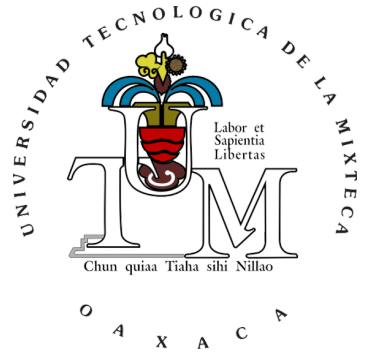


UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

Instituto de Electrónica y Mecatrónica

Control de un vehículo auto balanceado simplificado tipo Segway mediante sensores de un dispositivo móvil basado en Android

Tesis para obtener el título de:

Ingeniero en Mecatrónica

Presenta

Carolina Hernández Guzmán

Director:

Dr. Richard Jacinto Marquez Contreras

Huajuapan de León, Oaxaca, 13 de julio de 2018

A mi hermana, Paola.

Agradecimientos

Primero quiero agradercele a mi director de tesis, el Dr. Richard Jacinto Marquez Contreras por todo su apoyo a lo largo de este camino, por su paciencia y sus palabras de aliento.

A mis revisores: Dr. Jorge Luis Barahona, Dr. Hugo Fermín Leyva y Dr. Felipe Santiago, por los aportes a este trabajo de tesis, su tiempo y conocimientos aportados. A los profesores del instituto, gracias por todos los conocimientos y su guía en este camino.

A mis padres, por enseñarme el único camino, el del trabajo. Por darme el apoyo para seguir adelante y por estar siempre a mi lado. A mis hermanas, Daniela y Julieta que a la distancia me han alentado para continuar, que sepan que confío en ellas y se que lograrán mucho más.

A mi hermana Paola, si algo tengo claro es que no hubiera podido terminar esto sin ti. Éste también es tu logro. Porque fuiste la única que estuvo a mi lado desde el inicio y hasta el final, nadie mejor que tú sabe que tan difícil fue esto para mi y aún así te mantuviste conmigo. Gracias.

A Julio, principal fuente de retroalimentación de este trabajo. Gracias por impulsarme siempre a ser mejor, por enseñarme nuevos caminos y dejarme ver lo que puedo lograr. Gracias por permitirme crecer y quedarte a mi lado en los momentos difíciles. Por apoyarme y sin duda, creer en mí.

Finalmente, gracias a mis compañeros y más que eso, amigos que estuvieron conmigo a lo largo de este viaje.

Índice general

Índice de figuras

Glosario de siglas y símbolos

Siglas

API	Interfaz de Programación de Aplicaciones.
$L2CAP$	Protocolo de control y adaptación del enlace lógico.
LGR	Lugar Geométrico de las Raices.
$LIPO$	Polímero de Litio.
$MEMS$	Sistemas-Micro-Electro-Mecánicos.
PWM	Modulación por Ancho de Pulsos.
RF	Radio Frecuencia.
$RFCOMM$	Comunicación por radio frecuencia.
$RS232$	Estándar Recomendado 232.
SDP	Protocolo de Socket Directo.
$UUID$	Identificador Único Universal.
WIP	Péndulo Invertido con Llantas.
$WIPR$	Robot Péndulo Invertido con Llantas.

Símbolos

g	Aceleración gravitacional.
θ	Ángulo entre el péndulo y la vertical.
ϕ	Ángulo entre la llanta y la vertical.
T_m	Torque producido entre la llanta y el péndulo.
m_p	Masa del péndulo.
r_p	Distancia del centro de masa a la base del péndulo.
I_p	Inercia del péndulo.
m_w	Masa de la llanta.
r_w	Radio de la llanta.
I_w	Inercia de la llanta.
P_r	Fuerza sobre la vertical.
H_r	Fuerza sobre la horizontal.
H_{fr}	Fuerza tangencial.
F_{nw}	Fuerza normal que actua sobre la llanta.
R	Resistencia del motor.
k_v	Constante de fuerza electromotriz del motor.
k_t	Constante de torque del motor.

Resumen

El modelo del péndulo invertido móvil, cuya clasificación es de los robots móviles con llantas, ha sido estudiado en diferentes áreas a lo largo del tiempo debido a la inestabilidad natural del sistema.

Con la finalidad de contribuir a la línea de investigación del Instituto de mecatrónica y electrónica se desarrolla una plataforma experimental tipo péndulo invertido móvil en la que se incluya el uso de los dispositivos móviles como sensores alternos. Debido a la inestabilidad mecánica natural de la plataforma experimental que se desarrolla, se hace necesaria la implemetación de un controlador del sistema.

El siguiente documento presenta el desarrollo de un vehículo de autobalanceo mediante el uso de sensores de un dispositivo móvil. La desviación del cuerpo del péndulo fuera de la vertical se mide con el uso de sensores implementados en dispositivos móviles Android® que incluyen sensores basados en software y hardware que se encuentran a disposición de los desarrolladores para su uso. El envío de datos entre el dispositivo móvil y la tarjeta de implementación para el algoritmo de control se hace mediante protocolo de comunicación Bluetooth®.

El desarrollo que tuvo la plataforma y que presenta este documento se divide en tres áreas de trabajo: construcción de la estructura mecánica, implementación del algoritmo de control y el dispositivo móvil como sensor de orientación.

Capítulo 1

Introducción

Los robots son máquinas autónomas que involucran el conocimiento de diferentes áreas como la mecánica, electrónica, control, entre otras. Los robots móviles pueden dividirse en dos grandes categorías: con eslabones y con ruedas. En el caso de los robots con ruedas la dinámica es más simple respecto a los robots con eslabones y son más eficaces en términos de consumo de energía. Dentro de la clasificación de los robots con ruedas, los de dos ruedas presentan una dinámica más complicada que los robots con tres o más llantas. Sin embargo, éstos llegan a ser más pequeños que los robots con más llantas y eso los hace más versátiles.

Uno de los robots móviles más famosos en esta configuración (de dos ruedas) es el robot Segway®. Los robots Segway® y otros similares de dos llantas han sido diseñados bajo el modelo de un robot péndulo invertido móvil con llantas (WIPR, por sus siglas en inglés). El objetivo principal de control de este vehículo es mantener el balance vertical y prevenir colisiones del vehículo. Por otra parte, se desea que la respuesta del sistema sea rápida y así poder mantener al conductor de pie sobre la plataforma de manera que el vehículo pueda seguir desplazándose cuando el conductor requiera trasladarse de un lugar a otro. Este objetivo de control se logra obteniendo la inclinación del vehículo mediante diferentes tipos de sensores, como acelerómetros y giroscopios. Este tipo de sensores se encuentran disponibles en dispositivos móviles Android® listos para utilizarse.

Android® permiten crear aplicaciones para acceder a sus recursos (como sensores de posición, movimiento o de ambiente) y así obtener ángulo de orientación o velocidad angular en cualquiera de los tres ejes del dispositivo móvil (ejes x, y y z). Así pues en este trabajo se introduce la idea de utilizar los recursos que ofrecen los dispositivos móviles basados en Android®, como señal de entrada a un controlador en Arduino®, el cual está montado en una plataforma experimental de tipo péndulo invertido con llantas.

1.1. Estado del arte

Los robots tipo péndulo invertido con llantas, en el marco de transporte personal como alternativa de transporte, ha crecido de manera rápida. En la figura 1.1 se muestra un robot WIP comercializado: el Segway® HT inventado por Dean Kamen, el cual es capaz de equilibrar a una persona parada en su plataforma mientras se mantiene en movimiento.

Existen diferentes modelos para los vehículos de autobalanceo que van desde los que utilizan una sola rueda sobre la cual mantienen el equilibrio como en el caso del llamado *Ballbot*, hasta los vehículos, como el Segway® que cuentan con dos ruedas y se basan en el

modelo péndulo invertido móvil con llantas (WIP). El presente documento se centra en los de la segunda categoría.
Los robots tipo péndulo invertido móvil tienen un cuerpo con dos ruedas para moverse y poder estabilizar la postura del cuerpo del robot en el eje vertical. La idea de un robot que pueda auto equilibrarse sobre sus ruedas ha sido un modelo muy estudiado debido a su no linealidad.

En los últimos 30 años se han realizado diferentes trabajos de investigación e implementación para estabilizar a los robots de dos ruedas por la versatilidad que tienen de operar en el medio ambiente. El primer péndulo invertido con llantas fue diseñado y construido en Tokio, Japón en 1986 [1]. Actualmente, una de las configuraciones comerciales más conocidas de este tipo de robots es el transporte personal Segway® (ver Figura 1.1), que en 2001 fue inventado y manufacturado por Dean Kamen. Este tipo de robots se denominan diferenciales ya que la dirección se logra a partir de la diferencia de velocidades entre ambas ruedas. Básicamente, el vehículo está conformado, mecánicamente, por una plataforma horizontal en la que el conductor se mantiene de pie con una rueda a cada costado; cuenta con un manubrio vertical para su manejo, el cual al momento de salir de la posición vertical avanza hacia adelante.

Figura 1.1: Transporte personal Segway®. Tomado de [13]

Para poder hacer el desarrollo de un robot WIP se requieren del estudio de varios aspectos: la estructura mecánica, los sensores, los lazos de control y los actuadores. Tomando como referencia el robot de transporte comercial Segway®, en el caso de los sensores, cuenta con un sensor especial de velocidad angular de estado sólido construido de silicio. Este tipo de giroscopio determina la rotación de un objeto utilizando el efecto Coriolis a una escala reducida. Adicionalmente el Segway® cuenta con cinco sensores giroscópicos que son redundantes. Se han realizado implementaciones de los sistemas móviles péndulo invertido, como en [2], en los que se utilizan giroscopios comerciales como sensores o bien inclinómetros que son de menor costo pero con un tiempo de respuesta lento.

En [3] se utilizan sistemas micro-electromecánicos (MEMS por sus siglas en inglés) como

giroscopio, acelerómetro y encoder. El giroscopio se encarga de la medición de la velocidad angular del balanceo del robot en tres ejes. El acelerómetro mide la aceleración total externa del robot incluyendo la aceleración gravitacional.

En los trabajos presentados en [4, 5, 6, 7] se muestra el inconveniente de trabajar con sistemas micro-electromecánicos como giroscopio e inclinómetro de bajo costo. Debido a la forma en que estos sensores trabajan, al obtener las señales se presenta ruido. Para trabajar de mejor manera las señales obtenidas por el inclinómetro y giroscopio son sometidas a filtros pasa bajas y pasa altas, respectivamente. Estos sensores, comúnmente utilizados, son sensores llamados *inerciales*. Se trata de un observador atrapado dentro de una esfera completamente blindada, tratando de determinar los cambios de posición de la esfera respecto a un sistema de referencia inercial exterior. Los sensores inerciales aprovechan las fuerzas inerciales que actúan sobre un objeto para determinar su comportamiento dinámico. Estos sensores funcionan utilizando transductores que convierten las fuerzas de inercia causadas por la aceleración de entrada o algunos cambios fisicos como deflexión de masas a una señal eléctrica [8] (ver Figura 1.2).

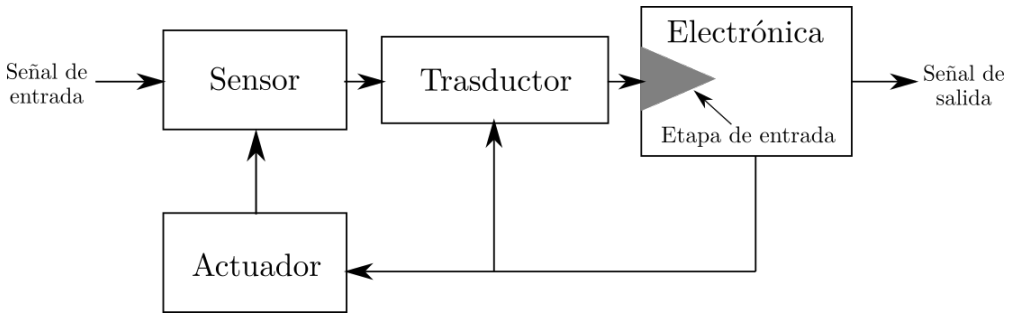


Figura 1.2: Arquitectura general del sistema de un sensor. Tomado de [8]

En cuanto a los dispositivos Android® como sensores alternos en implementaciones de robótica, en [9] se utiliza un dispositivo móvil Android® como cuerpo del péndulo y como sensores, sin embargo hace uso de una tarjeta especial llamada *IOIO-OTG* que se trata de una tarjeta de desarrollo para dispositivos móviles. Esta tarjeta busca agregar capacidades avanzadas de hardware a las aplicaciones Android®. Cuenta con un microcontrolador PIC y una biblioteca de nivel de aplicación.

En [20] se hacen uso de los sensores implementados en los dispositivos móviles para la estimación de la orientación. El giroscopio, acelerómetro y magnetómetro que incluyen los *smartphones* son los que proveen los datos para la estimación. Sin embargo, la estimación se realiza mediante diferentes algoritmos en MATLAB.

En [17] se hace uso de los sensores de un dispositivo móvil Android® para determinar el comportamiento de un conductor. Los sensores utilizados en [17] son los sensores de hardware: giroscopio, acelerómetro y magnetómetro. Esta implementación no hace uso de sensores externos para determinar si el comportamiento de un conductor es considerado como adecuado.

Además del uso que se le da a los dispositivos móviles en las implementaciones anteriores, normalmente los dispositivos móviles tienen papeles como dispositivos de control remoto en implementaciones de robótica.

Por otra parte, la tecnología de control que utiliza el medio de transporte comercial Segway® para llegar de un punto A a un punto B es mediante modos deslizantes. Sin embargo, en otras implementaciones como [3, 11, 12] se muestran controladores PID implementados en plataformas experimentales los cuales también pueden ser utilizados y trabajan satisfactoriamente. El objetivo general de la implementación de los lazos de control es que a partir de un ángulo de inclinación se genere una aceleración en las llantas como respuesta, haciendo que el sistema conmute entre dos estados: avanzar hacia atrás o adelante dependiendo del ángulo de inclinación fuera de la vertical.

1.2. Planteamiento del problema

El péndulo invertido es un problema clásico en la teoría de control ya que es un sistema inestable debido a que su centro de masa está por encima de su punto de pivote. Esta investigación tiene el fin de diseñar un sistema de control de balanceo usando sensores más económicos que los de los sistemas actuales. El auto-balanceo del Segway®, y en general de los robots de equilibrio, es posible gracias al lazo cerrado basado en sensores como el giroscopio, el inclinómetro y el acelerómetro con los que cuenta el transporte. En un sistema de control retroalimentado, los sensores permiten mantener una relación preestablecida entre la salida y alguna entrada de referencia, comparándolas y utilizando la diferencia como medio de control [22].

En el caso particular de los sistemas móviles péndulo invertido, la variable regulada corresponde a la posición vertical del sistema, y por consiguiente es común el uso de sensores de inclinación, aceleración angular y giroscopios como se mencionó con anterioridad. La mayoría de los transportes de dos llantas y más aún en el robot de transporte comercial Segway® son de elevados costos debido al precio de los sensores, las tarjetas de procesamiento de datos, entre otros.

Actualmente el costo del robot de transporte Segway® más económico está alrededor de los 4,000 pesos en México, aunque se trata de un modelo que no es reciente.

El vehículo Segway® es conformado por componentes y subsistemas, entre ellos el *conjunto sensorial de equilibrio* el cual integra el uso de cinco sensores de velocidad angular (giroscopios de estado sólido) y dos sensores de inclinación. La Figura 1.3 muestra la ubicación en el vehículo del conjunto sensorial de equilibrio.

Los sensores de estado sólido que se pueden hallar en el mercado tienen un precio que va de los 45 dólares a los 1200 dólares, dependiendo del fabricante y las características que se buscan del sensor. En el diseño propio de un vehículo de auto balanceo, el problema del alto costo de sensores inclinómetros puede resolverse al utilizar sensores inerciales de bajo costo que se encuentran disponibles en los dispositivos móviles.

Entre las características con las que cuentan los dispositivos móviles Android® (ver Figura 1.4) se encuentran los sensores que implementa el dispositivo, el módulo de comunicación Bluetooth®, pantalla para una interfaz gráfica con el usuario y la accesibilidad de desarrollo de aplicaciones para el sistema (entorno de desarrollo de software libre).

Para el desarrollo de una aplicación que mida la inclinación del dispositivo y pueda sustituir sensores inerciales de bajo costo, como los utilizados en implementaciones antes mencionadas, se hará un péndulo invertido móvil con llantas como plataforma experimental en la cual se utilizará el dispositivo móvil como sensor.

La implementación de un dispositivo móvil como sensor de orientación de una plataforma

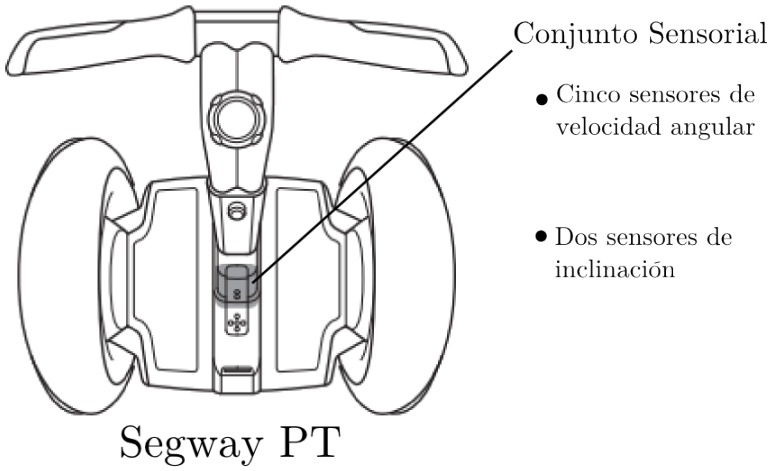


Figura 1.3: Control sensorial del vehículo Segway® TP. Tomado de [13]

tipo péndulo invertido móvil se divide en tres áreas de desarrollo:

1. Aplicación Android® para obtener la orientación del dispositivo móvil.
2. Control de auto-balanceo a partir de los datos obtenidos por el dispositivo móvil.
3. Estructura mecánica de la plataforma experimental.

1.3. Justificación

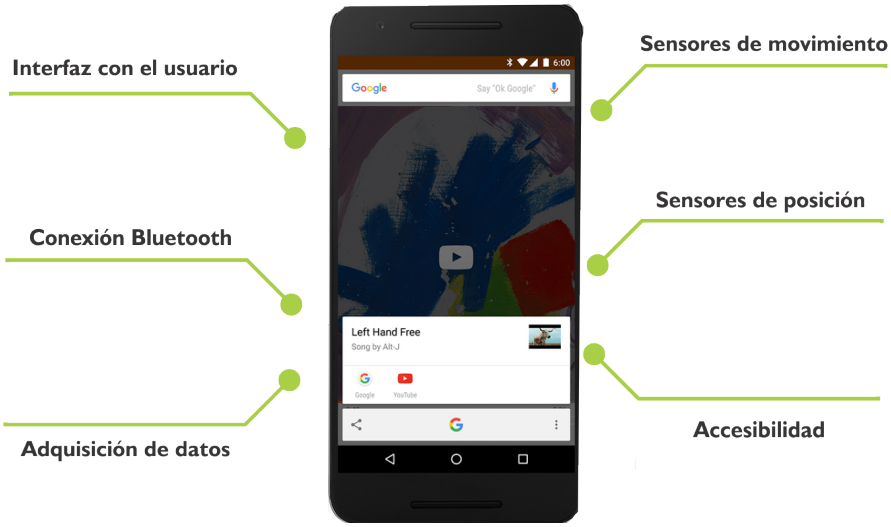


Figura 1.4: Características funcionales de los dispositivos móviles para su aplicación en el presente proyecto.

La presente investigación plantea el empleo de forma multidisciplinaria de los conocimientos académicos adquiridos en la carrera con el fin de aportar una posible solución a un problema latente en los sistemas de control de equilibrio. El aporte se refleja en el diseño e implementación de un sistema de control de equilibrio por medio del uso de un dispositivo móvil, el cual no está limitado a un único modelo, pues cualquier móvil que cumpla con los requisitos mínimos de software y hardware podrá establecer la comunicación con el sistema simplificado.

1.3.1. Pertinencia

El desarrollo de este trabajo abarca la implementación de los conceptos adquiridos durante los cursos de dinámica de sistemas, control, programación y microcontroladores. La aportación principal de este proyecto se enfoca, por un lado, a la obtención de la inclinación mediante sensores de bajo costo integrados en dispositivos móviles Android®, por otro lado, a la integración de electrónica, microelectrónica, programación y mecánica. Esto con el fin de aportar a la línea de investigación de control de sistemas mecatrónicos realizados en el Instituto de Electrónica y Mecatrónica. Además de aportar un enfoque de software en la obtención de la inclinación en un robot tipo Segway®.

1.3.2. Relevancia

La relevancia de esta propuesta se centra en la implementación de los sensores de bajo costo, los cuales se tiene al alcance como recursos de los dispositivos móviles que usamos continuamente. A partir del dispositivo móvil se podrán establecer las señales de entrada al sistema de control de balanceo del péndulo con el fin de disminuir del costo total de un transporte de auto balanceo.

