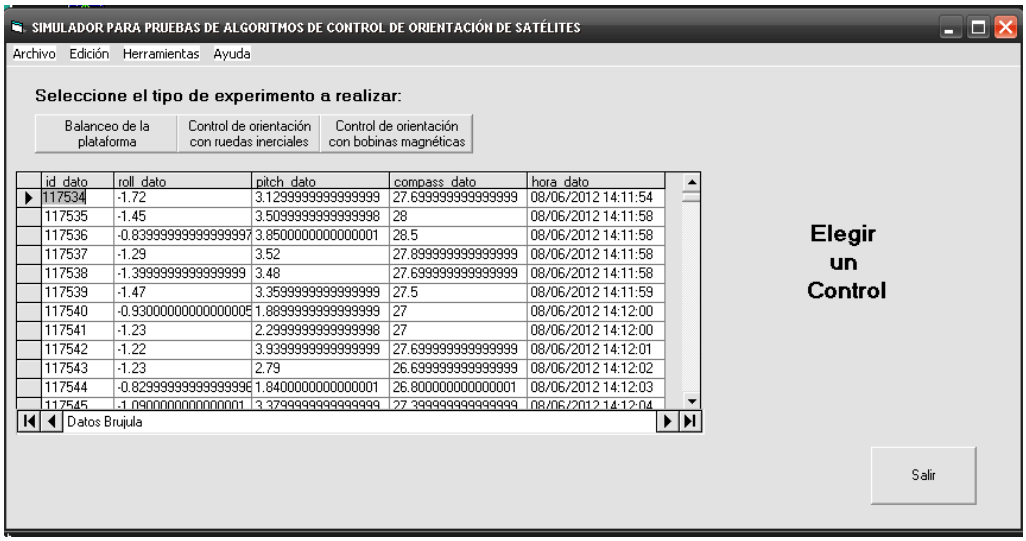
Con este nuevo programa de despliegue escrito en visual Basic, se tiene una representación un poco más realista de la plataforma de simulación, sin embargo, todavía carece de la precisión de un dibujo hecho en CAD. Una de las desventajas de este nuevo método de despliegue, es que la generación de la nueva perspectiva del dibujo, al cambiar la orientación de la plataforma, se genera con un retraso en el monitor de la estación terrena. Esto se debe a que el dibujo en realidad está compuesto por una gran cantidad de elementos, así, por ejemplo, cada bobina está hecha de 250 cilindros pequeños. Para generarla, hay que describir el tamaño de estos componentes, así como su posición relativa con respecto al centro de la plataforma, para de esta manera formar la bobina completa. Esto mismo se aplica para todos los componentes, por eso se genera un retraso de aproximadamente 2, segundos.

Aunque esta es una limitante, no obstante, el despliegue es mucho más cercano a la realidad. Los datos importantes para hacer la evaluación del sistema son aquellos que se almacenan en la tabla de datos y que se grafican posteriormente al ensayo para su evaluación.

4.4.3. Archivo de almacenamiento de datos

Los archivos de almacenamiento de datos han permanecido con el mismo formato y la misma manera de funcionar, en esta caso no fue necesario hacer ninguna

modificación. Solamente hay que recordar que es necesario salvar el archivo anterior, antes de iniciar un nuevo ensayo.



id_dato	roll_dato	pitch_dato	compass_dato	hora_dato
117534	-1.72	3.1299999999999999	27.699999999999999	08/06/2012 14:11:54
117535	-1.45	3.5099999999999998	28	08/06/2012 14:11:58
117536	-0.83999999999999997	3.8500000000000001	28.5	08/06/2012 14:11:58
117537	-1.29	3.52	27.899999999999999	08/06/2012 14:11:58
117538	-1.3999999999999999	3.48	27.699999999999999	08/06/2012 14:11:58
117539	-1.47	3.3599999999999999	27.5	08/06/2012 14:11:59
117540	-0.93000000000000005	1.8899999999999999	27	08/06/2012 14:12:00
117541	-1.23	2.2999999999999998	27	08/06/2012 14:12:00
117542	-1.22	3.9399999999999999	27.699999999999999	08/06/2012 14:12:01
117543	-1.23	2.79	26.699999999999999	08/06/2012 14:12:02
117544	-0.82999999999999996	1.8400000000000001	26.800000000000001	08/06/2012 14:12:03
117545	-1.0900000000000001	3.3799999999999999	27.399999999999999	08/06/2012 14:12:04

Figura 4.24. Tabla de datos de orientación almacenados por el programa de monitoreo.

Figura 4.25. Pantalla principal del programa de monitoreo

Capítulo

5

Conclusiones y Recomendaciones.

En este capítulo se describen de manera puntual las conclusiones y de hacen las recomendaciones para dar continuidad al desarrollo de este sistema.

5.1. Conclusiones.

- Se ha diseñado, construido y probado un sistema de seguimiento servomecánico de la posición de una plataforma de simulación satelital, para compensar los pares gravitacionales externos.

- Se diseñó, construyó y probó un conjunto nuevo de masas deslizantes que permitieron reducir significativamente las oscilaciones en la plataforma de simulación.

- A partir de la calibración de la plataforma de simulación, se desarrolló un nuevo método de balanceo automático.

- Se llevaron a cabo mejoras en el despliegue gráfico en tiempo real de la orientación de la plataforma de simulación.

5.2. Recomendaciones.

Fabricar las masas deslizantes en un taller mecánico para darles un acabado profesional.