1.4. Hipótesis

Los sensores implementados en los dispositivos móviles pueden utilizarse como sensores alternos para la obtención de la orientación de una plataforma experimental tipo péndulo invertido.

1.5. Objetivos

1.5.1. Objetivo general

Diseñar un inclinómetro usando los sensores de un dispositivo móvil basado en Android® para el balanceo de un robot de transporte tipo péndulo invertido móvil.

1.5.2. Objetivos específicos

- Diseñar un inclinómetro virtual competente a partir de los recursos que proporciona un dispositivo móvil.
- Modelar y parametrizar la plataforma experimental.
- Implementación de un algoritmo de control a partir de los datos obtenidos por el dispositivo móvil para el equilibrio de la plataforma experimental.

- Diseñar una aplicación para dispositivos Android® que mantenga comunicación Bluetooth® con el controlador de la plataforma experimental y funcione como parámetro de entrada al controlador.

1.6. Metas

- Modelado la plataforma experimental, parametrización y simulación de la misma.
- Diseño de lazos de control del sistema.
- Simulaciones de la plataforma experimental y lazos de control.
- Establecer comunicación Bluetooth® entre el dispositivo móvil y el controlador.
- Desarrollo de una aplicación Android® capaz de obtener la orientación del dispositivo móvil.
- Implementación de la aplicación y los lazos de control en la planta simulada y experimental.

1.7. Metodología de desarrollo

En el presente documento se describe el desarrollo e implementación de una plataforma experimental tipo péndulo invertido móvil con llantas, que se divide en tres áreas de desarrollo (ver Figura 1.5):

- Dispositivo móvil como sensor de orientación. Capítulo 2.
- Diseño del sistema de control encargado del auto-balanceo. Capítulo 3.
- Construcción de la plataforma experimental. Capítulo 4.

Emplear un dispositivo móvil como sensor de bajo costo aplicado a un robot de auto-balanceo, divide su desarrollo en tres tareas que se integran para obtener los objetivos planteados en la sección 1.5. La Figura 1.6 muestra un diagrama de la metodología utilizada en el desarrollo de este trabajo.

Los tres módulos de trabajo son: aplicación Android®, algoritmo de control y estructura mecánica, que se desarrollan de manera concurrente. Después del desarrollo de cada módulo se realiza la integración de los elementos como implementación de la plataforma experimental. Se realizan pruebas experimentales y dependiendo de los resultados obtenidos se rediseña alguno de los módulos. El desarrollo concurrente permite el rediseño de los módulos y que éstos mejoren su implementación en la plataforma como un conjunto.

1.8. Organización del documento de tesis

Como se mencionó en la metodología, la solución de utilizar dispositivos móviles como sensor de orientación se aborda con un diseño modular de tres áreas. En el capítulo 2 se explica el desarrollo de la aplicación en el dispositivo móvil, y se proporciona una introducción a los recursos que los dispositivos móviles Android® ofrecen y cómo acceder a ellos en el desarrollo

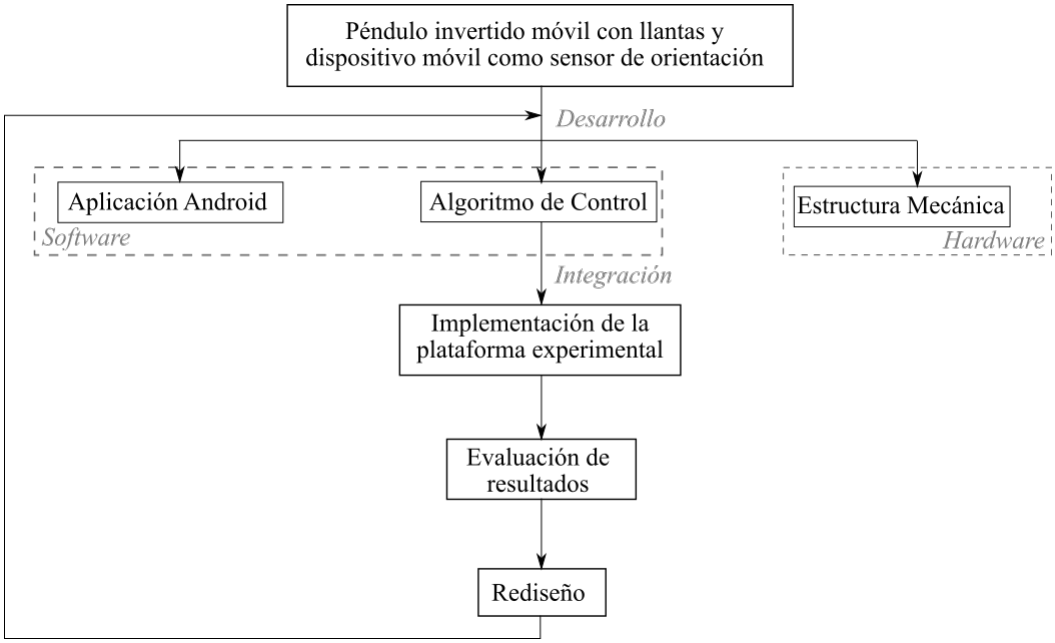


Figura 1.5: Diagrama a bloques de desarrollo para plataforma experimental

de una aplicación. El capítulo incluye la comunicación Bluetooth® en los dispositivos móviles, los sensores que se implementan en los dispositivos móviles y como son capaces de obtener la orientación del dispositivo mediante diferentes sensores. Se incluye el concepto *Sensor Fusion* en los dispositivos móviles. Se describen, también, las tareas que se deben realizar para enviar datos mediante protocolo de cominicación Bluetooth® y como se reciben los datos en la tarjeta Arduino®.

Para el desarrollo del algoritmo de control primero se obtiene el modelo dinámico de la plataforma. El capítulo 3 muestra el modelo dinámico de la plataforma experimental, el análisis de la respuesta del sistema en lazo abierto y lugar geométrico de las raíces. Además se presenta la simulación de la plataforma experimental con implementación de un controlador de tipo PID y cómo actúa en lazo cerrado. Se incluye la implementación del algoritmo de control en la tarjeta Arduino®.

En el capítulo 4 se muestra el desarrollo de la estructura mecánica para la plataforma experimental. El hardware en el que se implementan los módulos de software (tanto el algoritmo de control como la aplicación de orientación) se describe en este capítulo, así como los elementos electrónicos que conforman la plataforma experimental y la implementación de todos los módulos para obtener una plataforma experimental final.

Por último, el capítulo 5 presenta los resultados obtenidos de las pruebas experimentales y el análisis de cada uno de ellos. El capítulo 6 presenta las conclusiones finales del presente trabajo de tesis y las propuestas de trabajo futuro.

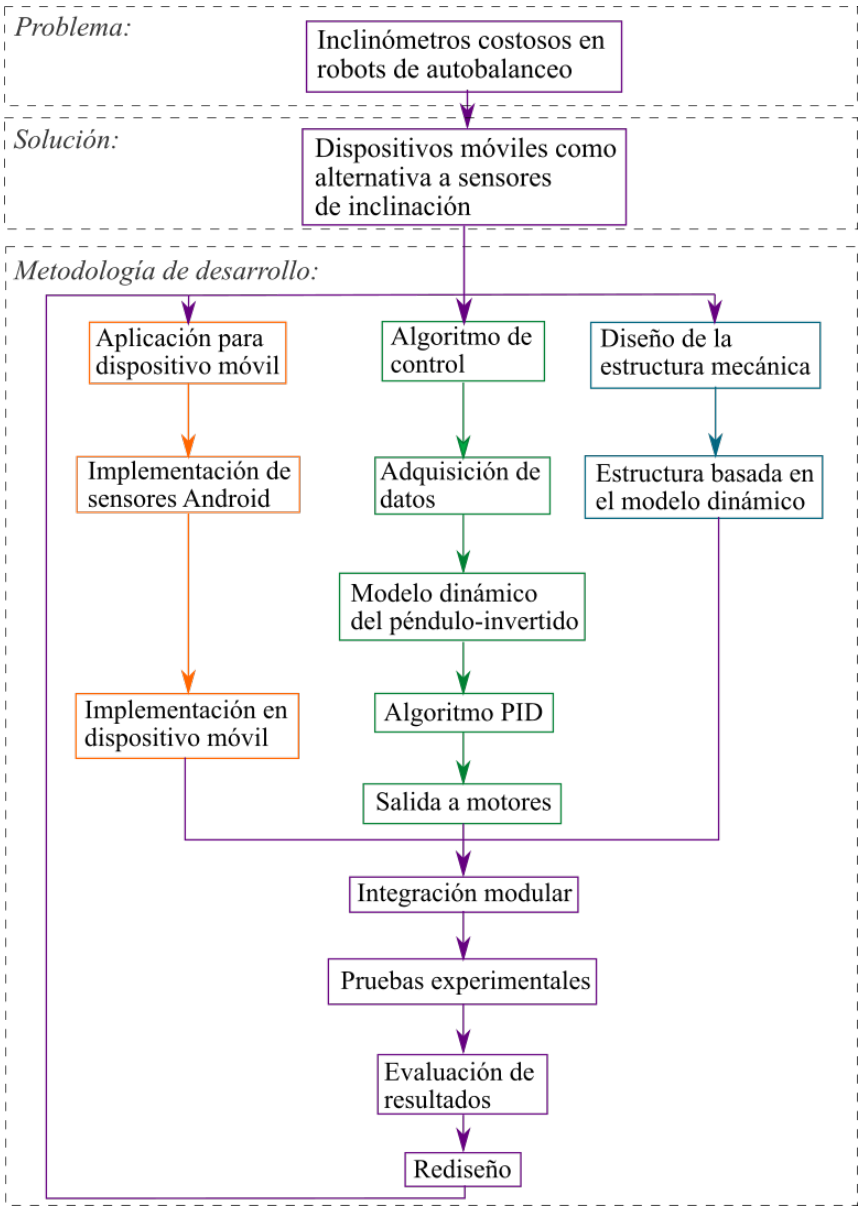


Figura 1.6: Metodología de desarrollo para plataforma experimental péndulo invertido móvil.

Capítulo 2

Dispositivo móvil como sensor de orientación

Android® es un sistema operativo basado en el núcleo de Linux. La Figura 2.1 muestra la arquitectura de Android®[14]. Los desarrolladores de aplicaciones para este sistema operativo trabajan, generalmente, en las capas:

- Aplicación.
- Marco de referencia de la aplicación.
- Bibliotecas.
- Tiempo de ejecución.
- Núcleo de Linux.

Cuando se desarrolla en Android® y dependiendo del tipo de aplicación desarrollada y los recursos que se utilicen, se definen diferentes métodos. Para poder ejecutar la aplicación es necesario sobrescribir tres métodos que describen el comportamiento de la aplicación al ser inicializada. Estos métodos se explican a continuación:

- onCreate: Indica los recursos que va a utilizar la actividad o lo primero que ejecutará. Normalmente, aquí se invoca a la interfaz gráfica principal con la que el usuario interactúa.
- onResume: Indica las tareas que ejecuta la aplicación al interactuar con el usuario.
- onPause: Indica las tareas de ejecución cuando el usuario ha salido de la aplicación, o bien la aplicación no es el foco del usuario.

Existen algunos otros métodos que deben ser definidos dependiendo de los recursos que se estén utilizando. Por ejemplo, en el caso de utilizar sensores (cualquier tipo de ellos), deben sobrescribirse los métodos: *onSensorChanged* y *onAccuracyChanged*.

Figura 2.1: Arquitectura general del sistema operativo Android®. Tomado de [14]

2.1. Entorno de desarrollo para Android®

Android Studio es el entorno de desarrollo integrado oficial para la plataforma Android®. Anunciado el 16 de mayo de 2013 en la conferencia Google® y reemplazó a Eclipse® como el IDE oficial para el desarrollo de aplicaciones para Android®.

Está basado en el software IntelliJ IDEA de JetBrains y ha sido publicado de forma gratuita a través de la Licencia Apache 2.0. Se encuentra disponible para las plataformas Microsoft Windows, macOS y GNU/Linux y ha sido diseñado específicamente para el desarrollo de Android®. La versión en la que se desarrolla la aplicación para el presente trabajo de tesis es Android Studio 2.2 con versión de Java Development Kit (JDK) 8 [14]. La Figura 2.2 muestra la interfaz de la ventana principal del entorno de desarrollo Android Studio®.

La última versión estable de Android Studio® es la versión 3.1.2 lanzada el 1 de abril de 2018. Programada en Java® y escrito en Java® y Kotlin®.

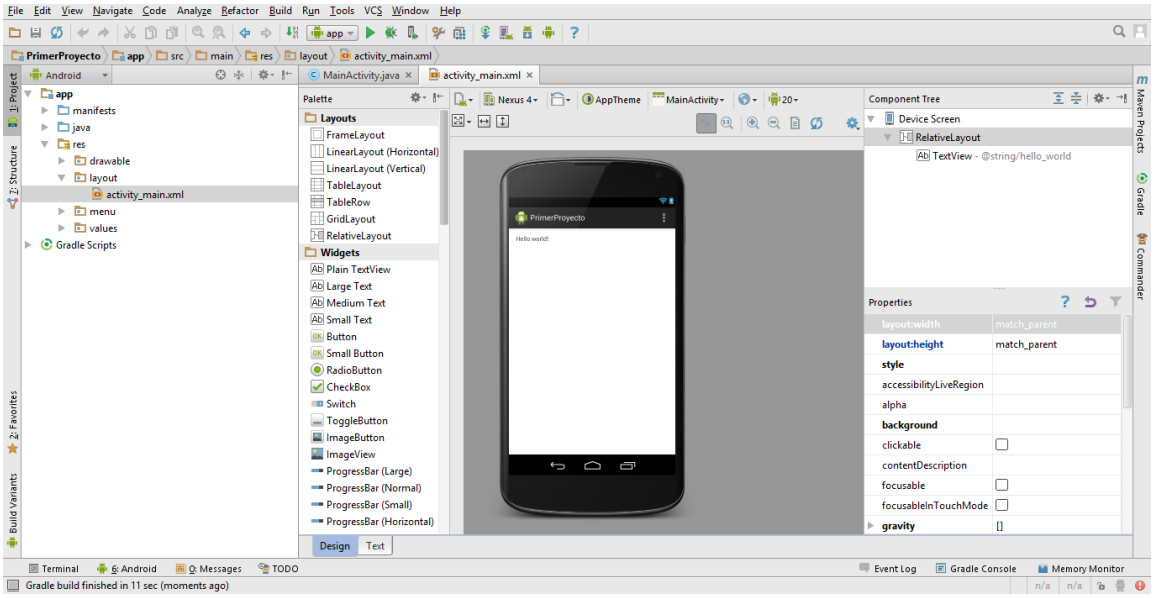


Figura 2.2: Ventana principal de Android Studio®.

En este capítulo se presenta el desarrollo de la aplicación para la obtención de la orientación del dispositivo móvil y la recepción de datos en la tarjeta Arduino®. Primeramente se presenta el desarrollo en Android® para la aplicación del dispositivo móvil y posteriormente la recepción de estos datos en la tarjeta Arduino® Nano mediante el protocolo de comunicación Bluetooth (ver figura 2.3).

El desarrollo del inclinómetro virtual divide su implementación en dos etapas: una en el dispositivo móvil y otra en la tarjeta Arduino®. El diagrama a bloques de la figura 2.3 muestra el funcionamiento del inclinómetro. La primera parte de la implementación se hace en el dispositivo móvil Android®, en donde primero se declaran los sensores a utilizar, se obtiene la orientación del dispositivo mediante los sensores de Android® y se envían los datos mediante comunicación Bluetooth al controlador. La segunda parte del inclinómetro se encuentra en Arduino®, donde se reciben los datos enviados por el dispositivo móvil y se convierten de una cadena de caracteres a un número flotante, como entrada al controlador.

2.2. Desarrollo de la aplicación en dispositivo móvil Android®

Para el desarrollo de la aplicación primero se define cual es la secuencia en la que la aplicación se ejecuta, tanto interactuando con el usuario como internamente en las capas de Android®.

La función del dispositivo móvil es ser una alternativa a los sensores MEMS de bajo costo que diferentes implementaciones utilizan(como se menciona en la sección 1.1), por lo cual, la aplicación desarrollada debe obtener la orientación del dispositivo y enviarla a la tarjeta Arduino® para lograr el autobalanceo de la plataforma experimental. Para cumplir con los objetivos descritos se consideran las siguientes tareas:

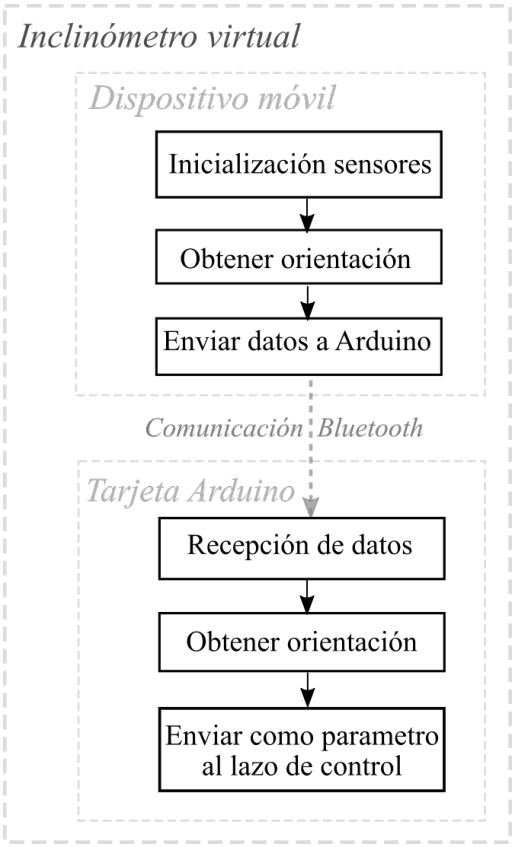


Figura 2.3: Diagrama a bloques del funcionamiento del inclinómetro desarollado.

1. Establecer comunicación con el módulo Bluetooth® de la tarjeta Arduino®.
2. Acceder y procesar datos de los sensores para obtener la orientación del dispositivo móvil.
3. Envío de datos del sensor a la tarjeta Arduino®.
4. Recepción de datos en la tarjeta Arduino®.

El objetivo principal de la aplicación es medir la orientación del dispositivo móvil sobre el eje y de forma que esta señal se utilice como entrada a un sistema de control. Para que los datos obtenidos por los sensores del dispositivo móvil cumplan con el objetivo de control, los sensores que se utilicen deben ser capaces de muestrear rápidamente los cambios en la orientación del dispositivo sin que esto genere valores con muchas variaciones. La Figura 2.4 muestra el diagrama a bloques de la estructura general de ejecución que la aplicación debe seguir: La aplicación se ejecuta mediante dos actividades distintas: *DeviceList* y *MainActivity*. La primera que se encarga de establecer la conexión entre el dispositivo móvil y el módulo bluetooth de la tarjeta Arduino®. En esta actividad el usuario inicia la comunicación con el módulo Bluetooth® de la tarjeta Arduino®. La segunda actividad se encarga de la lectura de los sensores y el procesamiento de estos datos para la obtener la orientación. En esta actividad también se hace el envío de datos a la tarjeta Arduino®, ver figura 2.4.

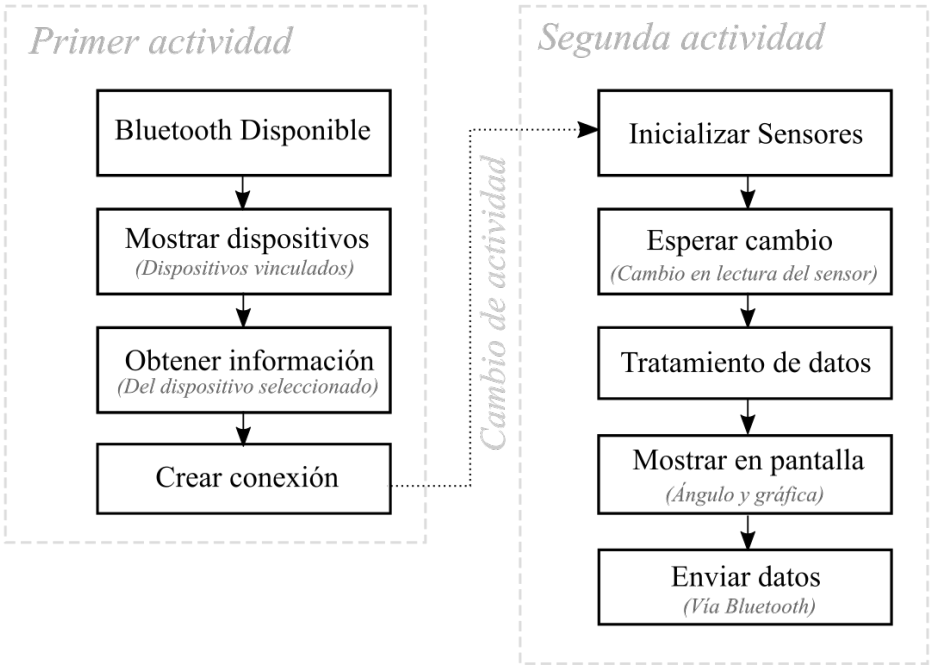


Figura 2.4: Diagrama a bloques de la aplicación para el dispositivo móvil.