Apéndice A

Código de programación para llevar a cabo la calibración del sistema de masas deslizantes con servomecanismo.

```
/*************************************************************
  UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

  Instituto de Geografía, 2012
  Calibración de las masas deslizantes con servomecanismo.
  Realizado por:

  *** César David Huante Arana
  *** Francisco Gabriel Mendez Salazar.

        Adaptado por:

  *** Laura Areli Durán Florentino

  Asesor :

  ****Dr. Jorge Prado Molina

  Programa diseñado para el kit LP3500 microcontrolador Rabbit 3000.

  Descripción
  ===========
  El programa principal obtiene los datos de la brújula electrónica EZ-CCOMPASS-3
  del puerto C de la tarjeta de desarrollo LP3500.
  Las características de comunicación son las siguientes:
  Comunicación asíncrona RS-232.
  1 bit de inicio, 8 bits of datos, 1 bit de parada son bit de paridad y Bauds 9600.
  Esta es la configuración por default para el microcontrolador.

  Instrucciones
  ============
  1. Compilar y correr este programa.
  2. Con el cable aún conectado se despliegan los datos de interés en la ventana STDIO.
  3. Una vez depurado el programa, desconectar el cable de programación, sin detener la
ejecución.
  4. Ya se puede utilizar sin el cable de programación.
*/

/*    CONFIGURACIÓN DE LOS PUERTOS SERIALES:
//El puerto B para la brújula electrónica ez-compass3.
//El puerto E para el modem inalámbrico.
*/
```

```
#define BINBUFSIZE  511   // Número de bits en el búfer del compás electrónico.
#define BOUTBUFSIZE 511
#define EINBUFSIZE  511   // Número de bits en el búfer del Módem - PC.
#define EOUTBUFSIZE 511

//  Configurando los puertos

///////
// Baudaje del puerto serial B
///////
#ifndef _232BBAUD
#define _232BBAUD 9600
#endif

///////
// Baudaje del puerto serial E
///////
#ifndef _232EBAUD
#define _232EBAUD 9600       //Comunicación entre la plataforma y la PC.
#endif

#undef   OUTPUT_DRIVE
#define  OUTPUT_DRIVE  SOURCING

#define OUTCONFIG 0x00FF

#class auto         // Cambia el almacenamiento manual de variables por automático.
#memmap xmem          // Requerido para reducir el uso de memoria

int brujula[100];
int lenBrujula;

//////////////////////////////////////////////////////////////////////
/*Función que lee los datos del compras electrónico*/
int getBrujula()
{
  int pruebaCLetra,brujulaLetra,i,j;           //Declaración de variables
  i=0;
  lenBrujula=0;
  while(lenBrujula==0)                      //Iniciando la cadena de caracteres "bújula"
  {
    while ((brujulaLetra=serBgetc()) == -1);    //Lee el puerto B y obtiene los datos del compas,
guardando los datos en la variable "brujulaLetra"
    if(brujulaLetra=='$')                   //El programa espera el caracter de reconocimiento de
nuevo dato $ desde el compás.
    {
      i=0;
      brujula[i]=brujulaLetra;               //Salva el primer caracter en "brújula"
      i++;
      while ((brujulaLetra=serBgetc()) == -1);   //Lee el puerto B, obteniendo los datos del
compás y los guarda en la variable "brujulaLetra"
      while(brujulaLetra!='$')               //Mientras el caracter recibido sea diferente de $
```

```
      {
       brujula[i]=brujulaLetra;              // Guarda los datos de "brujulaLetra" en "brujula[i]"
       while ((brujulaLetra=serBgetc()) == -1);   //Salva el resto de los caracteres en "brujula".
       lenBrujula=i;
       i++;
       j=i-1;
      }
     }
   }
}

//////////////////////////////////////////////////////////////////////
/*Función que envía una cadena de caracteres al compás y escribe la cadena en la variable
"brujulaLetra"*/
int mandaBrujula()
{
  int brujulaLetra,i;
  i=0;
  while(i==0)                          //Ejecuta el ciclo hasta que el valor de "i" sea cero
  {
    while ((brujulaLetra=serBgetc()) == -1);   ///Lee el puerto B, obteniendo los datos de compas
y los guarda en la variable "brujulaLetra"
    if(brujulaLetra=='$');                 //$ es el primer dato que el compás manda de una cadena.
    {
     i=0;
     serEputc(brujulaLetra);               //Envía los datos al puerto E, para la comunicación entre
la plataforma y la PC.
     i++;
     while ((brujulaLetra=serBgetc()) == -1);     //Lee nuevamente por si el siguiente caracter es
el $ de la siguiente cadena.
     while(brujulaLetra!='$')
     {
      serEputc(brujulaLetra);
      while ((brujulaLetra=serBgetc()) == -1);    //Envía todo.
     }
    }
  }
}

/*Fución principal*/
void main()
{
  int comandoPC,i,j,k;
  char *endptr;
  char rollValor[6];
  char pitchValor[6];
  char azimuthValor[6];
  auto float ciclo, ciclo2,freq,incremento_base,incremento;
  int l;
  //DISPLAY
  brdInit();                 //Inicializa el controlador para mostrar progreso del programa en
pantalla.
```

```
  devPowerSet(DISPDEV, 1);     //Habilita el display y el búfer de datos por teclado.
  dispInit();                  //Inicializa el módulo.
  //DISPLAY
  lenBrujula=0;

  i=0;
  j=0;
  k=0;
  //Inicializa todos lor puertos
  brdInit();             //Habilita el Rs232.
  serBopen(_232BBAUD);    //Abre la comunicación de puerto serie B a 9600 baudios.
  serBwrFlush();          //Limpia el buffer de escritura del puerto serie B.
  serBrdFlush();          //Limpia el buffer de escritura del puerto serie B.

  serEopen(_232EBAUD);    // Abre el puerto de comunicación serial del puerto E a 9600
bauds.   PC-MÓDEM
  serEwrFlush();          //Limpia el buffer de escritura del puerto serie E.
  serErdFlush();          //Limpia el buffer de escritura del puerto serie E.
  serMode(0);             //Habilitar el puerto de comunicación serial que utilizara el micro (ver
manual).
  freq = pwmOutConfig(50ul);  //Define el valor de ancho de pulso del PWM (50ul-->50Hz)
  pwm_init(freq);

  //Valores iniciales para ciclo, ciclo2, incremento.
  ciclo=0.075;  //Esta variable mueve las masas a una posición de 50% del valor de ciclo PWM.
  ciclo2=0.075;
  incremento=0.001045;   // Este es el mínimo incremento o decremento en grados.
Equivalente a 2 grados.


  //CONFIGURANDO SALIDAS DE PWM
  pwmOut(0, ciclo);         //Salida Masa
  pwmOut(1, ciclo2);        //Salida Masa
  pwmOut(2, 0);            //Apagamos salida, no utilizada.


  while(1)    //While 1
  {
    costate     //costate
    {
      inicio:
      comandoPC=0;
      serEwrFlush();      //Limpia el bufér de transmisión.
      serErdFlush();      //Limpia el bufér de recepción.
      printf("\n\n\t\t***Start***\n\t***Select a control from the program of the PC.***");

      while(1)  //while 2
      {

        while ((comandoPC = serEgetc()) == -1)   ;   //While entre comando PC y datos del
puerto E
```

```
        {

          switch(comandoPC) //Espera el comando desde la PC
          {
      case 97: //Comando "a" en RS232 o prueba de balanceo en el programa en visual basic.
                    {
                              printf("\n\nIncrementos de 2 grados Eje X: %c",comandoPC);
                              serBwrFlush();      //Limpia el buffer de escritura del puerto serie E.
                              serBrdFlush();      //Limpia el buffer de lectura del puerto serie E.
                              printf("\nEspera, procesando datos de EZcompass...");
                              mandaBrujula();
                              getBrujula();       //Actualiza la variable lenbrujula y brujula
                              printf("\nEnviar e imprimir datos de EZcompass: ");

           for(i=1;i<15;i++)  //Ciclo que incremente el valor de PWM para un movimiento de 6°
en el servomotor.
                                                        {
                              ciclo = ciclo - (incremento*3); //Son 15 incrementos de 6 grados cada
uno.
                              pwmOut(1, ciclo);

                    printf("\nRetardo de unos cuantos segundos");
                  for(k=1;k<=25;k++) //Imprime valores desde el compás al tiempo que genera un
retardo de 25 segundos
                    {
                     printf("\t  %d",k);
                     getBrujula();
                                              mandaBrujula();
                    }
                    printf("\n\nNumero de Iteracion %d",i);
                    printf("\n\nPosicion en grados %d",j);
                    printf("\n\nValor ciclo PWM %f",ciclo);
                                    }
                     }
                    break;

                    default:
                              {
                              serEwrFlush();      //Limpia el buffer de escritura del puerto serie E.
                    serErdFlush();      //Limpia el buffer de escritura del puerto serie E.
                    goto inicio;
                              }//Cierra default
                    break;   //Cierra case.
          pwmOut(0, 0.075);
          pwmOut(1, 0.075);
        }
       }
      }

      }//Cierra while umbral.
    }//Cierra costate.
  }//Cierra while 1.
```