2.2.1. Enlace de comunicación Bluetooth® entre dispositivos

Para que un dispositivo pueda considerarse como Bluetooth® tiene que cumplir una serie de protocolos y perfiles:

- Protocolos: Describen cómo se realizan las tareas básicas como señalización telefónica, gestión de enlace y lo que se conoce como *Service Discovery*.
- Perfiles: Describen la forma en que diferentes protocolos y procedimientos básicos funcionan conjuntamente en diferentes productos y aplicaciones Bluetooth®.

Las señales de radio transmitidas con Bluetooth® cubren sólo distancias cortas, normalmente hasta 10 metros. Bluetooth® fue diseñado originalmente para conexiones inalámbricas de menor velocidad, aunque los avances tecnológicos en años recientes han aumentado considerablemente su rendimiento. Las primeras versiones de las conexiones estándar tenían una velocidad menor a 1 Mbps mientras que las versiones modernas soportan hasta 25 Mbps [10].

La tecnología Bluetooth® es un sistema de comunicación de corto alcance, diseñado específicamente para reemplazar a los cables que conectan equipos fijos y portátiles entre sí. Las características principales de éste tipo de tecnología inalámbrica se centra en su robustez y el bajo consumo de potencia [10].

Bluetooth® es un estándar inalámbrico muy flexible permitiendo descubrir, conectar y transferir información a través de redes en miniatura par a par. Como parte del proceso de descubrimiento y conexión, la pila de Bluetooth® se basa en un protocolo llamado *descubrimiento de servicios* (SDP) para recopilar información sobre los dispositivos que va descubriendo a determinar si tienen las capacidades adecuadas para justificar la conexión.

Como parte del SDP, los dispositivos deben publicar un UUID para cada servicio que tienen disponible.

Dos dispositivos Bluetooth® se conectan entre sí mediante un proceso llamado emparejamiento. Para las comunicaciones seriales punto a punto está el protocolo RFCOMM, que trabaja emulando el protocolo RS-232 a través de RF y mediante el protocolo L2CAP, el protocolo RFCOMM de Bluetooth es a menudo denominado emulación de puertos serie.

A fin de usar las funciones de Bluetooth® en la aplicación, se necesita permiso para establecer cualquier comunicación de Bluetooth®, como solicitar o aceptar una conexión y transferir datos. La declaración del permiso Bluetooth® en Android® se muestra en el siguiente código:

```
1 <manifest ... >
2   <uses-permission android:name= ''android.permission.BLUETOOTH"/>
3 </manifest>
```

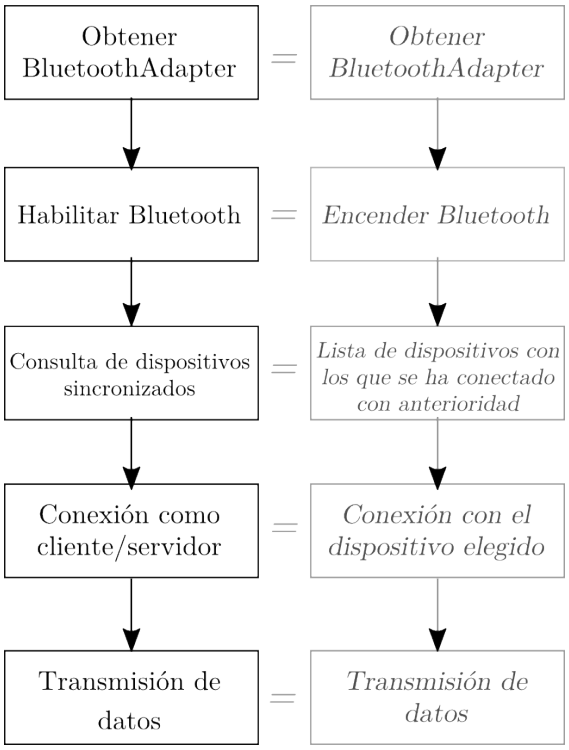


Figura 2.5: Diagrama a bloques de las tareas para realizar una conexión Bluetooth® en dispositivos Android®.

Para realizar una conexión Bluetooth® entre dispositivos, ver Figura 2.5, en Android® se deben realizar las siguientes tareas:

1. Obtener el adaptador Bluetooth del dispositivo.
2. Habilitar Bluetooth.
3. Consulta de dispositivos sincronizados.
4. Conexión como cliente.
5. Abrir conexión para transmitir datos.

1. Obtener el adaptador Bluetooth del dispositivo

El *BluetoothAdapter* es obligatorio para toda actividad de Bluetooth. Para obtener el *BluetoothAdapter*, se llama al método estático *getDefaultAdapter()*, éste regresa un *BluetoothAdapter* que representa el propio adaptador de Bluetooth del dispositivo, como se observa en el código siguiente:

```
1 BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
2 if (mBluetoothAdapter == null) {
3     //Acciones si no se soporta Bluetooth
4 }
```

2. Habilitar Bluetooth

Posteriormente, se asegura que Bluetooth esté habilitado mediante *isEnabled()*. Si el Bluetooth no está habilitado (encendido), se puede mostrar un cuadro de diálogo que solicite habilitarlo. Para solicitar la habilitación de Bluetooth se utiliza el método *startActivityForResult()*, que genera el cuadro de diálogo de solicitud de habilitación de Bluetooth, como se muestra en las siguientes líneas de código:

```
1 if (!mBluetoothAdapter.isEnabled()) {
2     Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE
          );
3     startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
4 }
```

El código anterior genera un cuadro de solicitud como se muestra en la figura 2.6. Al iniciar la aplicación, si el usuario no tiene habilitado el Bluetooth en su dispositivo se muestra una solicitud para encenderlo y continuar ejecutando la aplicación.

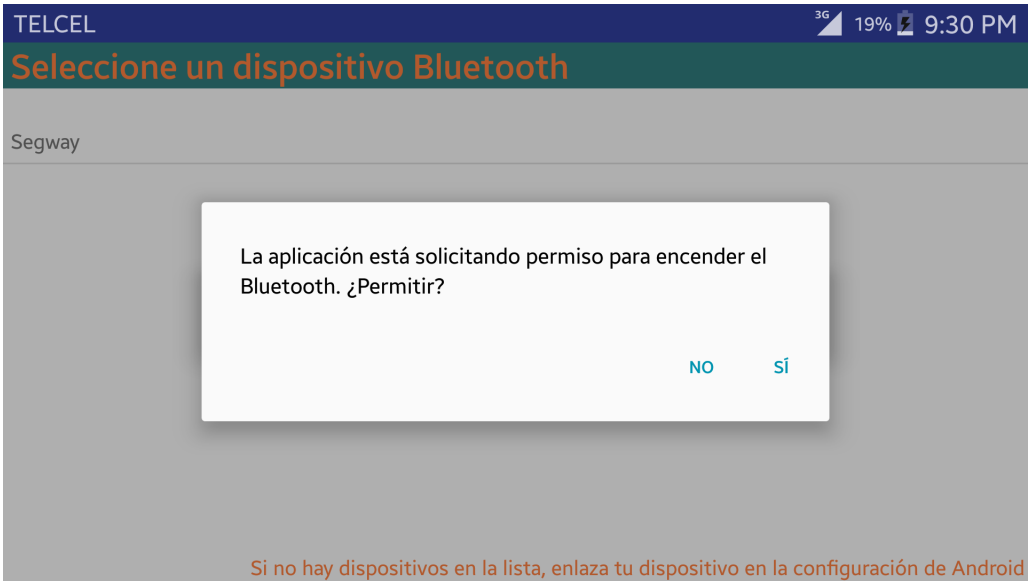


Figura 2.6: Diálogo de solicitud de habilitación de Bluetooth® de la aplicación Android

3. Busqueda de dispositivos sincronizados

El *BluetoothAdapter* es capaz de buscar dispositivos Bluetooth remotos por medio de la detección de dispositivos o la consulta de la lista de dispositivos sincronizados. La detección de dispositivos es un procedimiento de escaneo que busca en el área local dispositivos con Bluetooth habilitado y solicita información sobre uno de ellos. La actividad *DeviceList* es la primera interfaz con la que el usuario se encuentra y en la que es capaz de elegir el dispositivo al cual se va a conectar.

La diferencia entre un dispositivo conectado y uno vinculado, para enlazar dispositivos entre sí, recae en la dependencia del primero con el segundo. Para que un dispositivo pueda conectarse con otro deben mantener entre ellos un vínculo. Un dispositivo móvil es capaz de almacenar el nombre y dirección MAC de los dispositivos con los que se ha vinculado con anterioridad.

Antes de llevar a cabo la detección de dispositivos, es importante consultar el conjunto de dispositivos sincronizados a fin de ver si el dispositivo deseado ya es conocido. Para ello se llama al método *getBondedDevices()*. Ésto mostrará un conjunto de *BluetoothDevice* que representa los dispositivos sincronizados. La implementación de estas tareas se muestran en el siguiente fragmento de código:

```
1 Set<BluetoothDevice> pairedDevices = mBluetoothAdapter.getBondedDevices();
2 //Si son dispositivos emparejados
3 if (pairedDevices.size() > 0) {
4     //Lista de dispositivos
5     for (BluetoothDevice device : pairedDevices) {
6         //Muestra el nombre y direccion MAC del dispositivo
7         mArrayAdapter.add(device.getName() + "\n" + device.getAddress());
8     }
9 }
```

En el diagrama a bloques de la Figura 2.4 se muestra la primera actividad que la aplicación ejecuta: el enlace entre dispositivos. El usuario primero define con que dispositivo va a compartir la información que los sensores obtengan y que el resto del tiempo de ejecución muestree sin interrupciones.

La primera actividad que la aplicación debe realizar es mostrar al usuario los nombres de los dispositivos con las que su Bluetooth se ha vinculado para elegir con cual conectarse. La Figura 2.7 muestra la interfaz gráfica de la aplicación en la cual el usuario puede elegir con cual dispositivo (con los que ya se ha vinculado) desea conectarse. En la interfaz se muestra el nombre y dirección MAC de los dispositivos con los que se puede conectar el usuario.

Figura 2.7: Interfaz gráfica de la aplicación que muestra dispositivos vinculados.

La aplicación primero muestra los dispositivos con los que se ha emparejado. Cuando se ha seleccionado un dispositivo para conectar, se obtienen nombre y dirección MAC en una sola cadena. La longitud de la dirección MAC de cualquier dispositivo es de 17 caracteres, éste es el parámetro que se utiliza para abrir un canal de comunicación entre los dos dispositivos que permita enviar y recibir información.

En el apéndice B se muestra el código en Android para visualizar los dispositivos enlazados.

4. Conexión del dispositivo móvil como cliente

La conexión Bluetooth de un dispositivo móvil puede realizarse como cliente o servidor. En la conexión entre un dispositivo móvil y el módulo Bluetooth HC-05, el dispositivo móvil realiza una conexión como cliente mientras el módulo mantiene una conexión como servidor.

A fin de inicializar una conexión con un dispositivo remoto (un dispositivo que mantenga un *socket* de servidor abierto), primero se obtiene un objeto *BluetoothDevice* que represente el dispositivo remoto, en este caso específico el módulo HC-05. Posteriormente se usa el *BluetoothDevice* para obtener un *BluetoothSocket* e inicializar la conexión. Usando el *BluetoothDevice*, se obtiene un *BluetoothSocket* mediante el método *createRfcommSocketToServiceRecord(UUID)*, esto inicializa un *BluetoothSocket* que se conectará al *BluetoothDevice*. El

UUID debe coincidir con el UUID empleado por el dispositivo del servidor cuando abrió su *BluetoothServerSocket*.

La comunicación entre los dispositivos se realiza sobre un hilo propio, por lo cual, luego de tener el socket creado, se le asigna un hilo. Como se muestra en el siguiente fragmento de código:

```
1//Conexion entre dispositivos
2  //Se crea un adaptador Bluetooth
3  BTAdapter = BluetoothAdapter.getDefaultAdapter();
4  //Se asigna al adaptador remoto la direccion MAC
5  BluetoothDevice device = BTAdapter.getRemoteDevice(address);
6  //Se crea el socket RFCOMM al dispositivo
7  UUID BTMODULEUUID = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");
8  device.createRfcommSocketToServiceRecord(BTMODULEUUID);
9  //Establecer conexion entre dispositivos
10 BTSocket.connect();
11 //Crear el hilo nuevo al socket
12 mConnectedThread = new ConnectedThread(BTSocket);
13 //Inicializacion del hilo
14 mConnectedThread.start();
```

En el apéndice C muestra el código en Android® para inicializar comunicación Bluetooth entre dispositivos.

2.2.2. Interfaz gráfica de la aplicación de orientación

La Figura 2.8 muestra la interfaz gráfica de la segunda actividad de la aplicación. En esta actividad se declaran los elementos gráficos que el usuario puede ver en pantalla, en la cual se encuentran los siguientes elementos:

- Botón de comunicar.
- Botón para cambiar ganancias.
- Cajas de texto para nuevas ganancias.
- Texto que muestra la orientación del dispositivo.

Una vez que se ha elegido el dispositivo con el que se va a conectar y al cual se envían los datos obtenidos por el sensor, el trabajo de la aplicación es obtener la orientación del dispositivo, enviarla mediante comunicación Bluetooth® y mostrarla al usuario en pantalla.

Debido a la orientación en la que el dispositivo se coloca en la plataforma (ver capítulo 4), primero se bloquea la orientación en la que la aplicación se visualiza, para que el movimiento no provoque la rotación de la pantalla. Se pueden definir dos orientaciones diferentes: *LANDSCAPE* para cuando la aplicación desea verse de forma horizontal o bien *PORTRAIT* para verla verticalmente. Esto se logra con la siguiente línea de código:

```
1 setRequestedOrientation(
2 ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
```

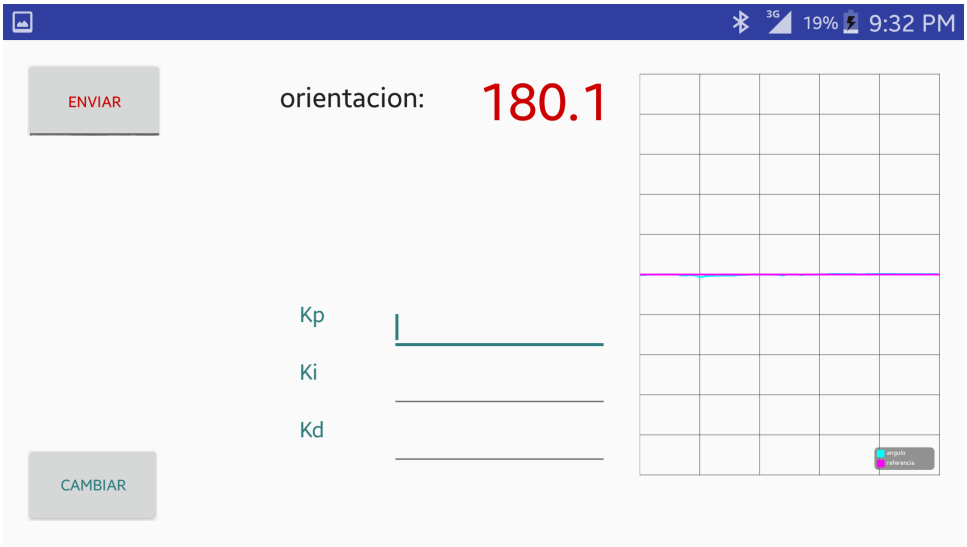


Figura 2.8: Vista principal de la aplicación.

En esta segunda actividad primero se declaran los elementos que se visualizan en pantalla:

```
//Declaracion de elementos en pantalla
 //Para las ganancias
kpA = (EditText) findViewById(R.id.kpU);
kiA = (EditText) findViewById(R.id.kiU);
kdA = (EditText) findViewById(R.id.kdU);
//Boton de comunicar
comunicar = (ToggleButton) findViewById(R.id.tbComunicar);
//Boton para enviar los cambios de ganancias
cambios = (Button) findViewById(R.id.bGanancias);
//texto que muestra el angulo medido
textViewX = (TextView) findViewById(R.id.voX);
```

El botón de ***comunicar*** es un tipo de botón de enclavamiento, funciona como una bandera que indicará cuando empezar a enviar datos. Si el botón ha sido presionado se queda en estado 1 hasta que vuelva a ser presionado para cambiar a estado 0.

Por su parte, el botón de ***cambios*** se encarga de realizar una tarea específica cuando es presionado. Dicha tarea consiste en tomar los datos de las ganancias y enviarlos mediante comunicación Bluetooth® a Arduino®. Para indicar que se envían nuevas ganancias primero se manda un caracter especial y posteriormente una cadena que contenga las nuevas ganancias separadas por un caracter especial. El siguiente código ilustra la tarea descrita.

```
cambios.setOnClickListener(new View.OnClickListener() {
  public void onClick(View v) {
   //Se envia un caracter especial que indique nuevas ganancias
   mConnectedThread.write("!");
   //Se convierte el numero de ganancia kp a una cadena
   String gkp = kpA.getText().toString();
   //Se convierte el numero de ganancia ki a una cadena
   String gki = kiA.getText().toString();
   //Se convierte el numero de ganancia kd a una cadena
   String gkd = kdA.getText().toString();
```

```
11      //Se envian las ganancias separadas por caracteres especiales
12      mConnectedThread.write(gkp + "$" + gki + "$" + gkd + "$"+"!");
13    }
14});
```

2.2.3. Sensores y tratamiento de datos

El marco de referencia de cada sensor implementado en los dispositivos móviles utiliza un estándar de sistema de coordenadas de 3 ejes para adquirir los datos del móvil. En la Figura 2.9 se muestra el sistema de referencia de coordenadas de un dispositivo móvil, independientemente del sensor que se esté implementando, el cual no cambia si la orientación de la pantalla es horizontal o vertical. Para acceder a los datos que el sistema adquiere de los sensores implementados, se accede al vector de datos proporcionado. La mayoría de los sensores obtienen datos en cualquiera de los tres ejes (x, y, z), almacenándolos en un arreglo donde la primera posición guarda el dato correspondiente al eje x y así sucesivamente.

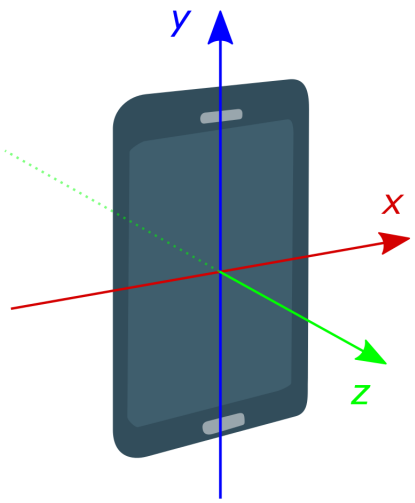


Figura 2.9: Sistema de coordenadas x, y, z en dispositivos móviles Android®.

El punto más importante para entender el sistema de coordenadas es que los ejes no se intercambian cuando cambia la orientación de la pantalla del dispositivo, es decir, el sistema de coordenadas del sensor nunca cambia a medida que el dispositivo se mueve. La mayoría de los dispositivos móviles con Android® han incorporado en ellos sensores que miden el movimiento, la orientación y otras condiciones ambientales. Estos sensores son capaces de proporcionar datos en bruto con alta precisión y exactitud. Los dispositivos móviles Android® cuentan con diferentes sensores que pueden ser clasificados en tres categorías: sensores de posición, movimiento y entorno. Una clasificación diferente es de sensores basados en software o hardware. Los sensores basados en hardware son componentes físicos incorporados en un dispositivo móvil, como micro mecanismos electro-mecánicos (MEMS), cuyos datos se derivan de mediciones directas a las propiedades físicas. Los sensores basados en sofware, por el contrario,

derivan sus datos de uno o más de los sensores basados en hardware implementando algún tipo de procesamiento de datos y son también conocidos como sensores virtuales o sintéticos.

La segunda actividad de la aplicación se encarga de obtener la orientación mediante los sensores de Android® y enviar los datos al dispositivo conectado. Una vez declarados los elementos gráficos de la aplicación se declaran los sensores a utilizar, como se hace en el siguiente fragmento de código:

```
1 //Declaracion de diferentes sensores
2 //Se obtiene el servicio de sensor
3 mSensorManager=(SensorManager) getSystemService(Context.SENSOR_SERVICE);
4 //Se declaran los sensores que se utilizan
5 rotacion=mSensorManager.getDefaultSensor(Sensor.TYPE_ROTATION_VECTOR);
6 acelerometro=mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
7 magnetometro=mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
8 giroscopio=mSensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
```

La velocidad de muestreo de los sensores depende del retardo que se le asigne a cada tipo de sensor. El retardo de cada sensor puede ser de uno de los siguientes cuatro tipos:

- *GAME*: 20 ms.
- *FASTEST*: 0 ms.
- *NORMAL*: 200 ms.
- *UI*: 60 ms.

El siguiente fragmento de código ejemplifica la declaración del uso del sensor de vector de rotación con retardo de tipo *game*:

```
1 //Se obtiene el servio de sensor
2 mSensorManager=(SensorManager) getSystemService(Context.SENSOR_SERVICE);
3 //Se declaran los sensores que se utilizan
4 rotacion=mSensorManager.getDefaultSensor(Sensor.TYPE_ROTATION_VECTOR);
5 //Se define el retardo que tendra en el muestreo cada sensor
6 mSensorManager.registerListener(this,rotacion,SensorManager.
      SENSOR_DELAY_GAME);
```

La orientación de un dispositivo Android® puede ser medida mediante sensores de movimiento o de posición. La orientación del dispositivo respecto al eje y puede obtenerse de diferentes formas y con diferentes sensores, ya sea mediante sensores basados en hardware como el *giroscopio*, *magnetómetro* y *acelerómetro*. O bien mediante sensores basados en software como el sensor de *vector de orientación lineal*. Ver Figura 2.10.

De los sensores basados en hardware: Los acelerómetros miden cualquier aceleración y devuelven un vector en el marco de referencia del dispositivo.
Los giroscopios miden las rotaciones y devuelven un vector de rotación angular también en el marco de referencia del dispositivo. La salida del giroscopio es bastante suave y muy sensible a pequeñas rotaciones. Los magnetómetros miden campos magnéticos y devuelven un vector correspondiente al campo magnético acumulativo debido a los imanes cercanos (incluida la Tierra).

La representación de la orientación, generalmente se representa mediante cuaterniones, ideales para rotaciones en tres dimensiones.

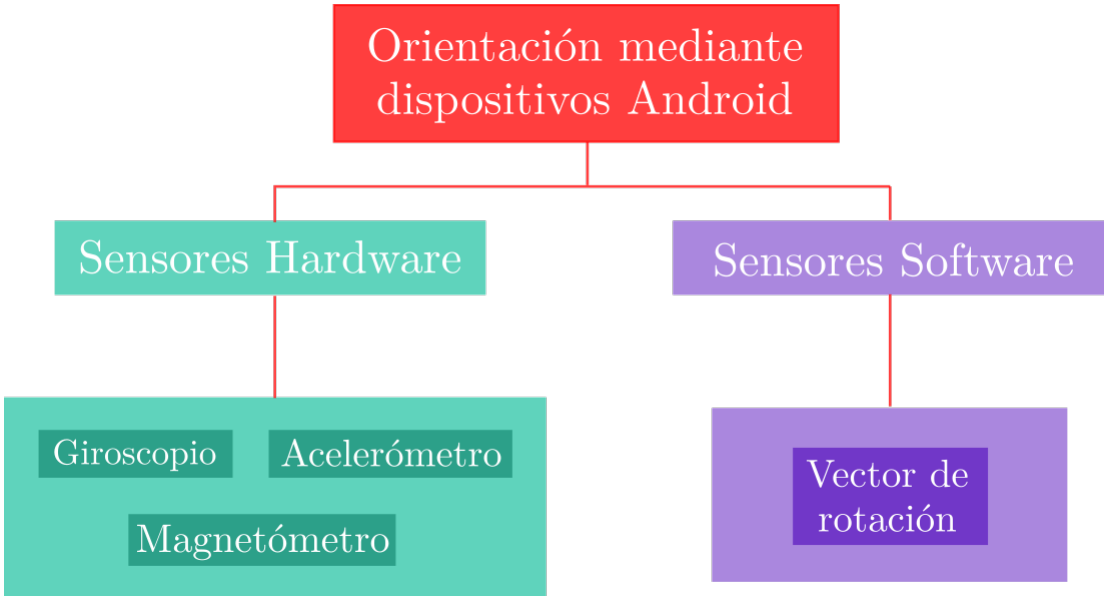


Figura 2.10: Sensores Android® para medir la orientación del dispositivo.

Se considera q un cuaternión si es de la forma:

$$q = s + xi + yj + zk, \tag{2.1}$$

en donde $s, x, y, z \in \mathbf{R}$.

Los cuaterniones son una extensión generada de manera análoga añadiendo las unidades imaginarias i, j y k a los números reales tal que: $i^2 = j^2 = k^2 = ijk = -1$.

2.2.3.1. Fusión de sensores en dispositivos móviles

Las mediciones que obtienen los sensores inerciales presentan errores, ya sea de tipo humano cuando se hace la medición, de retardo o de ruido. Para dar una solución a esto y obtener las mediciones deseadas se hace un procesamiento de las señales para corregir los posibles errores como filtros y *fusión de sensores*.

Cuando se utilizan dos o más sensores, se introduce el concepto de *fusión de sensores* para aprovechar las fortalezas de cada sensor y amortiguar las debilidades. Una manera de obtener la inclinación del dispositivo mediante sensores basados en hardware es mediante el enfoque de fusión de sensores. Ya sea utilizando un giroscopio, acelerómetro o magnetómetro, se requiere hacer algún tipo de procesamiento al combinar las señales de diferentes sensores.

Los giroscopios no tienen idea de donde se encuentran en relación con el mundo, mientras que los acelerómetros son muy ruidosos y no pueden proporcionar una estimación de orientación. La idea de la fusión de los sensores es tomar lecturas de cada sensor y proporcionar un resultado más útil que combine los puntos fuertes de cada sensor. La corriente fusionada resultante es mayor que la suma de sus partes.

A pesar de que la fusión de sensores puede ser desarrollada por uno mismo, los dispositivos desde 2010 que incluyen sensores InvenSense, probablemente también incorporen los algoritmos *Sensor Fusion* de la misma compañía. La forma en la que estos algoritmos trabajan

es obteniendo datos de orientación a partir de datos integrados del giroscopio pero amortiguan los errores comparando constantemente los datos con información del acelerómetro y magnetómetro [15].

2.2.3.2. Orientación mediante sensores basados en software

Uno de los sensores que provee Android® es el sensor de vector de rotación. Éste es un tipo de un sensor sintético (basado en software) que hace uso del acelerómetro, el magnetómetro y el giroscopio para producir información de orientación del dispositivo. Los tres elementos que el vector de rotación entrega, se expresan de la siguiente manera:

$$x \sin(\theta/2) \tag{2.2}$$

$$y \sin(\theta/2) \tag{2.3}$$

$$z \sin(\theta/2) \tag{2.4}$$

El vector de rotación representa la orientación del dispositivo como una combinación de un ángulo y un eje, en el que el dispositivo ha girado un ángulo θ alrededor de uno de sus ejes.

La Figura 2.11 muestra el diagrama a bloques de las tareas que se deben realizar para obtener la orientación del dispositivo mediante sensores basados en software. El vector de rotación devuelto por el sensor se puede convertir en una matriz de rotación con una llamada a *SensorManager.getRotationMatrixFromVector()* y la matriz de rotación resultante se puede pasar a *SensorManager.getOrientation ()* para obtener la orientación real del dispositivo.

El siguiente fragmento de código realiza las tareas antes mencionadas para obtener la orientación mediante el sensor basado en software que Android® incluye:

```
1public final void onSensorChanged (SensorEvent event) {
2    if (event.sensor.getType() == Sensor.TYPE_ROTATION_VECTOR) {
3         // convertir el vector de rotacion a matriz 4x4
4        SensorManager.getRotationMatrixFromVector(rotationV, event.values);
5        SensorManager.remapCoordinateSystem(rotationV, SensorManager.AXIS_Y,
             SensorManager.AXIS_X, adjustedRotationMatrix);
6        SensorManager.getOrientation(adjustedRotationMatrix, orientation);
7        orientation2[0] = (float) Math.toDegrees(orientation[0]);//radianes a
             grados
8        orientation2[1] = (float) Math.toDegrees(orientation[1]);//radianes a
             grados
9        orientation2[2] = (float) Math.toDegrees(orientation[2]);//radianes a
             grados
10       showResults(orientation2[1]);
11   }
12}
```

El código anterior muestra cómo obtener la orientación del dispositivo mediante el sensor sintético que provee Android®. La gráfica de la Figura 2.12 muestra la orientación obtenida mediante el sensor basado en software que provee Android®. La línea azul representa la referencia de 180° que el dispositivo debería sensar cuando se encuentra en posición con la pantalla hacia arriba y sin moverse. La línea roja muestra la orientación obtenida por el sensor vector de rotación. Se puede observar que el sensor obtiene una medición con un margen de error de $\pm 0.4°$.

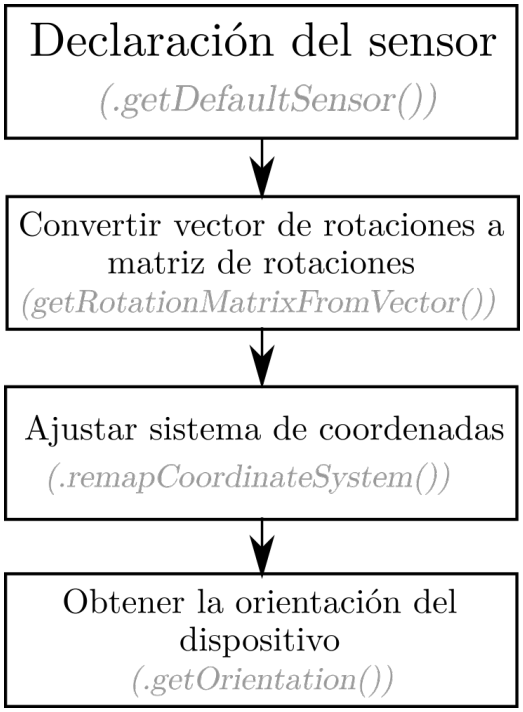


Figura 2.11: Diagrama a bloques para obtención de orientación mediante sensores basados en software.

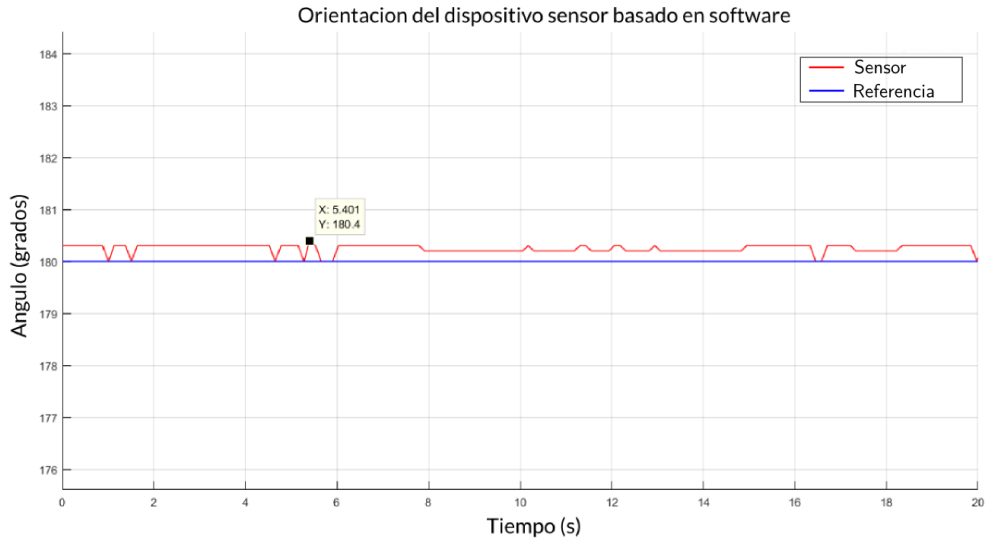


Figura 2.12: Gráfica de orientación obtenida mediante sensor de software de dispositivos móviles.

2.2.3.3. Orientación mediante sensores basados en hardware

Una manera de obtener la orientación del dispositivo es mediante el uso de sensores de hardware. Se puede obtener la rotación del dispositivo sobre el eje y combinando sensores

muy utilizados en aplicaciones de este tipo, como lo son el acelerómetro y el magnetómetro. Cuando se utilizan dos o más sensores para la estimación de un dato, es necesario hacer un procesamiento de las señales para obtener la estimación de la orientación.

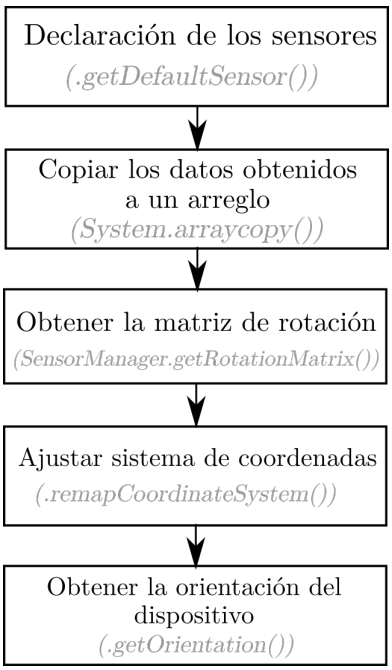


Figura 2.13: Diagrama a bloques para obtención de orientación mediante sensores basados en hardware.

La Figura 2.13 muestra el diagrama a bloques de las tareas que se deben realizar para obtener la orientación del dispositivo mediante sensores basados en hardware.

Los datos que proveen los sensores se deben copiar en diferentes arreglos para ser utilizados más adelante. Estos datos se utilizan para generar una matriz de rotación. Una vez que se calcula la matriz de rotación se hace un ajuste del sistema de coordenadas y posteriormente, se puede calcular la matriz de orientación que calcula el ángulo del dispositivo respecto a los ejes de rotación. Se utilizan los últimos valores de acelerómetro y magnetómetro para calcular una matriz de rotación sólo si ambos conjuntos de valores se han llenado con datos del sensor. El código siguiente muestra la implementación de dichos sensores para la obtención de la orientación del dispositivo:

```
1public final void onSensorChanged (SensorEvent event) {
2       if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
3          System.arraycopy(event.values, 0, mAccelerometerReading, 0,
               mAccelerometerReading.length);
4      }
5
6      if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD) {
7          System.arraycopy(event.values, 0, mMagnetometerReading, 0,
               mMagnetometerReading.length);
8      }
9      if (mAccelerometerReading!=null && mMagnetometerReading!=null) {
10         SensorManager.getRotationMatrix(mRotationMatrix, null,
               mAccelerometerReading, mMagnetometerReading);
11         //cambiamos el sistema de coordenadas
12         SensorManager.remapCoordinateSystem(mRotationMatrix, SensorManager.
               AXIS_X, SensorManager.AXIS_Z, mRotationMatrix);
13         //obtenemos la orientacion del dispositivo
14         SensorManager.getOrientation(mRotationMatrix, mOrientationAngles);
15         mOrientationAngles[0] = (float) Math.toDegrees(mOrientationAngles[0])
               ;
16         mOrientationAngles[1] = (float) Math.toDegrees(mOrientationAngles[1])
               ;
17         mOrientationAngles[2] = (float) Math.toDegrees(mOrientationAngles[2])
               ;
18         axisoX = mOrientationAngles[0];
19         axisoY = mOrientationAngles[1];
20         axisoZ = mOrientationAngles[2];
21     }
22}
```

El código anterior muestra cómo obtener la orientación del dispositivo mediante los sensores de hardware que provee Android®. La gráfica de la Figura 2.14 muestra la orientación medida por estos sensores. La línea azul representa la referencia para la orientación medida, la línea roja muestra la orientación medida por los sensores de hardware. Se observa que la orientación que se obtiene mediante los sensores de hardware es más ruidosa y llega a obtener valores con error de $\pm 1.5°$.

En el apéndice A se encuentra el código de la aplicación en Android ® en donde se utiliza el sensor de vector de rotación en Android®.

2.2.4. Envío de datos

Para obtener los datos del sensor cuando detecta un cambio se utiliza el método *onSensorChange*, el método se llama automáticamente cada vez que hay nuevos datos de sensores disponibles.

Una vez que se ha obtenido la orientación del dispositivo, se procesa el valor obtenido por el sensor. Se redondea a dos cifras decimales y se forma la cadena de datos que se va a enviar, como los caracteres especiales que indican un nuevo dato. Esta cadena se obtiene y envía mediante Bluetooth® por medio de una función que se describe en el código siguiente:

```
1public void showResults (float valor3){
2    float data;
3    data = valor3+180;
4    data = round(data,2);//Se redondea a dos decimales
```

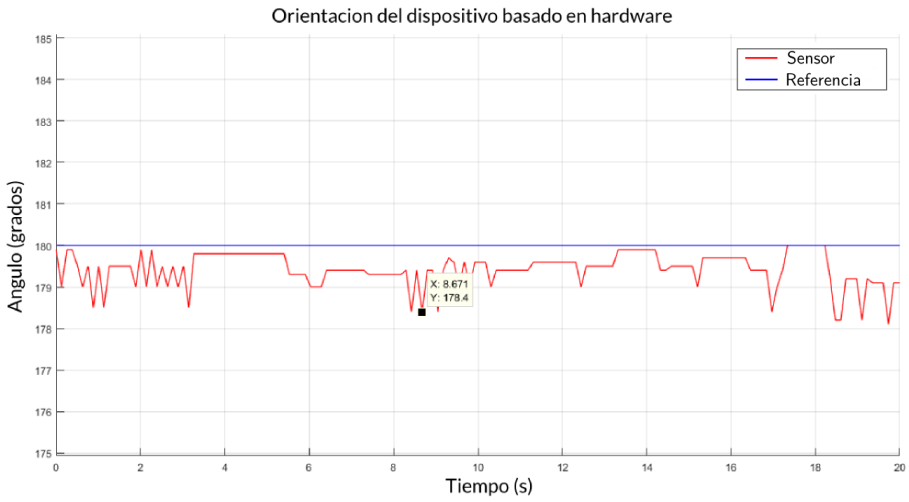


Figura 2.14: Gráfica de orientación obtenida mediante sensores de hardware de dispositivos móviles.

```
5     textViewX.setText(""+(data));//Se muestra el nuevo dato en pantalla
6     if (comunicar.isChecked()) { //Si el estado del boton de enclavamiento de
          enviar es verdadero
7         //se envia el dato obtenido
8         mConnectedThread.write("@");//Caracter especial que indica un nuevo
              dato
9         mConnectedThread.writeArduino(Float.toString(data));
10    }
11}
```

Para enviar datos a otro dispositivo se utiliza el hilo declarado y los datos se envían byte por byte. A continuación se describe la función *writeArduino* que se encarga de enviar los datos del dispositivo móvil a la tarjeta Arduino®. El envío de datos, como se explicó en 2.2.1, se realiza mediante comunicación serial. Los datos que se envían por medio de la conexión Bluetooth® deben ser separados por caracteres especiales que marquen el inicio y fin de cada dato con el objetivo de ser procesados en la tarjeta más adelante.

```
1//Funcion writeArduino que se encarga de enviar datos a otro dispositivo
2
3  private  OutputStream mmOutStream;
4  //Se recibe una cadena como parametro
5  public void writeArduino(String input) {
6        //Se declara un arreglo de bytes
7        byte[] msgBuffer;
8        //Convierte la cadena a bytes
9        msgBuffer = input.getBytes();
10       //Envia los datos
11       mmOutStream.write(msgBuffer);
12 }
```

2.3. Adquisición de datos en la tarjeta Arduino®

Arduino® es una plataforma de electrónica de código abierto cuyo software es capaz de ejecutarse sobre cualquier sistema operativo. Existen diferentes razones que hacen de Arduino® una plataforma viable para la realización de diferentes proyectos: económico, multiplataforma, con un entorno de programación sencillo y claro, de software de código abierto y extensible, es decir, el lenguaje puede expandirse a través de bibliotecas C++ o bien, al lenguaje de programación AVR C en el que se basa Arduino®. La tarjeta en la que se centra el presente documento es la placa Arduino® Nano, que es una tarjeta basada en un microcontrolador ATmega328P. (Ver Figura 2.15). El Arduino® Nano cuenta con las siguientes especificaciones técnicas:

- Microcontrolador: ATmega328.
- Arquitectura: AVR.
- Voltaje de funcionamiento: 5V.
- SRAM: 2KB.
- Velocidad de reloj: 16MHz.
- Pines I/O digitales: 22 (6 PWM).

La distribución de los pines puede verse en el ApéndiceE para más información sobre la plataforma Arduino®.

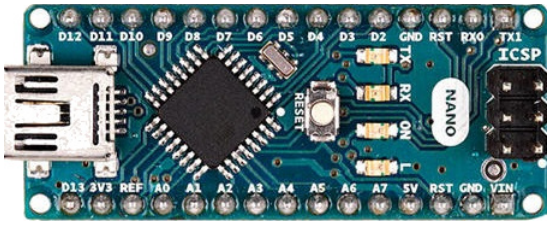


Figura 2.15: Controlador Arduino® Nano.

La plataforma Arduino® cuenta con diferentes módulos que pueden ser utilizados en conjunto con la tarjeta como son módulos de comunicación. La tarjeta cuenta con los pines TXD y RXD a los que se conecta el módulo de comunicación Bluetooth® con el que cuenta Arduino®.

2.3.1. Recepción de datos de la tarjeta Arduino®

La velocidad de transmisión y recepción se mide en baudios. Como la transmisión que se utiliza es binaria, un baudio equivale a un bit por segundo. Arduino es capaz de leer a diferentes frecuencias, sin embargo, en el caso de la conexión entre Arduino® y Android®, la comunicación se realiza a 115200 baudios.