Código de programación para llevar a cabo el seguimiento servomecánico básico del sistema de masas deslizantes.

```
/*************************************************************
  UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

  Instituto de Geografía, 2012

  Seguimiento de la posición con sistema de masas deslizantes con servomecanismo.

  Realizado por:

  *** César David Huante Arana
  *** Francisco Gabriel Mendez Salazar.

        Adaptado por:

  *** Laura Areli Durán Florentino

  Asesor :

  ****Dr. Jorge Prado Molina

        Programa diseñado para el kit LP3500 microcontrolador Rabbit 3000.

  Descripción
  ===========
  El programa principal obtiene los datos de la brújula electrónica EZ-CCOMPASS-3
  del puerto C de la tarjeta de desarrollo LP3500.
  Las características de comunicación son las siguientes:
  Comunicación asíncrona RS-232.
  1 bit de inicio, 8 bits of datos, 1 bit de parada son bit de paridad y Bauds 9600.
  Esta es la configuración por default para el microcontrolador.

  Instrucciones
  ============
  1.  Compilar y correr este programa.
  2.  Con el cable aún conectado se despliegan los datos de interés en la ventana STDIO.
  3.  Una vez depurado el programa, desconectar el cable de programación, sin detener la
ejecución.
  4.  Ya se puede utilizar sin el cable de programación.
*/


/*    CONFIGURACIÓN DE LOS PUERTOS SERIALES:
//El puerto B para la brújula electrónica ez-compass3.
//El puerto E para el modem inalámbrico.
*/
#define BINBUFSIZE  511   // Número de bits en el búfer del compás electrónico.
#define BOUTBUFSIZE 511
#define EINBUFSIZE  511   // Número de bits en el búfer del Módem - PC.
```

```
#define EOUTBUFSIZE 511

//  Configurando los puertos

///////
// Baudaje del puerto serial B
///////
#ifndef _232BBAUD
#define _232BBAUD 9600
#endif

// Baudaje del puerto serial E
///////
#ifndef _232EBAUD
#define _232EBAUD 9600       //Comunicación entre la plataforma y la PC.
#endif

#undef   OUTPUT_DRIVE
#define  OUTPUT_DRIVE  SOURCING

#define OUTCONFIG 0x00FF

#class auto         // Cambia el almacenamiento manual de variables por automático.
#memmap xmem          // Requerido para reducir el uso de memoria

int brujula[100];
int lenBrujula;

//////////////////////////////////////////////////////////////////////
/*Función que lee los datos del compras electrónico*/
int getBrujula()
{
  int pruebaCLetra,brujulaLetra,i,j;          //Declaración de variables
  i=0;
  lenBrujula=0;
  while(lenBrujula==0)                    //Iniciando la cadena de caracteres "bújula"
  {
    while ((brujulaLetra=serBgetc()) == -1);    //Lee el puerto B y obtiene los datos del compas,
guardando los datos en la variable "brujulaLetra"
    if(brujulaLetra=='$')                  //El programa espera el caracter de reconocimiento de
nuevo dato $ desde el compás.
    {
      i=0;
      brujula[i]=brujulaLetra;              //Salva el primer caracter en "brújula"
      i++;
      while ((brujulaLetra=serBgetc()) == -1);  //Lee el puerto B, obteniendo los datos del
compás y los guarda en la variable "brujulaLetra"
      while(brujulaLetra!='$')              //Mientras el caracter recibido sea diferente de $
      {
        brujula[i]=brujulaLetra;            // Guarda los datos de "brujulaLetra" en "brujula[i]"
        while ((brujulaLetra=serBgetc()) == -1);  //Salva el resto de los caracteres en "brujula".
        lenBrujula=i;
```

```
      i++;
      j=i-1;
     }
    }
  }
}

//////////////////////////////////////////////////////////////////////
/*Función que envía una cadena de caracteres al compás y escribe la cadena en la variable
"brujulaLetra"*/
int mandaBrujula()
{
  int brujulaLetra,i;
  i=0;
  while(i==0)                          //Ejecuta el ciclo hasta que el valor de "i" sea cero
  {
   while ((brujulaLetra=serBgetc()) == -1);   ///Lee el puerto B, obteniendo los datos de compas
y los guarda en la variable "brujulaLetra"
   if(brujulaLetra=='$');                 //$ es el primer dato que el compás manda de una cadena.
   {
    i=0;
    serEputc(brujulaLetra);              //Envía los datos al puerto E, para la comunicación entre
la plataforma y la PC.
    i++;
    while ((brujulaLetra=serBgetc()) == -1);     //Lee nuevamente por si el siguiente caracter es
el $ de la siguiente cadena.
    while(brujulaLetra!='$')
    {
     serEputc(brujulaLetra);
     while ((brujulaLetra=serBgetc()) == -1);    //Envía todo.
    }
   }
  }
}
/*Función principal*/
void main()
{
  int comandoPC,i,j,optimoBalanceo,ejeOpcion,optimoRuedas,controlPC,l; //Declaración de
variables
  int posRoll,lenRoll,vueltasRoll;
  char *endptr;
  auto float umbral,roll,pitch;
  auto float ciclo, ciclo2,freq,incremento,k,n;
  //DISPLAY
  brdInit();            //Habilita el Rs232.
  serBopen(_232BBAUD);    //Abre la comunicación de puerto serie B a 9600 baudios.
  serBwrFlush();         //Limpia el buffer de escritura del puerto serie B.
  serBrdFlush();         //Limpia el buffer de escritura del puerto serie B.

  serEopen(_232EBAUD);    // Abre el puerto de comunicación serial del puerto E a 9600
bauds.   PC-MÓDEM
  serEwrFlush();         //Limpia el buffer de escritura del puerto serie E.
```

```
  serErdFlush();         //Limpia el buffer de escritura del puerto serie E.
  serMode(0);            //Habilitar el puerto de comunicación serial que utilizara el micro (ver
manual).
  freq = pwmOutConfig(50ul); //Define el valor de ancho de pulso del PWM (50ul-->50Hz)
  pwm_init(freq);

  //Valores iniciales para ciclo, ciclo2, incremento.
  ciclo=0.075;  //Esta variable mueve las masas a una posición de 50% del valor de ciclo PWM.
  ciclo2=0.075;
  incremento=0.001045;   // Este es el mínimo incremento o decremento en grados.
Equivalente a 2 grados.


  //CONFIGURANDO SALIDAS DE PWM
  pwmOut(0, ciclo);         //Salida Masa
  pwmOut(1, ciclo2);        //Salida Masa
  pwmOut(2, 0);             //Apagamos salida, no utilizada.

  while(1)    //While 1
  {
    costate     //costate
    {
      inicio:
      comandoPC=0;
      printf("\n\n\t\t***Start***\n\t***Select a control from the program of the PC.***");

      while(1)  //while 2
      {
                              serBwrFlush();      //Limpia el buffer de escritura del puerto serie E.
      serBrdFlush();      //Limpia el buffer de escritura del puerto serie E.

        while ((comandoPC = serEgetc()) == -1)    ;   //While entre comando PC y datos del
puerto E
        {

          switch(comandoPC) //Espera el comando desde la PC
          {

                    case 97:  //Comando "a" en RS232 o prueba de balanceo en el programa en
visual basic.

              {
                        printf("\nEspera, procesando datos de EZcompass...");
                printf("\nSend and print the value of the compass: ");
                        serBwrFlush();      //Limpia el buffer de escritura del puerto serie E.
                        serBrdFlush();      //Limpia el buffer de lectura del puerto serie E.
                        printf("\nWait, reading from the compass...");
                              for(k=0;k<=9;k++) //Adquiere 10 datos de orientación desde el
compás.
                       {
                                getBrujula();
                                mandaBrujula();
```

```
                    waitfor(DelayMs(500));    //Retarddo de 500 ms
                  }

                  for(i=0;i<=lenBrujula;i++)  //Obtiene la posición en la variable
"brujulaLetra" para los caracteres (Roll) y P (pitch).
                  {
                    printf("%c",brujula[i]);
                    if(brujula[i]=='R')
                    {
                      posRoll=i;

                    }

                    if(brujula[i]=='P')
                    {
                    posPitch=i;

                    }

                    if(brujula[i]=='T')
                    {
                      posTemperatura=i;

                    }
                  }

                  j=0;

                  lenRoll=posPitch-posRoll-1;

                  for(i=posRoll+1;i<posPitch;i++)        // Separa en la variable rollValor el
valor de Roll.
                      {
                    rollValor[j]=brujula[i];
                    j++;
                      }

                  for(;j<6;j++)
                      {
                    rollValor[j]='0';
                      }

                  roll = strtod(rollValor, &endptr);     //Convierte la cadena "rollValor" en
número.
                  printf("\n\n roll, %f",roll);

                  j=0;

                  lenPitch=posTemperatura-posPitch-1;

                  for(i=posPitch+1;i<posTemperatura;i++)  // Separa en la variable pitchValor
el valor de Pitch.
```

```
                    {
                      pitchValor[j]=brujula[i];
                      j++;
                    }

                    for(;j<6;j++)
                    {
                      pitchValor[j]='0';
                    }

                    pitch = strtod(pitchValor, &endptr);   //Convierte la cadena pitchValor en
número.
                    printf("\n\n pitch, %f",pitch);

                 //Las variables pitch  y roll contienen valores numéricos


                ciclo=((0.0009*((roll+8.952)/0.2013)))+ 0.03;  //Ecuación que relaciona el valor de
ciclo PWM con la posición de la platina en el eje X.


                printf("\n\n Valor PWM X, %f",ciclo);