Primeramente se hace la declaración de los pines que se utilizan para leer y transmitir. Posteriormente, se hace la declaración de la frecuencia a la que se lee la comunicación Bluetooth®, como se muestra en el siguiente fragmento de código:

```
SoftwareSerial BT(2,4);
void setup (){
   BT.begin(115200);
    Serial.begin(115200);
}
```

La forma en la que la tarjeta Arduino® recibe los datos enviados por el dispositivo móvil es leyendo caracter a caracter el dato recibido en el puerto. El dato de orientación es conformado por el número entero (tres dígitos) y la parte flotante (dos decimales). La recepción de los datos es caracter a caracter, por esto, los caracteres que integran el valor de la orientación forman parte de una cadena. El inicio de los datos que conforman dicha cadena los marca el caracter especial "@". La cadena se integra por solamente cinco caracteres. Una vez que se tienen todos los caracteres que conforman la cadena, se convierte a un tipo de dato flotante mediante la función *.toFloat()*. Lo anterior se muestra en el siguiente fragmento de código:

```
if(BT.available() > 0) {
  data = BT.read();
  if (data == '@'){
     for (i =0; i<5; i++){
      data = BT.read();
      if ((data >= 48 && data <= 57)||data==46){//no procesar basura
         sData+= (char)data;
      }
     }
 aux = sData.toFloat();
 if (aux >= 100){
  valorFinal = aux;
 }
}
```

En Arduino® también se programa el algoritmo de control de los motores (actuadores). La implementación del controlador PID se desarrolla y explica en el siguiente capítulo.

Capítulo 3

Control de equilibrio del sistema péndulo invertido móvil de dos llantas

El arreglo del carro - péndulo invertido es considerado uno de los problemas típicos en el área de control. Este sistema dinámico es multivariable, no lineal y mecánicamente inestable en lazo abierto. Sin embargo, con un sistema de control, se comporta de manera estable en un rango muy pequeño de movimiento.

En el presente capítulo se abordan los fundamentos teóricos requeridos para el diseño del algoritmo del control PID. Se plantea el modelo matemático del péndulo invertido móvil de dos llantas y posteriormente el diseño del sistema de control.

El sistema péndulo invertido móvil con llantas es un sistema con dinámica compleja, sin embargo presenta un comportamiento similar al del carro-péndulo. Las variables que definen la condición dinámica del sistema carro-péndulo invertido en todo momento, son la posición x del carro y el ángulo θ del péndulo con respecto a la vertical. El sistema carro - péndulo invertido se concibe como un cuerpo rígido cuyo movimiento se limita a dos dimensiones (plano x,y). En la Figura 3.1 se encuentra el diagrama de cuerpo libre del péndulo invertido móvil utilizado en el presente trabajo de tesis. Donde:

- m_p = Masa del péndulo.
- m_w = Masa de las llantas.
- g = Aceleración gravitacional.
- θ = Ángulo entre el péndulo y la vertical.
- L = Longitud total del péndulo.
- l = Longitud del pivote al centro de masa del péndulo.
- F = Fuerza externa.
- x = Desplazamiento en el eje x.

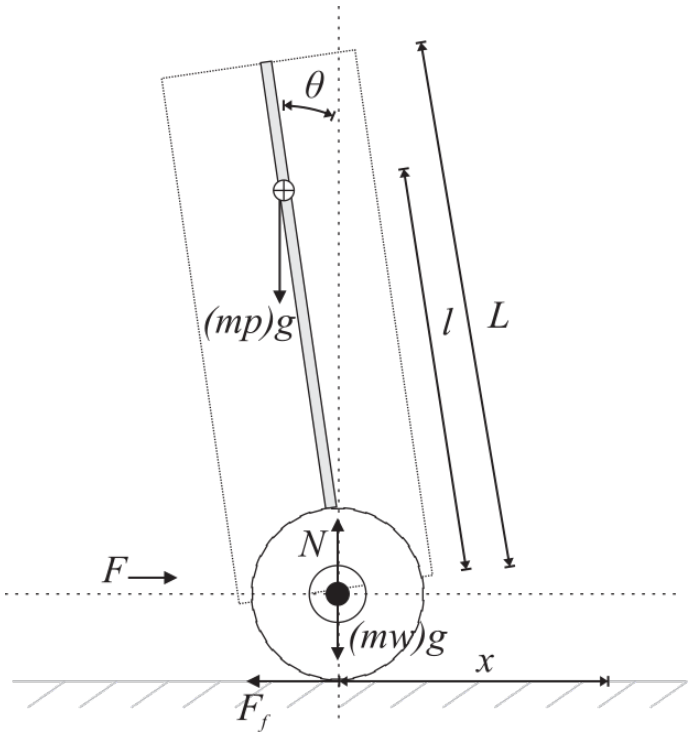


Figura 3.1: Diagrama de cuerpo libre del sistema péndulo invertido móvil.

3.1. Modelo dinámico del péndulo invertido con llantas

En la implementación de la plataforma experimental se utilizan como actuadores un par de motores de corriente directa. Para obtener el modelo que describe la dinámica de un motor de corriente directa de imánes permanentes el sistema se descompone en sus partes mecánica y eléctrica, como se muestra en la Figura 3.2.

3.1.1. Modelo dinámico de un motor de cd

La parte eléctrica del motor está compuesta por una resistencia interna del motor R_a, una inductancia de embobinado L_a, el voltaje de entrada V_a y una corriente electromotriz e_b que es generada en la armadura del motor. La ecuación que describe la dinámica eléctrica es definida como:

$$-V_a + R_a i_a + V_{L_a} + e_b = 0, \tag{3.1}$$

donde V_{L_a} es el voltaje en la bobina y e_b la fuerza contra electromotriz que pueden ser representadas por las ecuaciones:

$$V_{L_a} = L_a \frac{di_a}{dt}, \tag{3.2}$$

$$e_b = k_B \omega, \tag{3.3}$$

en donde la constante k_B representa la constante de velocidad correspondiente del motor y ω la velocidad angular de salida.

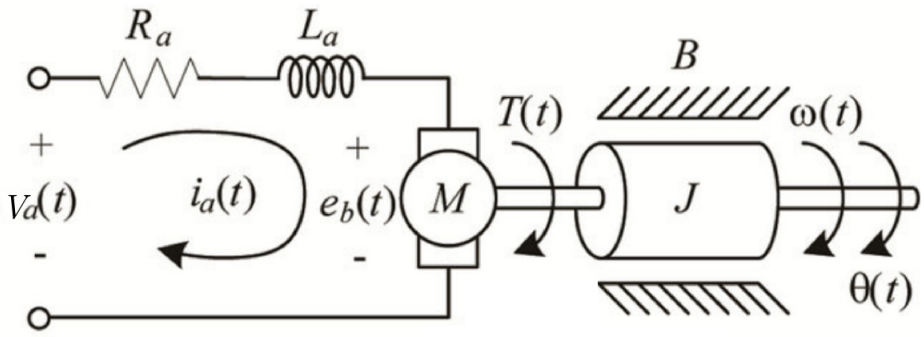


Figura 3.2: Diagrama simplificado de un motor de corriente directa.

Para obtener la ecuación que describe la dinámica del sistema, se sustituyen las ecuaciones (3.2) y (3.3) en (3.1) y se despeja $\frac{di_a}{dt}$:

$$\frac{di_a}{dt} = \frac{V_a}{L_t} - \frac{R_t}{L_t} i_T - \frac{k_B}{L_a}\omega. \tag{3.4}$$

La parte mecánica describe el movimiento de la flecha, las fuerzas y reacciones. Utilizando la ecuación dinámica de un cuerpo en rotación y considerando una inercia constante y fricción presente en el sistema se tiene:

$$\tau_m = J_m \frac{d\omega}{dt} + B_\omega, \tag{3.5}$$

donde J_m es la inercia rotacional del eje conectado al motor, τ_m el par del motor y B el coediciente de fricción viscosa del eje. La ecuación (3.6) relaciona el par con la corriente de la parte eléctrica del motor:

$$\tau_m = k_i i_T, \tag{3.6}$$

sustituyendo (3.6) en (3.5) y despejando $\frac{d\omega}{dt}$ se obtiene la ecuación:

$$\frac{d\omega}{dt} = \frac{k_i}{J_m} i_T - \frac{B}{J_m}\omega. \tag{3.7}$$

Las ecuaciones (3.4) y (3.7) aproximan de manera matemática la dinámica de un motor de corriente directa de imánes permanentes.

Así, las ecuaciones dinámicas de un motor de corriente directa con carga representadas en variables de estado:

$$\begin{bmatrix} \frac{di_a(t)}{dt} \\ \frac{d\omega_m(t)}{dt} \\ \frac{d\theta_m(t)}{dt} \end{bmatrix} = \begin{bmatrix} \frac{-Rt}{L_t} & \frac{-k_b}{L_t} & 0 \\ \frac{k_i}{J_m} & \frac{-B_m}{J_m} & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} i_a \\ \omega_m \\ \theta_m \end{bmatrix} V_a - \begin{bmatrix} 0 \\ \frac{1}{J_m} \\ 0 \end{bmatrix} T_L(t), \tag{3.8}$$

donde:

- i_a = corriente de armadura.
- ω_m = desplazamiento angular del motor.

- θ_m = desplazamiento angular del motor.
- V_a = voltaje suministrado al motor.
- R_t = resistencia de armadura.
- L_t = inductancia de armadura.
- J_m = inercia del motor.
- B_m = coeficiente de fricción viscosa.
- $k_b = k_i$ = constante de fuerza contraelectromotriz.
- T_L = carga del motor.

Este modelo es utilizado en el modelo dinámico del péndulo invertido móvil para obtener la relación entre el voltaje de entrada a los motores y el par necesario para mantener el equilibrio del robot sobre la vertical. Los actuadores del péndulo invertido móvil son dos motores de corriente directa Namiki [21]. La controlabilidad del modelo de los motores de corriente se presenta en la sección 3.2.

3.1.1.1. Obtención de los parámetros de un motor de corriente directa

A continuación se describen algunas técnicas utilizadas para la obtención de los parámetros de un motor de CD basados en [19].
Para establecer el valor de R_a del motor se mide con multímetro la resistencia entre las terminales del motor. El valor medido fue de $R_a = 6\Omega$.

El valor de la inductancia L_a es determinada mediante un instrumento especial para la medición de la misma. En el caso de la determinación de los valores de k_b, J_m y B_m deben medirse otros parámetros como corriente y voltaje suministrados al motor.

Para determinar el valor numérico de k_b se puede partir de la ecuación:

$$k_b = \frac{V_a - R_a i_a}{\omega}, \tag{3.9}$$

es necesario medir el valor de corriente y velocidad del motor a diferentes voltajes suministrados y definiendo k_b como un valor promedio de las mediciones tomadas.

Tensión de entrada(V)	Corriente consumida(A)	Frecuencia(rad/seg)	k_b
1.5	3.53e-2	1.56	0.8257
2.1	4.06e-2	2.25	0.8250
3.0	4.5e-2	3.4	0.8029
4.05	4.8e-2	4.63	0.8125
5.02	5.1e-2	5.88	0.8017
6.0	5.3e-2	7.1	0.8002
7.02	5.51e-2	8.4	0.7963
8.0	5.6e-2	9.71	0.7892

Tabla 3.1: Mediciones para determinar k_b.

El valor promedio de k_b se obtiene de los valores de la Tabla 3.1, en la que el valor promedio de la constante es $k_b = 0.8067V/(rad/seg)$.

Para la determinación de la inercia del motor se utiliza una prueba al escalón. El motor se aproxima a un sistema de primer orden y cuando se introduce una entrada escalón se presenta una constante de tiempo τ. Esta constante está definida en los motores eléctricos por la ecuación:

$$\tau = \frac{J_m R_a}{k_b k_i}, \tag{3.10}$$

cabe recordar que k_b y k_i son equivalentes numéricamente acorde a la teoría de máquinas eléctricas. Para calcular τ se debe alimentar el motor con una entrada escalón y encontrar el tiempo de subida al 63.2 % del valor final. Deben tomarse varias medidas para calcular un promedio de τ y obtener J_m a partir de la ecuación (3.10).

Medición	τ
1	3.43e-2
2	3.20e-2
3	3.31e-2
4	3.09e-2
5	3.46e-2
6	3.10e-2
7	3.51e-2
8	3.40e-2

Tabla 3.2: Mediciones del tiempo de subida.

La Tabla 3.2 muestra diferentes mediciones para calcular el promedio para τ. El tiempo de subida es $\tau = 33ms$. Por lo tanto,

$$J_m = \frac{\tau k_b^2}{R_a} = \frac{(33e-3)(0.8067)^2}{6} = 3.59e^{-}3kgm^2 \tag{3.11}$$

Para determinar el valor de la constante B_m se considera el estado estacionario en donde no existe aceleración angular y se obtiene B_m como:

$$B_m = \frac{k_i i_a}{\omega}. \tag{3.12}$$

De nueva cuenta, es conveniente tabular diferentes valores de las variables para obtener un valor promedio y determinar B_m.

Tensión de entrada(V)	Corriente consumida(A)	Frecuencia(rad/seg)	k_i	B_m
1.5	3.53e-2	1.56	0.8257	0.0186
2.1	4.06e-2	2.25	0.8250	0.0148
3.0	4.5e-2	3.4	0.8029	0.0106
4.05	4.8e-2	4.63	0.8125	0.0084
5.02	5.1e-2	5.88	0.8017	0.0069
6.0	5.3e-2	7.1	0.8002	0.0059
7.02	5.51e-2	8.4	0.7963	0.0052
8.0	5.6e-2	9.71	0.7892	0.0045

Tabla 3.3: Mediciones para determinar B_m.

El valor promedio obtenido para B_m fue $B_m = 0.0079 Nmseg/rad$.

Como resultado de la caracterización, se obtuvieron los parámetros del motor **DC NAMIKI 22CL-3501PG** [19]:

R_a	6Ω
L_a	104.3e-6 H
J_m	3.54e-3 kgm
B_m	7.5599e-3 $\frac{Nm}{rad/s}$
k_b, k_i	0.8028724 $\frac{Nm}{A}$

Tabla 3.4: Parámetros del motor Namiki 22CL.

3.1.2. Modelo dinámico del péndulo invertido con llantas

Para simplificar el análisis, el sistema se puede dividir como se indica en el diagrama de cuerpo libre de la Figura 3.3 en dos cuerpos: el péndulo y el carro (actuadores).

En la Figura 3.3 a) se observa el esquema de análisis de una sola llanta, se incluyen ambas llantas en el análisis del péndulo. En la Figura 3.3 b) el péndulo incluye en su análisis el torque y reacciones generadas por las dos llantas que incluye la plataforma.

El modelo del péndulo invertido con llantas tiene una dinámica compleja similar a un sistema carro-péndulo. La dinámica del péndulo y las llantas se analizan por separado, el modelo que se presenta a continuación se basa en [12]. El comportamiento del robot se ve influenciado por el par que las llantas reciben de los motores. En la Figura 3.3(a) se muestra el diagrama de cuerpo libre correspondiente a las llantas del sistema. Donde:

- θ_w = Ángulo desplazado de la llanta.
- C_R = Par aplicado en la llanta.
- m_w = Masa de la llanta.
- H_R = Fuerza sobre la horizontal.
- P_R = Fuerza sobre la vertical.

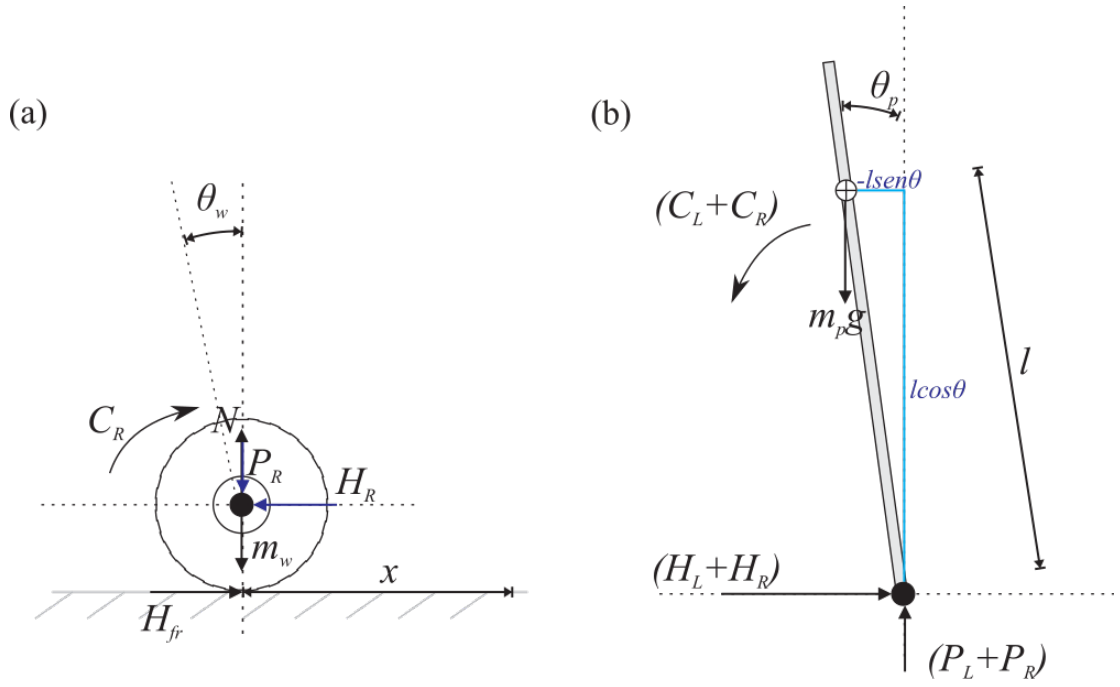


Figura 3.3: Diagrama de cuerpo libre del (a)carro y (b)péndulo invertido.

Utilizando la segunda ley de Newton, la sumatoria de fuerzas horizontales sobre el eje x se expresan como:

$$\sum F_x = Ma, \tag{3.13}$$

$$m_w\ddot{x} = H_{fr} - H_R. \tag{3.14}$$

Mientras que la suma de fuerzas alrededor de las llantas es definido como:

$$\sum M_o = I\alpha, \tag{3.15}$$

$$I_r\ddot{\theta} = C_R - H_{fr}r, \tag{3.16}$$

donde C_R es el par en la llanta recibido de los motores. De la dinámica del motor de corriente directa, el par del motor puede ser expresado por:

$$C_R = \frac{-k_m k_e}{R}\dot{\theta_w} + \frac{km}{R}. \tag{3.17}$$

Sustituyendo 3.17 en 3.16:

$$I_w\ddot{\theta_w} = \frac{-k_m k_e}{R}\dot{\theta_w} + \frac{km}{R}V - H_{fr}r_w, \tag{3.18}$$

$$H_{fr} = \frac{-k_m k_e}{Rr_w}\dot{\theta_w} + \frac{km}{Rr_w}V - \frac{I_w}{r_w}\ddot{\theta_w} - H_L. \tag{3.19}$$

Sustituyendo 3.19 en la ecuación 3.14 se obtiene el modelo de las llantas expresado como:

$$m_w\ddot{x} = \frac{-k_m k_e}{Rr_w}\dot{\theta_w} + \frac{km}{Rr_w}V - \frac{I_w}{r_w}\ddot{\theta_w} - H_L. \tag{3.20}$$

La rotación angular puede transformarse en movimiento lineal considerando:

$$\ddot{\theta_w} = \frac{\ddot{x}}{r_w},$$
$$\dot{\theta_w} = \frac{\dot{x}}{r_w}.$$

Considerando que el robot péndulo invertido tiene dos llantas, se hace la suma de ambas para el modelo final:

$$2(m_w + \frac{I_w}{r_w^2})\ddot{x} = \frac{-2k_m k_e}{Rr_w^2}\dot{x} + \frac{2k_m}{Rr_w}V - (H_L + H_R). \tag{3.21}$$

En la Figura 3.3(b) se muestra el diagrama de cuerpo libre correspondiente al péndulo. Se observa que además del peso, interactúan las fuerzas de reacción H y P sobre la articulación con el carro. La aceleración horizontal del péndulo se expresa como:

$$\sum F_x = m_p\ddot{x}, \tag{3.22}$$
$$(H_L + H_R) - m_p l\ddot{\theta_p}\cos\theta_p + m_p l\dot{\theta_p}^2\sin\theta_p = m_p\ddot{x}, \tag{3.23}$$

y su componente de fuerza de reacción vertical como:

$$\sum F_{xp} = m_p\ddot{x}\cos\theta_p, \tag{3.24}$$
$$(H_L + H_R)\cos\theta_p + (P_L + P_R)\sin\theta_p - m_p g\sin\theta_p - m_p l\ddot{\theta} = m_p\ddot{x}\theta_p. \tag{3.25}$$

La suma de momentos al rededor del centro de masa del péndulo se definen como:

$$\sum M_o = I\alpha, \tag{3.26}$$
$$-(H_L + H_R)l\cos\theta_p - (P_L + P_R)l\sin\theta_p - (C_L + C_R) = I\ddot{\theta}, \tag{3.27}$$

donde el torque aplicado al péndulo es proporcionado por los motores:

$$C_L + C_R = \frac{-2k_m k_e\dot{x}}{Rr_w} + \frac{2k_m V}{R}, \tag{3.28}$$

en donde r_w es el radio de las llantas y el resto, son parámetros conocidos del motor. Sustituyendo la ecuación anterior en 3.27 se tiene:

$$-(H_L + H_R)l\cos\theta_p - (P_L + P_R)l\sin\theta_p = I_p\ddot{\theta_p} - \frac{2k_m k_e\dot{x}}{Rr_w} + \frac{2k_m V}{R}. \tag{3.29}$$