                if(ciclo>0.03) //Restringe el valor de PWM dentro del rango de operación del
servomotor.
                 {
                 if(ciclo<0.12)
                    {
                   pwmOut(0, ciclo);
                   k=1;
                            for(k=1;k<=25;k++)
                          {
                          getBrujula();
                          mandaBrujula();
                          }

                   }
                 else{}
                 }
                else
                            {
                   printf("\n\n El valor está fuera del rango de operacion. Probar nuevamente");

                   }


              }//Cierre case 97

              break;
```

```
                    case 98: //Comando "b" en RS232 o prueba de ruedas inerciales en el
programa en visual basic.

                          {
                  printf("\nEspera, procesando datos de EZcompass...");
                   serBwrFlush();      //Limpia el buffer de escritura del puerto serie E.
                                              serBrdFlush();      //Limpia el buffer de escritura del
puerto serie E
                           printf("\nWait, reading from the compass...");
                   printf("\nSend and print the value of the compass: ");
                              for(k=0;k<=10;k++) //Adquiere 10 datos de orientación desde el
compás.
                    {
                      getBrujula();
                      mandaBrujula();
                      waitfor(DelayMs(500));
                      }


                    for(i=0;i<=lenBrujula;i++)  //Obtiene la posición en la variable "brujulaLetra" para
los caracteres (Roll) y P (pitch).
                           {
                      printf("%c",brujula[i]);
                      if(brujula[i]=='R')
                      {
                        posRoll=i;

                      }

                      if(brujula[i]=='P')
                      {
                      posPitch=i;

                      }

                      if(brujula[i]=='T')
                      {
                        posTemperatura=i;

                      }
                     }

                     j=0;

                     lenRoll=posPitch-posRoll-1;

                     for(i=posRoll+1;i<posPitch;i++)         // Separa en la variable rollValor el
valor de Roll.
                         {
                      rollValor[j]=brujula[i];
                      j++;
                         }
```

```
                    for(;j<6;j++)
                        {
                      rollValor[j]='0';
                        }

                    roll = strtod(rollValor, &endptr);     //Convierte la cadena "rollValor" en
número.
                    printf("\n\n roll, %f",roll);

                    j=0;

                    lenPitch=posTemperatura-posPitch-1;

                    for(i=posPitch+1;i<posTemperatura;i++)  // Separa en la variable pitchValor
el valor de Pitch.
                    {
                      pitchValor[j]=brujula[i];
                      j++;
                    }

                    for(;j<6;j++)
                    {
                      pitchValor[j]='0';
                    }

                    pitch = strtod(pitchValor, &endptr);   //Convierte la cadena pitchValor en
número.
                    printf("\n\n pitch, %f",pitch);

                //Las variables pitch  y roll contienen valores numéricos


                     ciclo2=((0.0009*((pitch+9.0048)/0.206)))+0.03; //Ecuación que relaciona
el valor de ciclo PWM con la posición de la platina en el eje Y.
                     printf("\n\n Valor de ciclo PWM en eje y, %f",ciclo2);

                     if(ciclo2>0.03) //Restringe el valor de PWM dentro del rango de operación
del servomotor.
                      {
                      if(ciclo2<0.12)
                        {
                        pwmOut(1, ciclo2);
               k=1;
                                for(k=1;k<=25;k++)
                               {
                               getBrujula();
                               mandaBrujula();
                        }
               }

                     else{}
```

```
                }

                        else
                          {
                           printf("\n\n El valor está fuera del rango de operacion. Probar
nuevamente");

                          }



                      }//Cierre case 98
                      break;



              case 99: //Comando "c" en RS232 o prueba de bobinas magnéticas en el
programa en visual basic.

                      {
              printf("\nEspera, procesando datos de EZcompass...");
              serBwrFlush();      //Limpia el buffer de escritura del puerto serie E.
                                    serBrdFlush();      //Limpia el buffer de escritura del puerto
serie E
                      printf("\nWait, reading from the compass...");
                      printf("\nSend and print the value of the compass: ");

               for(k=0;k<=11;k++) //Adquiere 12 datos de orientación desde el compás.
                {
                        getBrujula();
                        mandaBrujula();
                        waitfor(DelayMs(300));
                        }

               for(i=0;i<=lenBrujula;i++)  //Obtiene la posición en la variable "brujulaLetra" para
los caracteres (Roll) y P (pitch).
                      {
                        printf("%c",brujula[i]);
                        if(brujula[i]=='R')
                        {
                          posRoll=i;

                        }

                        if(brujula[i]=='P')
                        {
                        posPitch=i;

                        }

                        if(brujula[i]=='T')
                        {
```

```
                posTemperatura=i;

              }
            }

            j=0;

            lenRoll=posPitch-posRoll-1;

            for(i=posRoll+1;i<posPitch;i++)        // Separa en la variable rollValor el
valor de Roll.
                {
              rollValor[j]=brujula[i];
              j++;
                }

            for(;j<6;j++)
                {
              rollValor[j]='0';
                }

            roll = strtod(rollValor, &endptr);    //Convierte la cadena "rollValor" en
número.
            printf("\n\n roll, %f",roll);

            j=0;

            lenPitch=posTemperatura-posPitch-1;

            for(i=posPitch+1;i<posTemperatura;i++)  // Separa en la variable pitchValor
el valor de Pitch.
            {
              pitchValor[j]=brujula[i];
              j++;
            }

            for(;j<6;j++)
            {
              pitchValor[j]='0';
            }

            pitch = strtod(pitchValor, &endptr);   //Convierte la cadena pitchValor en
número.
            printf("\n\n pitch, %f",pitch);

          //Las variables pitch  y roll contienen valores numéricos


           ciclo=((0.0009*((roll+8.952)/0.2013)))+ 0.03;  //Ecuación que relaciona el valor
de ciclo PWM con la posición de la platina en el eje X.
              ciclo2=((0.0009*((pitch+9.0048)/0.206)))+0.03;   //Ecuación que relaciona
el valor de ciclo PWM con la posición de la platina en el eje Y.
```

```
                printf("\n\n Valor de ciclo PWM en eje x, %f",ciclo);
                printf("\n\n Valor de ciclo PWM en eje y, %f",ciclo2);

                        if(ciclo2>0.03)   //Restringe el valor de PWM en ambos ejes dentro del
rango de operación del servomotor.
                         {
                         if(ciclo2<0.12)
                           {
                   pwmOut(1, ciclo2);
                   }
                         else{}
                  }
                 else
                  {
                  printf("\n\n El valor está fuera del rango de operacion. Probar nuevamente");
                  }
                  if(ciclo>0.03)  //Restringe el valor de PWM dentro del rango de operación del
servomotor.
                         {
                         if(ciclo<0.12)
                           {
                   pwmOut(0, ciclo);

                   }
                  else{}
                  }
                        else
                           {
                            printf("\n\n El valor está fuera del rango de operacion. Probar
nuevamente");
                           }
                                    for(k=0;k<=15;k++)   //Adquiere 15 datos desde el compás.
                                    {
                                    getBrujula();
                                    mandaBrujula();
                                    }
                              }//Cierre case 100
                              break;
                           default:
                           {
                       serEwrFlush();         //Limpia el buffer de escritura del puerto serie E.
                      serErdFlush();         //Limpia el buffer de escritura del puerto serie E.
                           goto inicio;
                           }//Cierra default
                            break;//Cierra opciones case
                           }//Cierre switch
                      }//Cierre de while 2

                 }//Cierre while serEgetc
          }//Cierre costate
         }//Cierre de while 1
        }//cierre main
```

Código de programación para el método de balanceo en dos ejes para plataforma de simulación con servomecanismo.

```
.
/*************************************************************
  UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

  Instituto de Geografía, 2012

  Método de balanceo en dos ejes para plataforma de simulación con servomecanismo.

  Realizado por:

  *** Laura Areli Durán Florentino

  Asesor :

  ****Dr. Jorge Prado Molina

  Descripción
  ===========

  Las características de comunicación son las siguientes:
  Comunicación asíncrona RS-232.
  1 bit de inicio, 8 bits of datos, 1 bit de parada son bit de paridad y Bauds 9600.
  Esta es la configuración por default para el microcontrolador.

  Instrucciones
  ============
  1.  Compilar y correr este programa.
  2.  Con el cable aún conectado se despliegan los datos de interés en la ventana STDIO.
  3.  Una vez depurado el programa, desconectar el cable de programación, sin detener la
ejecución.
  4.  Ya se puede utilizar sin el cable de programación.
*/

/*    CONFIGURACIÓN DE LOS PUERTOS SERIALES:
//El puerto B para la brújula electrónica ez-compass3.
//El puerto E para el modem inalámbrico.
*/
#define BINBUFSIZE  511   // Número de bits en el búfer del compás electrónico.
#define BOUTBUFSIZE 511
#define EINBUFSIZE  511   // Número de bits en el búfer del Módem - PC.
#define EOUTBUFSIZE 511

//  Configurando los puertos

///////
// Baudaje del puerto serial B
///////
#ifndef _232BBAUD
#define _232BBAUD 9600
```

```
#endif

// Baudaje del puerto serial E
///////
#ifndef _232EBAUD
#define _232EBAUD 9600       //Comunicación entre la plataforma y la PC.
#endif

#undef   OUTPUT_DRIVE
#define  OUTPUT_DRIVE  SOURCING

#define OUTCONFIG 0x00FF

#class auto         // Cambia el almacenamiento manual de variables por automático.
#memmap xmem          // Requerido para reducir el uso de memoria

int brujula[100];
int lenBrujula;


//////////////////////////////////////////////////////////////////////
/*Función que lee los datos del compras electrónico*/
int getBrujula()
{
  int pruebaCLetra,brujulaLetra,i,j;          //Declaración de variables
  i=0;
  lenBrujula=0;
  while(lenBrujula==0)                    //Iniciando la cadena de caracteres "bújula"
  {
    while ((brujulaLetra=serBgetc()) == -1);    //Lee el puerto B y obtiene los datos del compas,
guardando los datos en la variable "brujulaLetra"
    if(brujulaLetra=='$')                  //El programa espera el caracter de reconocimiento de
nuevo dato $ desde el compás.
    {
      i=0;
      brujula[i]=brujulaLetra;              //Salva el primer caracter en "brújula"
      i++;
      while ((brujulaLetra=serBgetc()) == -1);   //Lee el puerto B, obteniendo los datos del
compás y los guarda en la variable "brujulaLetra"
      while(brujulaLetra!='$')               //Mientras el caracter recibido sea diferente de $
      {
        brujula[i]=brujulaLetra;             // Guarda los datos de "brujulaLetra" en "brujula[i]"
        while ((brujulaLetra=serBgetc()) == -1);   //Salva el resto de los caracteres en "brujula".
        lenBrujula=i;
        i++;
        j=i-1;
      }
    }
  }
}

//////////////////////////////////////////////////////////////////////
```

```
/*Función que envía una cadena de caracteres al compás y escribe la cadena en la variable
"brujulaLetra"*/
int mandaBrujula()
{
  int brujulaLetra,i;
  i=0;
  while(i==0)                          //Ejecuta el ciclo hasta que el valor de "i" sea cero
  {
   while ((brujulaLetra=serBgetc()) == -1);   ///Lee el puerto B, obteniendo los datos de compas
y los guarda en la variable "brujulaLetra"
   if(brujulaLetra=='$');                 //$ es el primer dato que el compás manda de una cadena.
   {
    i=0;
    serEputc(brujulaLetra);              //Envía los datos al puerto E, para la comunicación entre
la plataforma y la PC.
    i++;
    while ((brujulaLetra=serBgetc()) == -1);     //Lee nuevamente por si el siguiente caracter es
el $ de la siguiente cadena.
    while(brujulaLetra!='$')
    {
     serEputc(brujulaLetra);
     while ((brujulaLetra=serBgetc()) == -1);    //Envía todo.
    }
   }
  }
}
/*Función principal*/
void main()
{
  int
comandoPC,umbralOpcion,i,j,optimoBalanceo,ejeOpcion,optimoRuedas,controlPC,ejeBobinas;
  int posRoll,lenRoll,vueltasRoll,comandoPicRoll,signoRuedaRoll;
  int posPitch,lenPitch,vueltasPitch,comandoPicPitch,signoRuedaPitch,vueltasAzimuthtmp;
  int
posTemperatura,pitchOroll,vueltas,posFinal,pitchOrollOazimuth,signoRueda,numeroPWM,nu
meroRueda,controlPCTemp;
  char  comandoPic;
  char *endptr;
  float umbral,roll,pitch,valor,azimuth,valor1,n;
  char rollValor[6];
  char pitchValor[6];
  auto float ciclo, ciclo2,freq,incremento_base,incremento,k;
  int l,m,x;
  //DISPLAY
  brdInit();            //Habilita el Rs232.
  serBopen(_232BBAUD);    //Abre la comunicación de puerto serie B a 9600 baudios.
  serBwrFlush();         //Limpia el buffer de escritura del puerto serie B.
  serBrdFlush();         //Limpia el buffer de escritura del puerto serie B.

  serEopen(_232EBAUD);    // Abre el puerto de comunicación serial del puerto E a 9600
bauds.   PC-MÓDEM
  serEwrFlush();         //Limpia el buffer de escritura del puerto serie E.
```

```
  serErdFlush();         //Limpia el buffer de escritura del puerto serie E.
  serMode(0);            //Habilitar el puerto de comunicación serial que utilizara el micro (ver
manual).
  freq = pwmOutConfig(50ul); //Define el valor de ancho de pulso del PWM (50ul-->50Hz)
  pwm_init(freq);

  //Valores iniciales para ciclo, ciclo2, incremento.
  //ciclo=0.075;  //Esta variable mueve las masas a una posición de 50% del valor de ciclo
PWM.
  //ciclo2=0.075;
  incremento_base=0.001045;   // Este es el mínimo incremento o decremento en grados.
Equivalente a 2 grados.
  incremento=0.0005;         //El valor que será sumado eo restado dela variable ciclo o ciclo2
para mover las masas en incremento o decremento.

  //CONFIGURANDO SALIDAS DE PWM
  //pwmOut(0, ciclo);         //Salida Masa
  //pwmOut(1, ciclo2);        //Salida Masa
  //pwmOut(2, 0);             //Apagamos salida, no utilizada.

  i=0;
  j=0;
  k=0;
  l=0;

  while(1)    //While 1
  {
    costate     //costate
    {
      inicio:
      optimoBalanceo=0;
      comandoPC=0;
      umbral=0;
      comandoPC=0;
                              serBwrFlush();      //Limpia el buffer de escritura del puerto serie E.
      serBrdFlush();      //Limpia el buffer de escritura del puerto serie E.

      printf("\n\n\t\t***Start***\n\t***Select a control from the program of the PC.***");

      while(1)  //while 2
      {

        while ((comandoPC = serEgetc()) == -1)    ;   //While entre comando PC y datos del
puerto E
        {
            serBwrFlush();      //Limpia el buffer de escritura del puerto serie E.
                                    serBrdFlush();      //Limpia el buffer de escritura del puerto
serie E.
            getBrujula();
                    mandaBrujula();
            for(i=0;i<=lenBrujula;i++)  //Obtiene la posición en la variable "brujulaLetra" para los
caracteres (Roll) y P (pitch).
```

```
                    {
                      printf("%c",brujula[i]);
                      if(brujula[i]=='R')
                      {
                        posRoll=i;

                      }

                      if(brujula[i]=='P')
                      {
                      posPitch=i;

                      }

                      if(brujula[i]=='T')
                      {
                        posTemperatura=i;

                      }
                    }

                    j=0;

                    lenRoll=posPitch-posRoll-1;

                    for(i=posRoll+1;i<posPitch;i++)        // Separa en la variable rollValor el
valor de Roll.
                        {
                      rollValor[j]=brujula[i];
                      j++;
                        }

                    for(;j<6;j++)
                        {
                      rollValor[j]='0';
                        }

                    roll = strtod(rollValor, &endptr);     //Convierte la cadena "rollValor" en
número.
                    printf("\n\n roll, %f",roll);

                    j=0;

                    lenPitch=posTemperatura-posPitch-1;

                    for(i=posPitch+1;i<posTemperatura;i++)  // Separa en la variable pitchValor
el valor de Pitch.
                    {
                      pitchValor[j]=brujula[i];