Las ecuaciones no lineales que describen la dinámica del sistema:

$$(I_p + m_p l^2)\ddot{\theta_p} - \frac{2k_m k_e\dot{x}}{Rr_w} + \frac{2k_m V}{R} + m_p gl\sin\theta_p = -m_p l\ddot{x}\cos\theta_p, \tag{3.30}$$

$$\frac{2k_mV}{Rr_w} = \left(2m_w + \frac{2I_w}{r_w^2} + m_p\right)\ddot{x} + \frac{2k_mk_e\dot{x}}{Rr_w^2} + m_pl\ddot{\theta_p}\cos\theta_p - m_pl\dot{\theta_p}^2\sin\theta_p. \tag{3.31}$$

Considerando que $\theta_p = \pi + \phi$, donde ϕ es una pequeña vecindad alrededor de la vertical y π es considerado el punto de partida del péndulo, es decir, en posición vertical hacia arriba,

$$\cos\theta_p = \cos(\pi + \phi) = -1, \tag{3.32}$$

$$\sin\theta_p = \sin(\pi + \phi) = -\phi, \tag{3.33}$$

$$\dot{\theta_p}^2 = \dot{\phi}^2 = 0. \tag{3.34}$$

Bajo esta consideración, las ecuaciones (3.30) y (3.31) se reescriben como:

$$\ddot{x_e} = \frac{2k_mV}{Rr_w\left(2m_w + \frac{2I_w}{r_w^2} + m_p\right)} - \frac{2k_mk_e\dot{x_e}}{Rr_w^2\left(2m_w + \frac{2I_w}{r_w^2} + m_p\right)} + \frac{m_pl\ddot{\phi_e}}{\left(2m_w + \frac{2I_w}{r_w^2} + m_p\right)}, \tag{3.35}$$

$$\ddot{\phi_e} = \frac{m_pl\ddot{x_e}}{(I_p + m_pl^2)} + \frac{2k_mk_e\dot{x_e}}{Rr_w(I_p + m_pl^2)} - \frac{2k_mV}{R(I_p + m_pl^2)} + \frac{m_pgl\phi_e}{(I_p + m_pl^2)}. \tag{3.36}$$

3.2. Principios de diseño del sistema de control

En la sección 3.1.2 se obtiene el modelo dinámico del péndulo invertido móvil con llantas. De las ecuaciones 3.35 y 3.36, que representan el modelo linealizado alrededor de una vecindad cercana a los 180°, la representación en el espacio de estados, se define como:

$$\begin{bmatrix}\dot{x_e}\\ \ddot{x_e}\\ \dot{\phi_e}\\ \ddot{\phi_e}\end{bmatrix} = \begin{bmatrix}0 & 1 & 0 & 0\\ 0 & \frac{2k_mk_e(m_plr_w - I_p - m_pl^2)}{Rr_w^2\alpha} & \frac{m_p^2gl^2}{\alpha} & 0\\ 0 & 0 & 0 & 1\\ 0 & \frac{2k_mk_e(r_w\beta - m_pl)}{Rr_w^2\alpha} & \frac{m_pgl\beta}{\alpha} & 0\end{bmatrix}\begin{bmatrix}x_e\\ \dot{x_e}\\ \phi_e\\ \dot{\phi_e}\end{bmatrix} + \begin{bmatrix}0\\ \frac{2k_m(I_p + m_pl^2 - m_plr_w)}{Rr_w\alpha}\\ 0\\ \frac{2k_m(m_pl - r_w\beta)}{Rr_w\alpha}\end{bmatrix}V_a, \tag{3.37}$$

en donde:

$$\beta = (2m_w + \frac{2I_w}{r^2} + m_p),$$
$$\alpha = I_p\beta + 2m_pl^2(m_w + \frac{I_w}{r_w^2}).$$

Los parámetros de la plataforma experimental utilizados en las simulaciomes de Matlab se presentan en la Tabla 4.2 en la sección 4.1.4.

De las ecuaciones en el espacio de estados, se definen las matrices **(A, B, C, D)** como:

$$A = \begin{bmatrix}0 & 1 & 0 & 0\\ 0 & \frac{2k_mk_e(m_plr_w - I_p - m_pl^2)}{Rr_w^2\alpha} & \frac{m_p^2gl^2}{\alpha} & 0\\ 0 & 0 & 0 & 1\\ 0 & \frac{2k_mk_e(r_w\beta - m_pl)}{Rr_w^2\alpha} & \frac{m_pgl\beta}{\alpha} & 0\end{bmatrix},$$

$$B = \begin{bmatrix} 0 \\ \frac{2k_m(I_p+m_pl^2-m_plr_w)}{Rr_w\alpha} \\ 0 \\ \frac{2k_m(m_pl-r_w\beta)}{Rr_w\alpha} \end{bmatrix},$$

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$D = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

La función de transferencia del sistema, considerando Φ como salida y V como entrada se obtiene mediante el comando de MATLAB *ss2tf()* que convierte la representación del espacio de estados a una función de transferencia, tomando como parámetros las matricezs **A, B, C** y **D**. La función de transferencia, obtenida mediante el comando de MATLAB es:

$$\frac{\Phi(s)}{V_a(s)} = \frac{-3.231s^2}{s^4 + 9.803s^3 - 5.709s^2 - 57.87s}. \tag{3.38}$$

Controlabilidad completa del estado de sistemas en tiempo continuo

Sea el sistema en tiempo continuo:

$$\dot{x} = Ax + Bu, \tag{3.39}$$

donde x = vector de estados (de dimensión n), u = señal de control (escalar), A = matriz de n x n, B = matriz de n x 1.

Si el sistema es de estado completamente controlable, entonces el rango de la matriz n x n, es decir:

$$\begin{bmatrix} B & \vdots & AB & \vdots & ... & \vdots & A^{n-1}B \end{bmatrix}, \tag{3.40}$$

es de rango n[22].

Para el modelo del motor de cd, la matriz de controlabilidad:

$$C = \begin{bmatrix} B & AB & A^2B \end{bmatrix} = \begin{bmatrix} 0 & -2.1745e6 & 1.2510e11 \\ 282.4859 & -0.0006e6 & -0.0049e11 \\ 0 & 0.0003e6 & 0 \end{bmatrix},$$

en donde el rango de la matriz de controlabilidad se obtuvo mediante el comando de MATLAB *rank()*. El rango de la matriz de controlabilidad es de rango 3.

Para el modelo del péndulo invertido 3.37, la matriz de controlabilidad:

$$C = \begin{bmatrix} B & AB & A^2B & A^3B \end{bmatrix} = \begin{bmatrix} 0 & 0.8079 & -7.7629 & 0.0743e3 \\ 0.8079 & -7.7629 & 74.3454 & -0.7120e3 \\ 0 & -2.0449 & 19.6494 & -0.2045e3 \\ -2.0449 & 19.6494 & -204.4605 & 1.9586e3 \end{bmatrix},$$

en donde el rango de la matriz de controlabilidad es de rango 4.

La respuesta en el tiempo de un sistema de control consta de dos partes: la respuesta transitoria y la respuesta en estado estacionario. La respuesta transitoria se refiere a la que va del estado inicial al estado final. Por respuesta en estado estacionario se entiende la manera como se comporta la salida del sistema conforme t tiende a infinito [22].

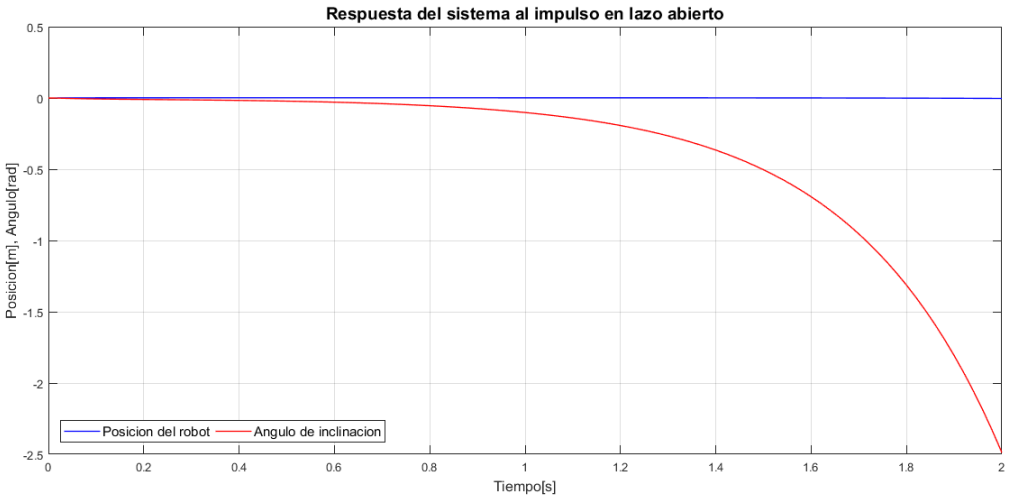


Figura 3.4: Respuesta ante el impulso de la plataforma péndulo invertido móvil en lazo abierto.

La Figura 3.4 muestra la respuesta al impulso del sistema péndulo invertido móvil en lazo abierto.

La Figura 3.4 muestra como el ángulo medido de la vertical tiende a incrementar conforme pasa el tiempo. En la gráfica se observa que luego de 1.2 segundos el ángulo de la plataforma fuera de la vertical es de 25° ($0.44rad$) y el desplazamiento de la plataforma es de apenas $3mm$. En la gráfica de la respuesta de la plataforma ante una señal impulso, el cuerpo del péndulo cae al piso antes de los dos segundos.

La respuesta que plataforma presenta a lazo abierto ante una señal impulso ejemplifica la inestabilidad del sistema.

La característica básica de la respuesta transitoria de un sistema en lazo cerrado se relaciona estrechamente con la localización de los polos en lazo cerrado. En una función racional, los *ceros* son las raíces del polinomio del numerador y los *polos* son las raíces del polinomio denominador. En el caso del sistema péndulo invertido los polos del sistema son:

0
-9.7817
2.4218
-2.4430

Tabla 3.5: Polos a lazo abierto de la función de transferencia del sistema.

El comando de MATLAB que se usa con frecuencia para dibujar el lugar geométrico de las raíces es *rlocus(num, den)*. La Figura 3.5 muestra el lugar geométrico de las raíces del sistema péndulo invertido móvil conforme la ganancia K, de un controlador proporcional tiende a infinito.

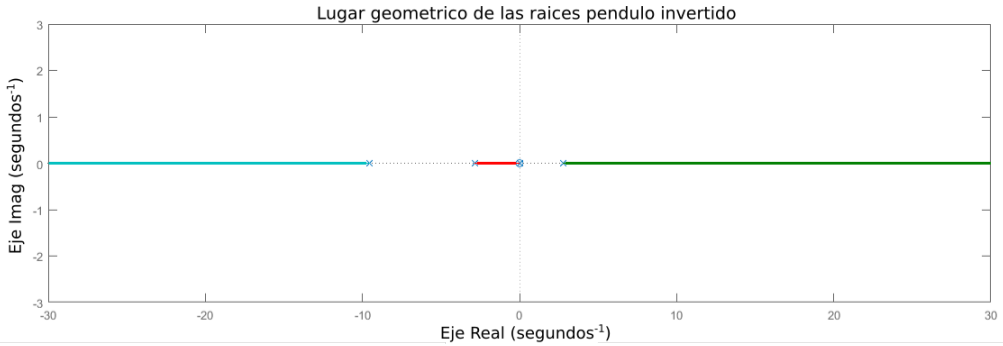


Figura 3.5: Ubicación de polos y ceros del sistema con controlador proporcional.

3.3. Controlador PID

Un controlador PID toma la referencia del sistema y la compara con el valor real de la señal de interés. La diferencia entre estas señales se conoce como el error del sistema. Si el error es cero, el controlador PID no actúa, sin embargo, si hay una disparidad entre las señales es necesaria una acción correctiva.

La Figura 3.6 muestra el diagrama a bloques de un sistema de control PID aplicado a una planta, en donde *referencia* es el valor deseado de la plataforma, *perturbación* es una fuerza aplicada a la plataforma, *salida* es la señal de interés de la plataforma, e la señal de error del sistema (*referencia* - *salida*) y u la entrada a la función de transferencia de la plataforma.

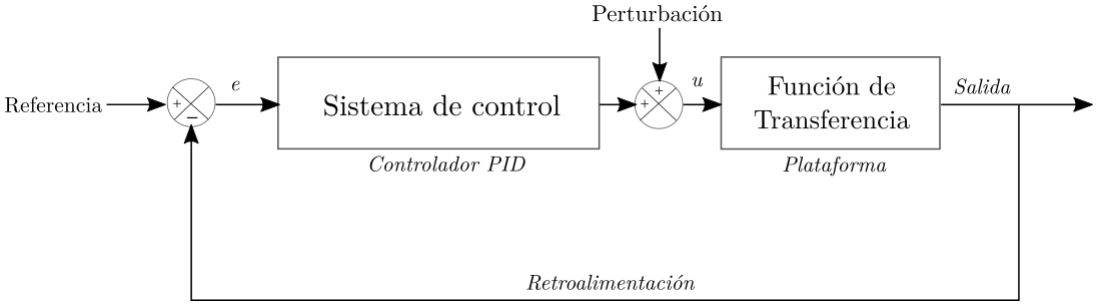


Figura 3.6: Diagrama a bloques de control PID de una plataforma.

Los controladores PID son un tipo de controlador que combinan la acción proporcional **P**, la acción integral **I** y la acción derivativa **D** sobre el error. Si a la entrada del controlador (error) se llama *e(t)* y a la salida del controlador (entrada a la planta) se le llama *u(t)*, se puede definir el PID de la siguiente forma:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}. \tag{3.41}$$

La acción de control proporcional **P** da una salida del controlador proporcional a la señal de error medida. Un controlador proporcional posee un error en régimen permanente. La acción integral **I** de un controlador es proporcional al error acumulado. La función de

transferencia de un controlador PID se define como:

$$C_{PID}(s) = K_p(1 + \frac{1}{T_i s} + T_d s). \tag{3.42}$$

El entorno de MATLAB proporciona herramientas de diseño (*PID Tuner*) para controladores PI, PD y PID que proporcionan de manera sencilla las ganancias K_p, K_i y K_d del controlador teniendo un sistema linealizado y proporcionando parámetros de velocidad de respuesta y robustez deseados. La Figura 3.7 muestra el diagrama a bloques de la implementación de un bloque de control PID con la plataforma experimental cuya función de transferencia proporciona el sistema linealizado para la herramienta *PID Tuner*.

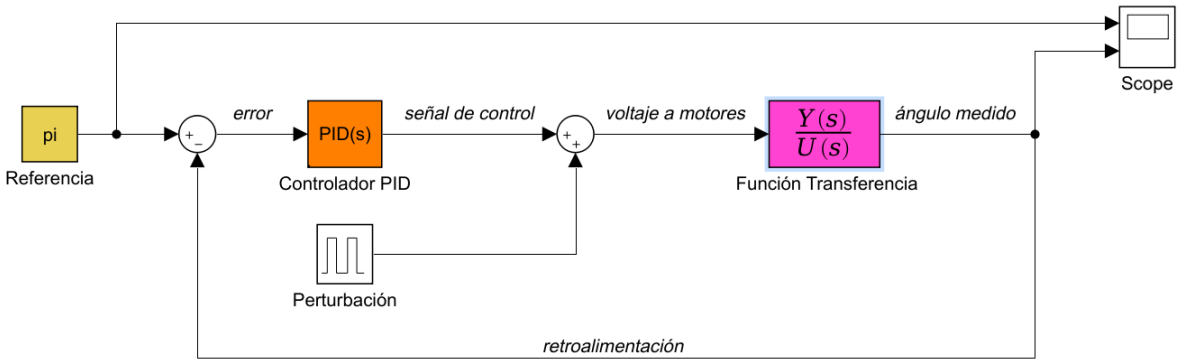


Figura 3.7: Diagrama a bloques de la respuesta de la plataforma con controlador PID a lazo cerrado.

La Figura 3.8 muestra la respuesta del sistema considerando el controlador PID en lazo cerrado.

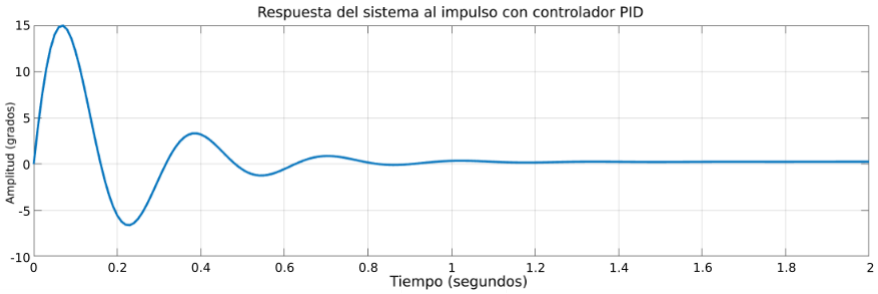


Figura 3.8: Respuesta de la plataforma con controlador PID a lazo cerrado.

La Figura 3.9 muestra la ubicación de los polos y ceros del sistema con un controlador PID.

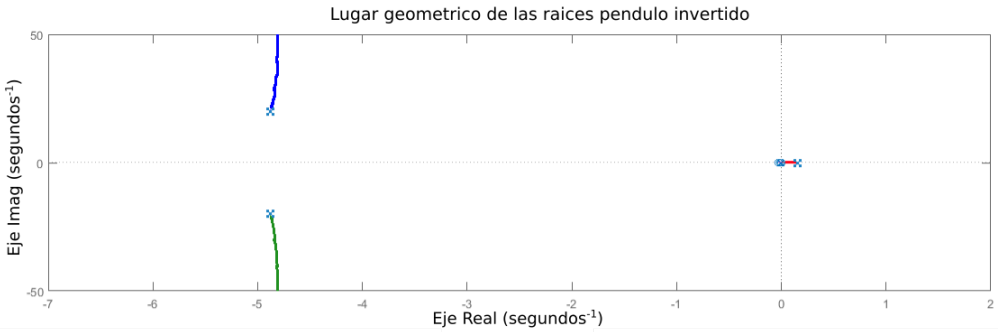


Figura 3.9: Ubicación de polos y ceros del sistema con controlador PID ante una señal impulso.

3.3.1. Implementación del controlador PID en la tarjeta Arduino®

Del diagrama de flujo de un controlador PID se determina lo siguiente:

El bloque de control *proporcional* consiste en el producto entre la señal de error y la constante proporcional.

El bloque de control *integral* actúa cuando hay una desviación entre la variable del sensor y el punto de referencia, integrando esta desviación en el tiempo y sumándola a la acción proporcional.

El bloque de control *derivativo* considera la tendencia del error y permite una repercusión rápida de la variable después de presentarse una perturbación en el proceso.

La implementación del algoritmo de control PID para la tarjeta Arduino tomó como referencia [18], el código de la implementación, se muestra a continuación:

```
1 // Variables utilizadas en el controlador PID.
2 unsigned long lastTime;
3 double Input, Output, Setpoint;
4 double errSum, lastErr;
5 double kp, ki, kd;
6 void calcularPID()
7 {
8       // Tiempo transcurrido desde el ultimo calculo.
9       unsigned long now = millis();
10      double timeChange = (double)(now - lastTime);
11      // Variables de error.
12      double error = Setpoint - Input;
13      errSum += (error * timeChange);
14      double dErr = (error - lastErr) / timeChange;
15      // Calculamos la funcion de salida del PID.
16   Output = kp * error + ki * errSum + kd * dErr;
17      // Guardamos el valor de algunas variables para el proximo ciclo de
            calculo.
18   lastErr = error;
19   lastTime = now;
20 }
```

Del código anterior, siendo el $error = referencia - entrada$, si se considera que no hay cambios en el *setpoint* o referencia entonces:

$$\frac{dError}{dt} = -\frac{dInput}{dt}.$$

```
1 // Variables utilizadas en el controlador PID.
2 unsigned long lastTime;
3 double Input, Output, Setpoint;
4 double errSum, lastErr;
5 double kp, ki, kd;
6 void calcularPID()
7       {
8       /* Tiempo transcurrido desde el ultimo calculo. */
9       unsigned long now = millis();
10      double timeChange = (double)(now - lastTime);
11      /* Variables de error. */
12      double error = Setpoint - Input;
13      errSum = ((lastErr+error) * timeChange);
```

```
14        double dInput = (Input - lastINput) / timeChange;
15        // Calculamos la funcion de salida del PID.
16    Output = kp * error + ki * errSum - kd * dInput;
17        // Guardamos el valor de algunas variables para el proximo calculo.
18    lastErr = error;
19    lastTime = now;
20 }
```

Del código anterior, las ganancias Kp, Ki y Kd son las ganancias obtenidas del análisis de la plataforma con la herramienta de MATLAB, ver apéndice D.

Arduino cuenta con 6 salidas digitales tipo PWM. Para cada controlador se requieren de tres pines: dos de ellos son los que definen el sentido del motor y el tercero define al PWM. Las conecciones entre los componentes electricos se observan en el diseño de la placa PCB en el Apéndice F.

Capítulo 4

Estructura mecánica e implementación de la plataforma experimental

4.1. La plataforma experimental

En los capítulos anteriores se describió el desarrollo de la aplicación para sensar la orientación del dispositivo y la implementación de un controlador PID para la plataforma experimental en la tarjeta Arduino®. A continuación se describe la implementación de la plataforma experimental.