                      j++;
                    }
```

```
                    for(;j<6;j++)
                    {
                      pitchValor[j]='0';
                    }

                    pitch = strtod(pitchValor, &endptr);   //Convierte la cadena pitchValor en
número.
                    printf("\n\n pitch, %f",pitch);

              //Las variables pitch  y roll contienen valores numéricos


        switch(comandoPC) //Espera el comando desde la PC
        {

         /*********************************************************************
                    Control de las masas deslizantes
         *********************************************************************
        */

        case 97:
        {
        printf("\n\nThe PC sent the command: %c",comandoPC);
                  while ((umbralOpcion = serEgetc()) == -1);  //while3
        {
        switch(umbralOpcion)    //Switch variable umbral depending of the date received.
        {
         case 48:
         {
         umbral=3;
         for(k=0;k<=9;k++) //Adquiere 10 datos de orientación desde el compás.
                  {
                  getBrujula();
                  mandaBrujula();
                  waitfor(DelayMs(100));    //Retarddo de 500 ms
                  }
         //ciclo=0.07673;    Valor PWM próximo a umbral 3 grados ejeX
         //ciclo2=0.0824481; Valor PWM para umbral 3 grados ejeY
         // [°] = 0.2013(%PWM) - 8.952 Ecuación que relaciona el ángulo de la plataforma con el
%PWM Eje X
         //[°] = 0.206(%PWM) - 9.0048 Ecuación que relaciona el ángulo de la plataforma con el
%PWM Eje Y

            ciclo=((0.0009*((roll+8.952)/0.2013)))+ 0.029; //Se compensa inestabilidad en eje X.
Valor obtenido de pruebas
            ciclo2=((0.0009*((pitch+9.0048)/0.206)))+0.0325;//Se compensa inestabilidad en eje
Y. Valor obtenido de pruebas
            printf("/n/nCiclo = %f",ciclo);
            printf("/n/nCiclo2 = %f",ciclo2);
            while((ciclo<0.076||ciclo>0.0777)||(ciclo2<0.0817||ciclo2>0.0827))  //Ciclo while
valores PWM
                    {
```

```
  if(ciclo>0.03&&ciclo<0.12)
            {
            if(ciclo<0.07673)
   {
   ciclo=(ciclo+incremento);
            pwmOut(0, ciclo);
            waitfor(DelayMs(100));
            getBrujula();
            mandaBrujula();
   printf("/n/nMoviendo plataforma en intervelos mínimos");
   }

   else
   {
   ciclo=(ciclo-incremento);
            pwmOut(0, ciclo);
            waitfor(DelayMs(100));
            getBrujula();
            mandaBrujula();
   printf("/n/nMoviendo plataforma en intervelos mínimos");
   }
                                                    }
       else
 {
  printf("\n\n El valor está fuera del rango de operacion. Probar nuevamente");
 }

if(ciclo2>0.03&&ciclo2<0.12)
                        {
  if(ciclo2<0.0824481)
   {
   ciclo2=(ciclo2+incremento);
            pwmOut(1, ciclo2);
            waitfor(DelayMs(100));
            getBrujula();
            mandaBrujula();
   printf("/n/nMoviendo plataforma en intervelos mínimos");
   }

   else
   {
   ciclo2=(ciclo2-incremento);
            pwmOut(1, ciclo2);
            waitfor(DelayMs(100));
            getBrujula();
            mandaBrujula();
   printf("/n/nMoviendo plataforma en intervelos mínimos");
                                                                  }
 }
            else
   {
   printf("\n\n El valor está fuera del rango de operacion. Probar nuevamente");
```

```
                }

            }//Cierra ciclo while valores PWM


        }
        break;

        case 49:
        {
        umbral=2;
        //ciclo=0.074495;  Valor PWM próximo a umbral 2 grados ejeX
        //ciclo2=0.0737103; Valor PWM próximo a umbral 2 grados eje Y

          ciclo=((0.0009*((roll+8.952)/0.2013)))+ 0.029; //Se compensa inestabilidad en eje X.
Valor obtenido de pruebas
          ciclo2=((0.0009*((pitch+9.0048)/0.206)))+0.0325;//Se compensa inestabilidad en eje
Y. Valor obtenido de pruebas
          printf("/n/nCiclo = %f",ciclo);
          printf("/n/nCiclo2 = %f",ciclo2);
          while((ciclo<0.0740||ciclo>0.0751)||(ciclo2<0.0727||ciclo2>0.0742))  //Ciclo while
valores PWM
                        {
                if(ciclo>0.03&&ciclo<0.12)
                        {
                        if(ciclo<0.074495)
                {
                ciclo=(ciclo+incremento);
                        pwmOut(0, ciclo);
                        waitfor(DelayMs(100));
                        getBrujula();
                        mandaBrujula();
                printf("/n/nMoviendo plataforma en intervelos mínimos");
                }

                else
                {
                ciclo=(ciclo-incremento);
                        pwmOut(0, ciclo);
                        waitfor(DelayMs(100));
                        getBrujula();
                        mandaBrujula();
                printf("/n/nMoviendo plataforma en intervelos mínimos");
                }
                                                        }
                    else
              {
                printf("\n\n El valor está fuera del rango de operacion. Probar nuevamente");
              }

            if(ciclo2>0.03&&ciclo2<0.12)
                                {
```

```
                    if(ciclo2<0.0737103)
                     {
                     ciclo2=(ciclo2+incremento);
                            pwmOut(1, ciclo2);
                            waitfor(DelayMs(100));
                            getBrujula();
                            mandaBrujula();
                     printf("/n/nMoviendo plataforma en intervelos mínimos");
                     }

                     else
                     {
                     ciclo2=(ciclo2-incremento);
                            pwmOut(1, ciclo2);
                            waitfor(DelayMs(100));
                            getBrujula();
                            mandaBrujula();
                     printf("/n/nMoviendo plataforma en intervelos mínimos");
                                                                              }
                   }
                            else
                     {
                    printf("\n\n El valor está fuera del rango de operacion. Probar nuevamente");
                     }

                  }//Cierra ciclo while valores PWM

          }
          break;

          case 50:
          {
          umbral=1;
          //ciclo=0.07226; Valor PWM próximo a umbral 1 grados eje X según ecuaciones de
calibración
          //ciclo2=0.0715258; Valor PWM próximo a umbral 1 grados ejeY según ecuaciones de
calibración
          //ciclo=0.07052; Valor PWM próximo a umbral 1 grados eje X  por acotamiento
experimental
          //ciclo2=0.0712; Valor PWM próximo a umbral 1 grados eje Y por acotamiento
experimental


                    ciclo=((0.0009*((roll+8.952)/0.2013)))+ 0.029;  //Se compensa inestabilidad en
eje X decrementando el valor de ciclo2
               ciclo2=((0.0009*((pitch+9.0048)/0.206)))+0.0325; //Se compensa inestabilidad en eje
Y decrementando el valor de ciclo2
               printf("/n/nCiclo = %f",ciclo);
               printf("/n/nCiclo2 = %f",ciclo2);
               while((ciclo<0.0695||ciclo>0.0712)||(ciclo2<0.0697||ciclo2>0.0718))  //El valor más
próximo a estabilizar en 0° la plataforma
                              {
```

```
   if(ciclo>0.03&&ciclo<0.12)
            {
            if(ciclo<0.07052) //Buscando el valor óptimo por apoximaciones
    {
    ciclo=(ciclo+incremento);
            pwmOut(0, ciclo);
            waitfor(DelayMs(100));
            getBrujula();
            mandaBrujula();
    printf("/n/nMoviendo plataforma en intervelos mínimos");
    printf("/n/nCiclo = %f",ciclo);
                                printf("/n/nCiclo2 = %f",ciclo2);
    }

    else
    {
    ciclo=(ciclo-incremento);
            pwmOut(0, ciclo);
            waitfor(DelayMs(100));
            getBrujula();
            mandaBrujula();
    printf("/n/nMoviendo plataforma en intervelos mínimos");
    printf("/n/nCiclo = %f",ciclo);
                                printf("/n/nCiclo2 = %f",ciclo2);
    }
                                                        }
        else
 {
   printf("\n\n El valor está fuera del rango de operacion. Probar nuevamente");
 }

if(ciclo2>0.03&&ciclo2<0.12)
                        {
   if(ciclo2<0.0712) //Buscando el valor óptimo por apoximaciones
    {