El robot tipo péndulo invertido móvil con llantas integra el desarrollo del dispositivo móvil como sensor de orientación, la implementación del controlador PID en la tarjeta Arduino® y el desarrollo estructural de la plataforma.

4.1.1. Modelo de diseño para la plataforma experimental

El péndulo invertido móvil que se describe a lo largo de este documento tiene una estructura similar al sistema *carro-péndulo*, Figura 4.1, en la que el *carro* se sustituye por los actuadores del péndulo invertido móvil (motores y llantas) y el *péndulo* por un bloque rectangular, para simplificar la caracterización de la plantaforma, en la que se encuentran el dispositivo móvil y los componentes electrónicos.

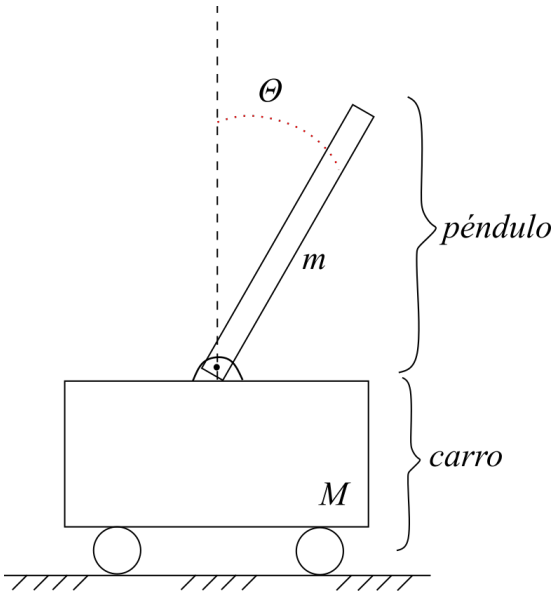


Figura 4.1: Modelo del sistema carro-péndulo.

La estructura mecánica de la plataforma experimental basa su diseño de construcción en la Figura 4.2, en donde el cuerpo del péndulo se construye como un bloque rectangular. El dispositivo móvil, encargado de sensar la orientación, envía los datos obtenidos mediante comunicación Bluetooth® con la tarjeta Arduino®. En Arduino® se implementa un algoritmo de control PID para el control de auto balanceo de la estructura mecánica.

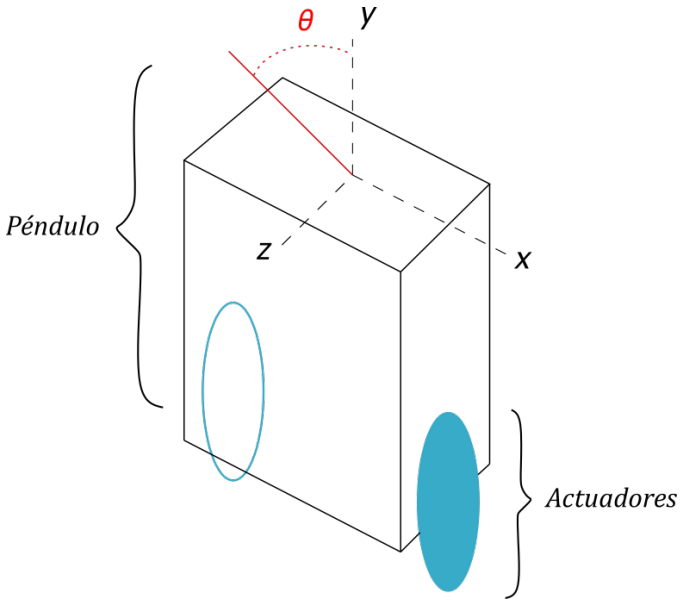


Figura 4.2: Modelo péndulo invertido móvil.

4.1.2. Placa electrónica de la plataforma experimental

En el capítulo 2 se mostró el desarrollo de la aplicación para el dispositivo móvil y la recepción de datos en la tarjeta Arduino®. Sin embargo, la placa electrónica que actúa como el *cerebro* de la plataforma experimental que consta de la conexión entre: la tarjeta Arduino®, los impulsores eléctricos (*motor driver*), los motores, el módulo de comunicación Bluetooth® y las baterías que alimentan a la placa Arduino® y a los controladores, se muestra a continuación. La Tabla 4.1 lista los componentes electrónicos de la plataforma experimental.

Tarjeta Arduino®	Arduino® Nano ATMega 328p
Motor driver	L298N
Motores de corriente directa	Motores Namiki con caja de reducción 80:1
Módulo de comunicación	Módulo HC-06
Baterías	LiPo 12V y recargable 9V

Tabla 4.1: Componentes electrónicos de la plataforma experimental.

El diagrama a bloques de la Figura 4.3 muestra la relación entre los componentes electrónicos de la plataforma.

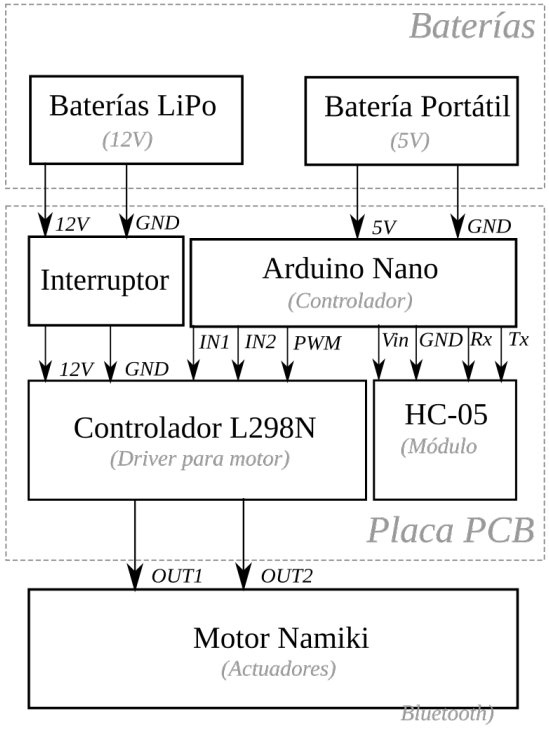


Figura 4.3: Diagrama a bloques de relación entre componentes electrónicos.

Las baterías que alimentan el sistema son: una batería de LiPo de 12V conectada a un interruptor que enciende y apaga los controladores del motor y una batería portátil que se encarga solo de alimentar la tarjeta Arduino® y el módulo Bluetooth®.

El controlador de los motores se conecta a las salidas digitales de Arduino® que funcionan como PWM, a los motores y a la alimentación de la placa de 12V. Arduino cuenta con 6 salidas digitales tipo PWM (Ver E), los controladores L298N cuentan con los pines de alimentación *12V* y *GND*, los pines de entrada *IN1*, *IN2* y *ENA* que proporciona la tarjeta Arduino® y los pines de salida *OUT1* Y *OUT2* que son la salida a los motores.

La placa PCB de conexiones entre los sistemas electrónicos descritos anteriormente, se encuentra en el Apéndice F.

4.1.3. Estructura mecánica

Como se mencionó en los capítulos anteriores, para la implementación se utiliza un dispositivo móvil como sensor que se comunica mediante protocolo Bluetooth®. Entre el sensor y el controlador no existe cableado, lo que hace posible montar y desmontar el teléfono de manera sencilla. La orientación del dispositivo móvil no se ve afectada por la orientación definida por la pantalla. Es por eso que es posible adaptar el dispositivo móvil a la plataforma y que no afecte a la estructura mecánica. Para la implementación del dispositivo móvil a la plataforma experimental, la posición cuya ubicación no afectaba tanto a la dinámica del sistema es con la pantalla hacia arriba (en posición vertical como se analiza más adelante en los resultados). La Figura 4.4 muestra cómo se obtiene la medición del ángulo de inclinación mediante el uso de la aplicación desarrollada.

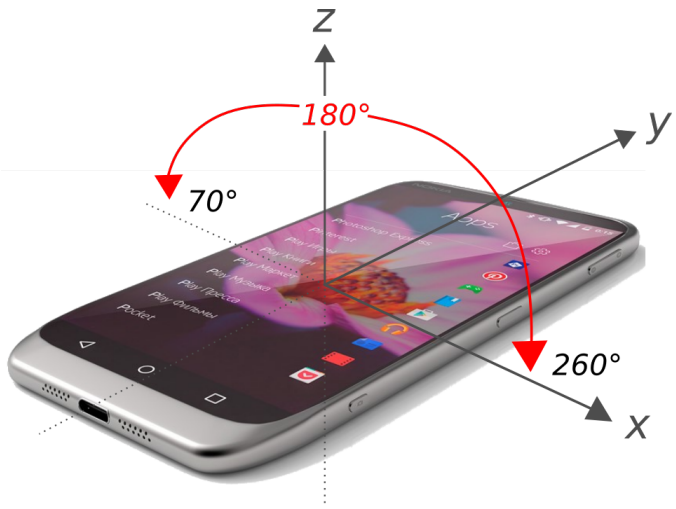


Figura 4.4: Rango de operación de la aplicación en el dispositivo móvil.

La plataforma experimental basa su estructura en el péndulo invertido móvil con llantas (como se muestra en la sección 4.1.1). En el caso específico de la plataforma, los actuadores conforman el carro. Los actuadores de la plataforma experimental son un par de motores de corriente directa con una caja de engranes 80:1, esto para obtener una buena relación torque-rapidez y que se acoplan a un par de llantas de diametro de 13.4cm (par de llantas

negras *monster truck* disponibles en *https://teslabem.com/tienda*). Ver Figura 4.5.

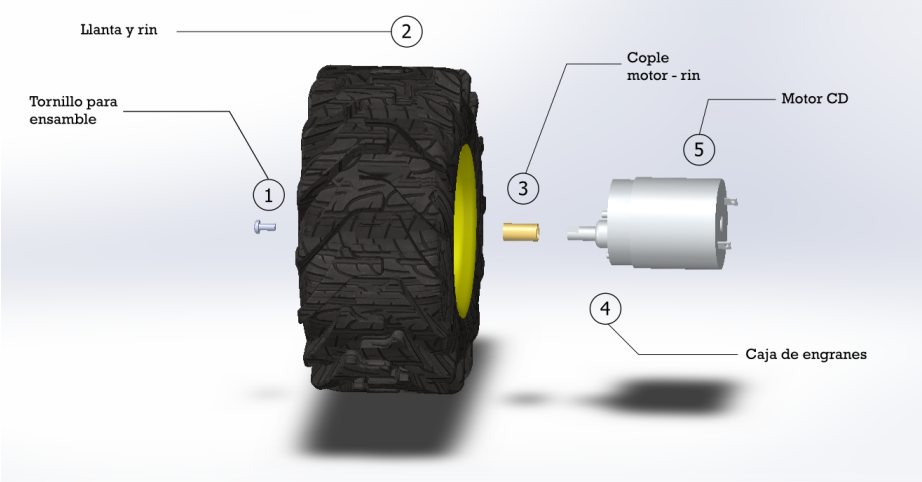


Figura 4.5: Elementos que conforman los actuadores de la plataforma experimental.

El péndulo, por su parte, está formado por tres niveles (ver figura 4.7): el primero en el que se localizan las baterías que alimentan la plataforma, el segundo que es donde se encuentra la placa electrónica formada por la tarjeta Arduino®, el módulo Bluetooth® y los controladores de los motores, ver Figura 4.6.

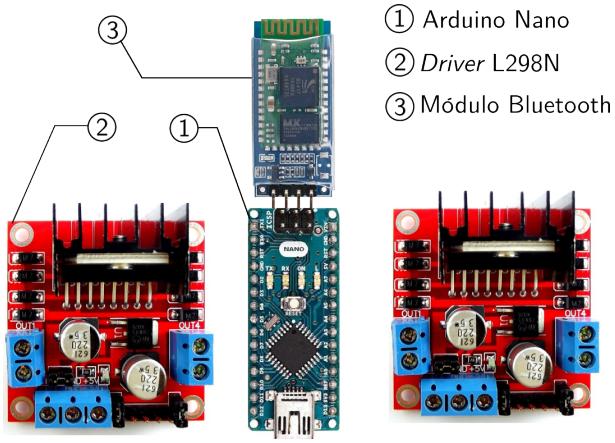


Figura 4.6: Elementos electrónicos que conforman el segundo nivel del péndulo.

El tercer nivel o nivel superior del cuerpo del péndulo es donde se localiza el dispositivo móvil como alternativa a los sensores inerciales (ver figura 4.7). La longitud total del péndulo es de 17.0cm.

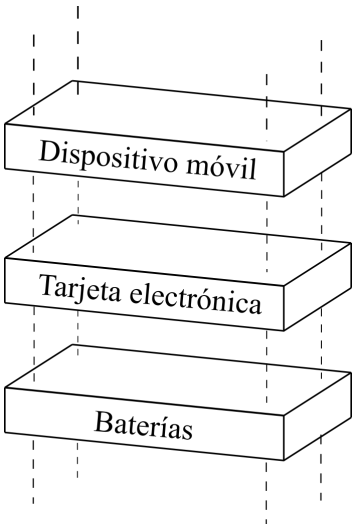


Figura 4.7: Módulos que conforman la estructura del péndulo en la plataforma experimental.

Las Figuras 4.8 y 4.9 muestran la estructura mecánica realizada para la plataforma experimental tipo péndulo invertido móvil.

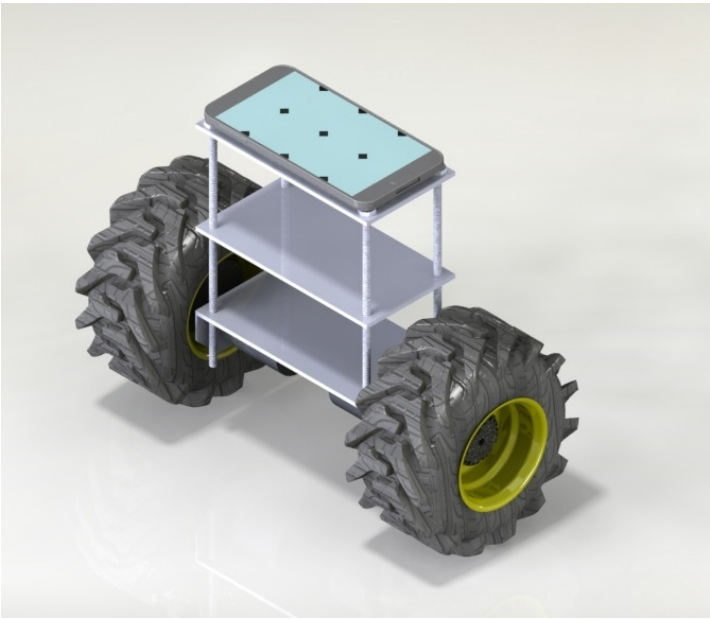


Figura 4.8: Estructura mecánica para la plataforma experimental final.

La Figura 4.10 muestra la plataforma experimental final con la integración de todos sus elementos.

Figura 4.9: Estructura mecánica para la plataforma experimental final (vista frontal).

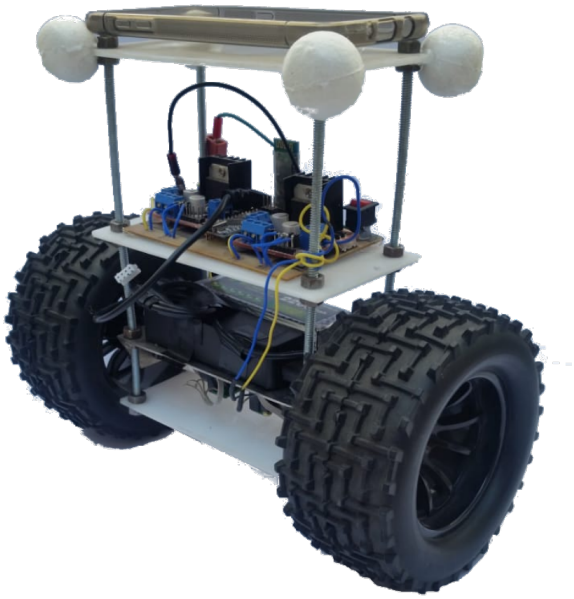


Figura 4.10: Estructura mecánica de la plataforma final.

4.1.4. Integración de la plataforma experimental

La plataforma final integra el software descrito en los capítulos 2 y 3 con el hardware de la plataforma experimental. El diseño estructural de la plataforma define el comportamiento físico e influye en la inestabilidad que puede llegar a presentar la plataforma en las pruebas experimentales. El funcionamiento completo de todos los elementos implementados como conjunto se observa en diagrama a bloques de la figura 4.11.

Parámetros de la plataforma experimental

La obtención de los parámetros de la plataforma experimental se calcularon en base a [23] cuyos cálculos y resultados se presentan a continuación:

Inercia de las llantas El momento de inercia de las llantas se calcula como el momento de un disco:

$$I_w = \frac{M_w(r_w)^2}{2}, \tag{4.1}$$

donde:

- M_w = Masa de las llantas.
- r_w = Radio de las llantas.

Sustituyendo los valores de la masa de las llantas y el radio, se obtiene:

$$I_w = \frac{0.620(0.065)^2}{2} = 0.00130975 kgm^2.$$

Inercia del péndulo

El momento de inercia del péndulo se calcula como el de un paralepípedo con eje de rotación fuera del centro:

$$\frac{1}{12} m_p(h_p^2 + l_p^2) + m_p d_p^2, \tag{4.2}$$

donde:

- m_p = Masa del péndulo.
- h_p = Altura del péndulo.
- l_p = Anchura del péndulo.
- d_p = Distancia al eje de rotación.

Sustituyendo los valores del bloque del péndulo:

$$\frac{1}{12} 1.2(0.18^2 + 0.15^2) + 1.2(0.09^2) = 0.0097929 kgm^2.$$

Para poder realizar las simulaciones de la sección 3.2 se consideraron los siguientes parámetros de la plataforma integral:

I_w	$0.0013 kgm^2$
I_p	$0.0097 kgm^2$
m_p	$0.800 kg$
m_w	$0.620 kg$
r_w	$0.0675 m$
l	$0.075 m$
R	6Ω
k_m	$0.80287 Nm/A$
k_e	$0.80287 Vs/rad$

Tabla 4.2: Parámetros plataforma experimental

El apéndice D muestra el código en MATLAB para el cálculo de los parámetros de la plataforma experimental.

En el siguiente capítulo se presentan los resultados obtenidos de cada módulo desarrollados en éste y los capítulos anteriores.

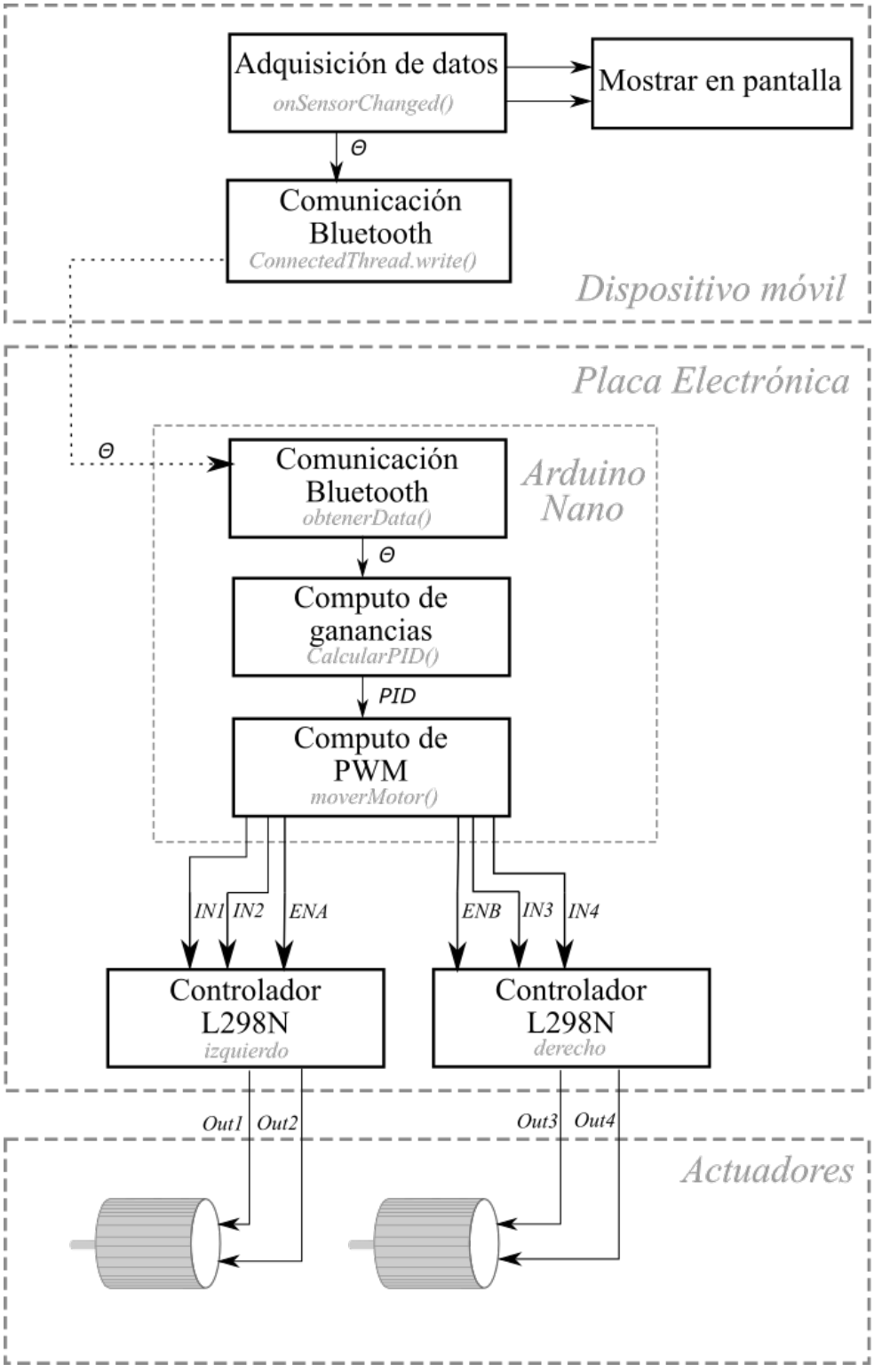


Figura 4.11: Diagrama a bloques del funcionamiento de la plataforma experimental completa.