    ciclo2=(ciclo2+incremento);
            pwmOut(1, ciclo2);
            waitfor(DelayMs(100));
            getBrujula();
            mandaBrujula();
    printf("/n/nMoviendo plataforma en intervelos mínimos");
    printf("/n/nCiclo = %f",ciclo);
                                printf("/n/nCiclo2 = %f",ciclo2);
    }

    else
    {
    ciclo2=(ciclo2-incremento);
            pwmOut(1, ciclo2);
            waitfor(DelayMs(100));
            getBrujula();
            mandaBrujula();
```

```
                printf("/n/nMoviendo plataforma en intervelos mínimos");
                printf("/n/nCiclo = %f",ciclo);
                                        printf("/n/nCiclo2 = %f",ciclo2);
                                                                        }
              }
                        else
                {
                printf("\n\n El valor está fuera del rango de operacion. Probar nuevamente");
                 }

              }//Cierra ciclo while valores PWM
        }
        break;

        case 51:
        {
        umbral=0.5;
        //ciclo=0.07002385;  Valor PWM próximo a umbral 1 grados eje X según ecuaciones de
calibración
        //ciclo2=0.0693414;  Valor PWM próximo a umbral 0.5 grados eje Y según ecuaciones
de calibración
        //ciclo=0.07052; Valor PWM próximo a umbral 1 grados eje X  por acotamiento
experimental
         //ciclo2=0.0712; Valor PWM próximo a umbral 1 grados eje Y por acotamiento
experimental

              ciclo=((0.0009*((roll+8.952)/0.2013)))+ 0.029; //Se compensa inestabilidad en
eje X. Valor obtenido de pruebas
          ciclo2=((0.0009*((pitch+9.0048)/0.206)))+0.0325;//Se compensa inestabilidad en eje
Y. Valor obtenido de pruebas
          printf("/n/nCiclo = %f",ciclo);
          printf("/n/nCiclo2 = %f",ciclo2);
          while((ciclo<0.0695||ciclo>0.071)||(ciclo2<0.069||ciclo2>0.071))  //Ciclo while
valores PWM
                        {
                if(ciclo>0.03&&ciclo<0.12)
                        {
                        if(ciclo<0.07002385)
                {
                ciclo=(ciclo+incremento/2);
                        pwmOut(0, ciclo);
                        waitfor(DelayMs(30));
                        getBrujula();
                        mandaBrujula();
                printf("/n/nMoviendo plataforma en intervelos mínimos");
                }

                else
                {
                ciclo=(ciclo-incremento/2);
                        pwmOut(0, ciclo);
                        waitfor(DelayMs(30));
```

```
                    getBrujula();
                    mandaBrujula();
                 printf("/n/nMoviendo plataforma en intervelos mínimos");
                 }
                                                          }
                    else
               {
                 printf("\n\n El valor está fuera del rango de operacion. Probar nuevamente");
               }

              if(ciclo2>0.03&&ciclo2<0.12)
                                  {
                 if(ciclo2<0.0693414)
                  {
                  ciclo2=(ciclo2+incremento/2);
                    pwmOut(1, ciclo2);
                    waitfor(DelayMs(30));
                    getBrujula();
                    mandaBrujula();
                  printf("/n/nMoviendo plataforma en intervelos mínimos");
                  }

                  else
                  {
                  ciclo2=(ciclo2-incremento/2);
                    pwmOut(1, ciclo2);
                    waitfor(DelayMs(30));
                    getBrujula();
                    mandaBrujula();
                  printf("/n/nMoviendo plataforma en intervelos mínimos");
                                                                        }
               }
                     else
                  {
                 printf("\n\n El valor está fuera del rango de operacion. Probar nuevamente");
                  }

              }//Cierra ciclo while valores PWM
}
break;
default:
{
 ciclo=0;
                                      ciclo2=0;
 optimoBalanceo=0;
 comandoPC=0;
 comandoPC=0;
 serEwrFlush();      //Clean the writer buffer in port serial E.
 serErdFlush();      //Clean the reader buffer in port serial E.
 goto inicio;
}
break;
```

```
        }//Cierra segundo case

          l=0;
          m=0;
          while(l==0) //ciclo toma datos brújula
                {
           serBwrFlush();      //Limpia el buffer de escritura del puerto serie E.
                                    serBrdFlush();      //Limpia el buffer de escritura del puerto
serie E.
           getBrujula();
                  mandaBrujula();
           for(i=0;i<=lenBrujula;i++)  //Obtiene la posición en la variable "brujulaLetra" para los
caracteres (Roll) y P (pitch).
                        {
                          printf("%c",brujula[i]);
                          if(brujula[i]=='R')
                          {
                            posRoll=i;

                          }

                          if(brujula[i]=='P')
                          {
                          posPitch=i;

                          }

                          if(brujula[i]=='T')
                          {
                            posTemperatura=i;

                          }
                        }

                        j=0;

                        lenRoll=posPitch-posRoll-1;

                        for(i=posRoll+1;i<posPitch;i++)        // Separa en la variable rollValor el
valor de Roll.
                            {
                          rollValor[j]=brujula[i];
                          j++;
                            }

                        for(;j<6;j++)
                            {
                          rollValor[j]='0';
                            }

                        roll = strtod(rollValor, &endptr);    //Convierte la cadena "rollValor" en
número.
```

```
                        printf("\n\n roll, %f",roll);

                        j=0;

                        lenPitch=posTemperatura-posPitch-1;

                        for(i=posPitch+1;i<posTemperatura;i++)  // Separa en la variable pitchValor
el valor de Pitch.
                        {
                          pitchValor[j]=brujula[i];
                          j++;
                        }

                        for(;j<6;j++)
                        {
                          pitchValor[j]='0';
                        }

                        pitch = strtod(pitchValor, &endptr);   //Convierte la cadena pitchValor en
número.
                        printf("\n\n pitch, %f",pitch);

                  //Las variables pitch  y roll contienen valores numéricos

                 printf("\n\n umbral, %f",umbral);
                 n=abs(roll);
                 printf("\n\n valor absoluto roll, %f",n);
                 printf("\n\n\t\tValor inicial m=%d",m);

                 if(abs(roll)<umbral&&abs(pitch)<umbral) //while 4
                      {
                   x=1;
                   printf("\n\n\t\tIncremento x = %d",x);
                   printf("\n\n\t\tIncremento m = %d",m);
                   m++;
                   }
                 else{
                 printf("/n/nAun no se estabiliza la plataforma, enviando retardo 3 segundos");
                 }

                 if(m==15)
                 {
                 l=1;
                 printf("/n/n Valor de l = %d",l);
                 }

                     }//cierra while toma de datos.

                   if(m==15)
                     {
                     for(k=0;k<=10;k++) //Adquiere 11 datos de orientación desde el compás.
                              {
```

```
                        getBrujula();
                        mandaBrujula();
                        waitfor(DelayMs(100));    //Retarddo de 100 ms
                        }
                serEputc(36);
                        waitfor(DelayMs(5));
                        serEputc(76);
                        waitfor(DelayMs(5));
                        serEputc(13);
                        waitfor(DelayMs(30));
                        printf("/n/nFinaliza prueba");
                        goto inicio;
                }
                else{}


                }//Cierre while 3
               }//cierre while comando PC
                  break;
       }//Cierre del while 2
      }//cierre case
           }//Cierre costate
   }//Cierre de while 1
         }//cierre main

}
```

Bibliografía

Contreras F., 2004. “Pruebas de control de estabilización para un satélite pequeño empleando bobinas magnéticas y ruedas inerciales”. Tesis de Licenciatura. Ingeniería Electrónica. Facultad de Ingeniería, UNAM. Pp. 1- 86

Escobedo Lugo, Luis; 2012 “Simulador para pruebas de control y orientación para nanosatélites”. Tesis de Licenciatura. Ingeniería Eléctrica y Electrónica. Facultad de Ingeniería, UNAM. Fecha de examen: Febrero de 2012.

Hibbeler, R. C. *Mecánica para ingenieros. Dinámica.* Décima edición. Pearson Educación. pp. 553- 604. México, 2004.

Hughes P. C. (1986) “Spacecraft Attitude Dynamics” Wiley and Sons, NY.

Irving L. Kosow. *Máquinas eléctricas y transformadores.* Pearson Educación, 1993. pp. 429. Consultado el 26 de enero de 2011.

Juarez A., 2001. “Balanceo automático de un simulador para control de orientación de satélites” Tesis de licenciatura. Ingeniería Mecánica Eléctrica. Facultad de Ingeniería, UNAM, pp. 1- 79.

Méndez F. y Huante D., 2009. “Simulador para pruebas de control de orientación de satélites”**.** Tesis de licenciatura. Ingeniería Electrónica. Facultad de Ingeniería, UNAM. Fecha de examen: Agosto de 2009. pp1- 86.

Peterson G. S. (1992) “Sun seeking three-axis thruster control of a small satellite attitude control simulator”. Master of Science thesis. Utah State University. pp1- 105.

(Prado Molina, Jorge, 2007). “Sistema de simulación para pruebas de algoritmos de orientación y control de satélites pequeños”**.** Tesis de Doctorado. Ingeniería Mecánica Aplicada. Facultad de Ingeniería, UNAM. pp1- 120.

Prado J., Bisiacchi G., Mesinas M., Ruiz D. (2002a) “Sistema de Monitoreo Inalámbrico de un Simulador Para Control de Orientación de Satélites”. SOMI XVII Congreso Nacional de Instrumentación. Mérida, Yucatán, México, Octubre 14-18, Memorias en CD. pp1-12. Trabajo ELECTRO35.

Rabbit semiconductor (2007), “Wireless Control Application Kit”. Rabbit and Maxtream Modules. (Manuales y hojas de datos en 5 discos compactos)

Rizos I, Arbes J and Raoult J.C. (1971) “A Spherical Air-Bearing-Supported Test Facility for Performance Testing of Satellite Attitude Control Systems”. ESRO-CR66, also 4th. IFAC Symposium. Dubrovnic, Jugoslavia. September. 6-10. Pp 3.41-3.48

Stockdale, L.A.; *Servomecanismos*; Ediciones URMO; 1972
Bucley, V. Ruth; *Fundamentos de servosistemas*; Editorial Labor S.A.; España; 1974.

Wang P., Yee J. and Hadaegh F. (1999) “Experimental Study of Synchronized Rotation of Multiple Autonomous Spacecraft with Rule-Based Controls”. In Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit, no 99-4150, (Portland, Oregon), pp. 1098-1108, August 9-11, 1999.

Mesografía:

AeroAstro (2006)
http://www.aeroastro.com/publications/SSCO3-X-7.pdf

Fuente datos servomotor
http://www.servocity.com/html/hs-645mg_ultra_torque.html

Manual general para servomotores HITEC:
http://www.hitecrcd.com/support/manuals/servos.html

Mecanismo piñón-cremallera
http://ar.kalipedia.com/popup/popupWindow.html?anchor=klpmatgeo&tipo=imprimir&titulo=Imprimir%20Art%EDculo&xref=20070822klpingtcn_62.Kes