Capítulo 5

Resultados

5.1. Estructura mecánica

El desarrollo de la plataforma se divide en tres módulos que no son independientes uno de otro. A medida que se desarrollaron los módulos surgieron cambios entre ellos. El diseño de la estructura influyó en la implementación del dispositivo móvil y por lo tanto en el desarrollo de la aplicación. El correcto funcionamiento de la aplicación del dispositivo móvil proporciona la entrada al controlador.

La estructura mecánica de la plataforma comenzó siendo contruida mediante el uso de Makeblock® y ajustada a lo largo de las pruebas realizadas. El primer diseño de la estructura mecánica se muestra en la Figura 5.1(a), se puede observar que la posición del dispositivo móvil se encuentra de manera vertical, es decir, con la pantalla del dispositivo perpendicular al piso. En esta estructura, el péndulo era conformado por el conjunto de baterías, controlador, y el manubrio a donde se fijaba el dispositivo móvil. Los inconvenientes que presentó la estructura mecánica con Makeblock® fueron los motores que la plataforma incluía. Los motores eran muy pequeños para la estructura mecánica y el dispositivo móvil. Debido a que la tarjeta de Makeblock® se conecta a motores y módulo Bluetooth mediante conectores RJ45, lo único que se mantuvo de la plataforma Makeblock® fueron los elementos mecánicos.

Para el diseño de la estructura del segundo prototipo, que se observa en la Figura 5.1(b), se cambiaron los motores por unos de 12V y caja reductora 80:1, se enfatiza la salida de los motores en el par en lugar de la velocidad, porque tiene que oponerse al momento de rotación que la gravedad aplica en la plataforma experimental. La plataforma presentó mejoría, sin embargo, la posición vertical en la que se colocó el dispositivo influye en el equilibrio del robot. El peso del dispositivo móvil se encontraba fuera de la vertical de la plataforma experimental.

Para el tercer prototipo, Figura 5.1(c), se construyó el péndulo como un bloque sobre los actuadores, similar a la Figura 4.2 en el capítulo 1.

Finalmente, la estructura mecánica que se implementó para la plataforma experimental se muestra y describe en el capítulo 4, Figura 4.10.

A partir del prototipo tres, en el que se define la posición en la que el dispositivo móvil se coloca en la plataforma, la aplicación define el eje y como el eje de interés.

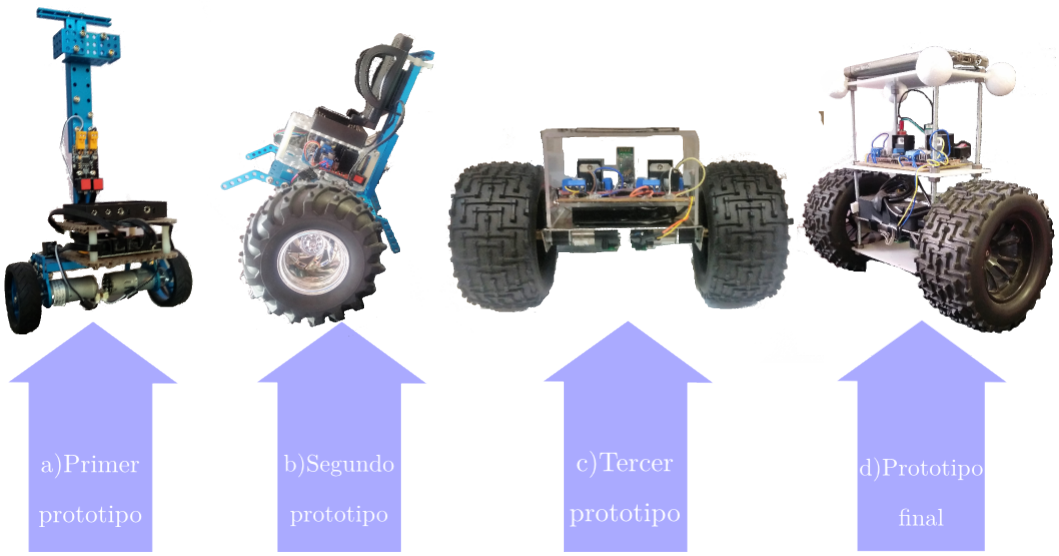


Figura 5.1: Estructuras mecánicas desarrolladas.

5.2. Dispositivo móvil como sensor de orientación

Finalmente la posición del dispositivo móvil se definió en una posición horizontal, con la pantalla hacia arriba (orientación *Landscape* de la aplicación). Los beneficios que ofrece esta implementación se reflejan en la dinámica del robot ya que la ubicación queda dentro del bloque que define al péndulo y no por fuera del centro de gravedad como en el prototipo uno y dos. Los sensores inerciales que proporciona el dispositivo móvil, son capaces de obtener datos en cualquiera de los tres ejes, x, y, y z, para la posición específica en la que el dispositivo móvil va a ser montado, las lecturas se realizan sobre el eje y del dispositivo móvil.

Para la obtención de la orientación del dispositivo móvil, primeramente se utilizó el sensor sintético *TYPE ROTATION VECTOR* cuya respuesta se mostró en la sección 2.2.3.2. En estado estático (con la pantalla hacia arriba), el sensor sintético obtiene mediciones que oscilan entre 180.0 y 180.4. A pesar de los resultados obtenidos, se hicieron pruebas implementando la *fusión de sensores*.

Los datos de inclinación obtenidos mediante la fusión de sensores se presentaron en la sección 2.2.3.3, cuyos datos oscilaban entre 180.0 y 178.2 (ver Figura 5.2).

El problema en la *fusión de sensores* es que, desafortunadamente, si el giroscopio no lee cero cuando está en modo estático, el error en la medición continuará agregándose al ángulo calculado, obteniendo una respuesta alejada de lo real. La estimación que el magnetómetro obtiene del norte magnético se ve fácilmente afectada por objetos metálicos externos. Agregado a esto, la salida de un acelerómetro es bastante ruidosa, cualquier lectura del acelerómetro incluye movimientos bruscos del dispositivo, aún en estado estático, lo cual influye en que la respuesta obtenida por estos sensores basados en hardware sea más ruidosa.

Por otro lado, los datos recibidos del sensor de vector de rotación se procesan de forma muy similar a los datos del acelerómetro y el magnetómetro. Las principales diferencias son que solo se utiliza un sensor y se olvida del procesamiento de cada uno de los datos proporcionados por los sensores.

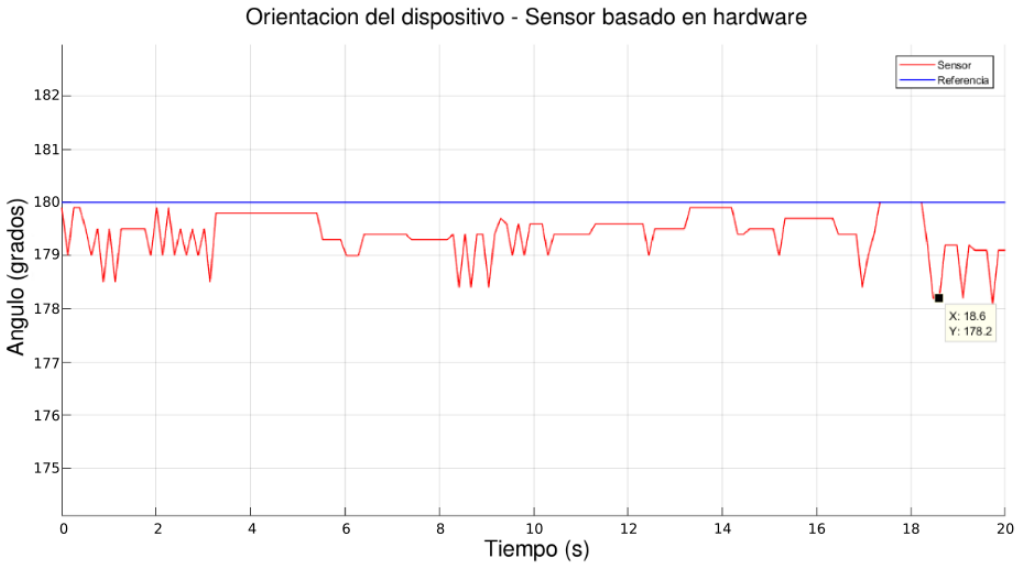


Figura 5.2: Respuesta obtención de inclinación mediante fusión de sensores.

Los algoritmos de **Invense®** (empresa desarrolladora de sensores de hardware para dispositivos móviles) sobre *Sensor Fusion* son exclusivos de la compañía, por lo que estos algoritmos de fusión de sensores no están disposnibles para uso de los desarrolladores. Sin embargo, si en el dispositivo móvil se encuentra disponible, se puede utilizar el sensor sintético *TYPE ROTATION*. Los dispositivos móviles de mayor antiguedad o que usan una marca diferente de sensores *MEMS* pueden utilizar la fusión de sensores de hardware mediante algoritmos propios del desarrollador como alternativa.

Ser dependiente de los datos del sensor de dos fuentes diferentes puede hacer que usar *SensorManager.getOrientation()* sea menos conveniente que usar datos de un solo sensor para detectar cambios simples en la orientación del dispositivo debido al método *getRotationMatrix ()*. La razón para evitar este método para producir una matriz de rotación es que la orientación debe ser relativamente estática y los datos que provee la *fusión de sensores* son muy ruidosas.

5.3. Integración de la plataforma experimental

La plataforma experimental final integra el dispositivo móvil como sensor de inclinación utilizando el sensor sintético de Android®. En la figura 5.3, la gráfica muestra la respuesta a lazo cerrado del control de balanceo de la plataforma experimental final tomando como punto de equilibrio 180° (con la pantalla del dispositivo móvil hacia arriba). En la gráfica de la Figura 5.3 se observa que la señal medida por el dispositivo móvil se mantiene en una vecindad muy cercana a los 180°. Sin embargo, la gráfica es de la respuesta de la plataforma experimental sin exponerse a alguna perturbación. La respuesta de la plataforma muestra un error en estado estacionario que se mantiene acotado en una región de ±0.5° (179.5° y 180.5°).

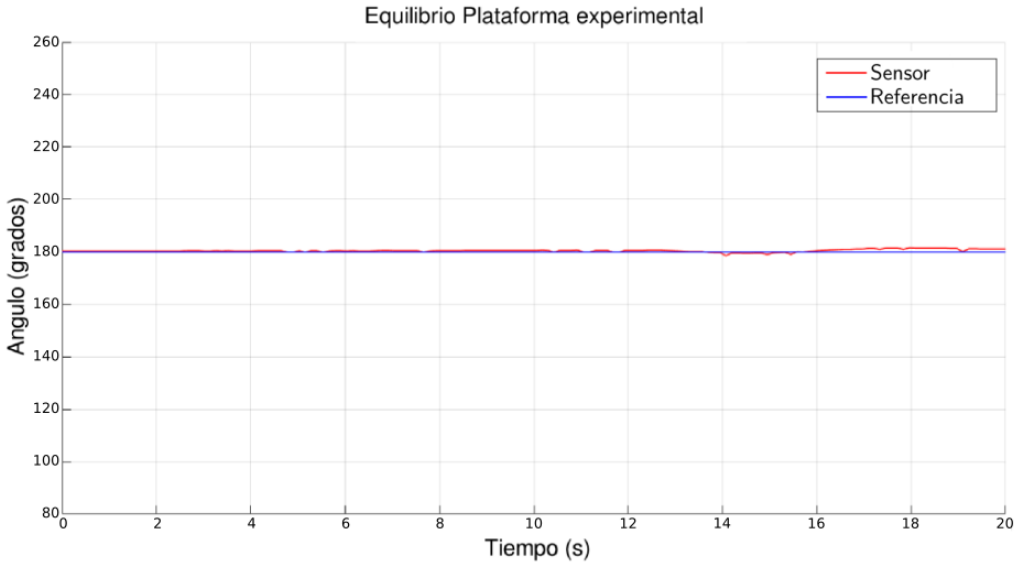


Figura 5.3: Respuesta de la plataforma experimental.

El par que los motores aplican como respuesta para lograr el balanceo de la plataforma no es grande debido a la pequeña diferencia entre el error y la referencia, lo cual no requiere de una acción drástica por parte de los actuadores. Obsérvese que el comportamiento dinámico del sistema se mantiene suave debido a que no existe perturbación aplicada a la plataforma experimental. La respuesta del controlador no es "golpeada" como se puede observar en la gráfica. Es importante recalcar que este comportamiento "ideal" sucede solo en una pequeña región cerca del punto de equilibrio.

Para observar como responde la plataforma experimental ante una perturbación, la gráfica de la Figura 5.5 muestra la respuesta de la plataforma ante esta situación. Considérese como perturbación una fuerza exógena que obliga a la plataforma a salir de su punto de equilibrio lo que provoca un incremento en el error medido por el sistema.

En la gráfica de la Figura 5.5 se puede observar que los valores entre los que oscila la plataforma experimental se encuentran en la región de los $\pm 1.5°$. La Figura 5.4 muestra el rango de trabajo dentro de la vecindad de $\pm 1.5°$ que son los valores entre los que la plataforma oscila.

Se puede apreciar que aunque suena como una desviación insignificante, la posición inclinada de la plataforma es notoria. Ante un error pequeño medido, pero que obligue al robot a salir de su punto de equilibrio (dentro de una vecindad muy cercana a la vertical de $180°$), la plataforma es capaz de mantener el equilibrio oscilando entre valores de $177.5°$ y $182.5°$ ($\pm 2.5°$).

La Figura 5.6 muestra a la plataforma desviada de la vertical por $2.5°$, a pesar de ser solo un grado más de desviación la inclinación de la plataforma es más drástica, lo que influye a un mayor desequilibrio.

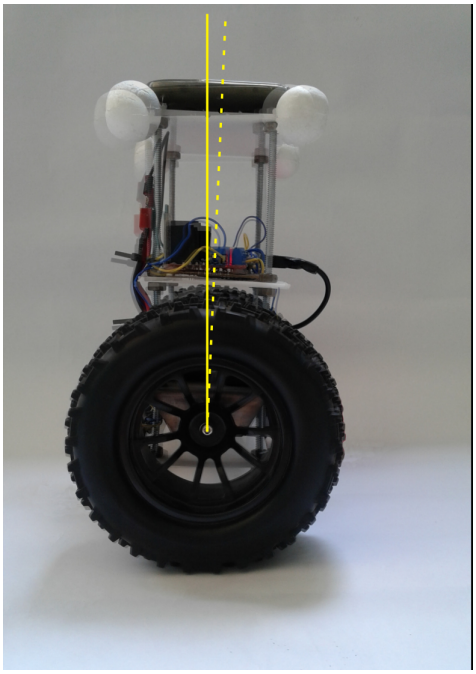


Figura 5.4: Desviación de la plataforma 1.5°.

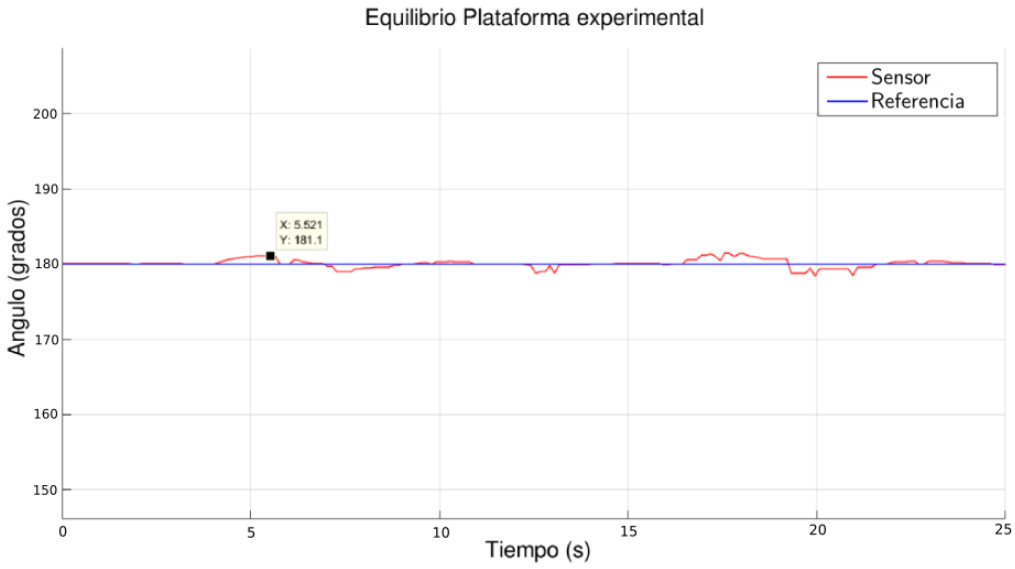


Figura 5.5: Respuesta de la plataforma experimental ante perturbación pequeña.

Las respuestas de las gráficas muestran como la plataforma experimental oscila entre diferentes valores cercanos a los 180° pero luego de varias oscilaciones muy suaves es capaz de alcanzar el equilibrio sobre la vertical y mantenerse en esa posición.

La Figura 5.7 muestra el movimiento de la plataforma cuando se alcanza el equilibrio luego de varias oscilaciones dentro de la región cercana a la vertical.

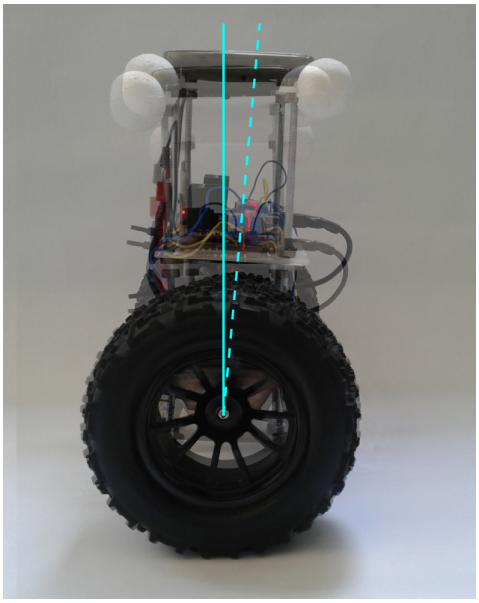


Figura 5.6: Desviación de la plataforma 2.5°.

La plataforma es capaz de alcanzar la vertical luego de 3 segundos en la cuarta oscilación.

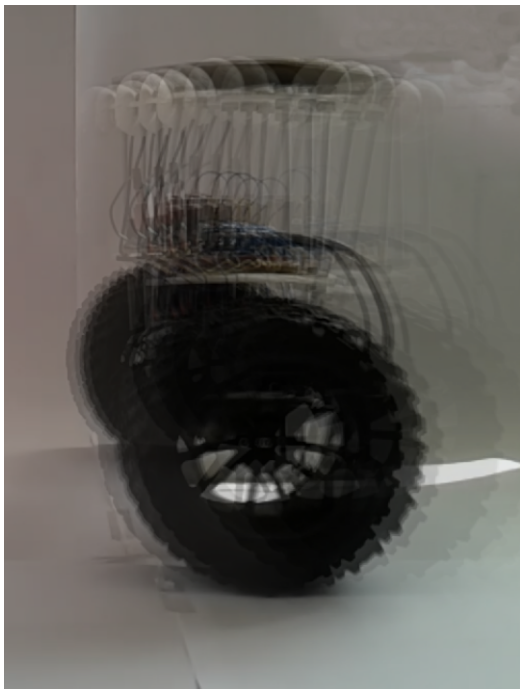


Figura 5.7: Movimiento de la plataforma guardando equilibrio.

En los momentos en que el robot sale de la vertical, los cambios detectados por el dispositivo móvil son tan rápidos que le permiten al controlador obtener un error pequeño y actuar de manera robusta manteniendo a la plataforma en equilibrio sin oscilaciones drásticas. Sin embargo, cuando se aplica una perturbación mayor el sistema dinámico tiende a provocar

oscilaciones en el equilibrio. A pesar que el sistema se mantiene acotado, el controlador no puede alcanzar el estado estacionario de vuelta. En la gráfica de la Figura 5.8 se observa que las oscilaciones de la plataforma experimental se presentan con mayor frecuencia y con mayor amplitud.

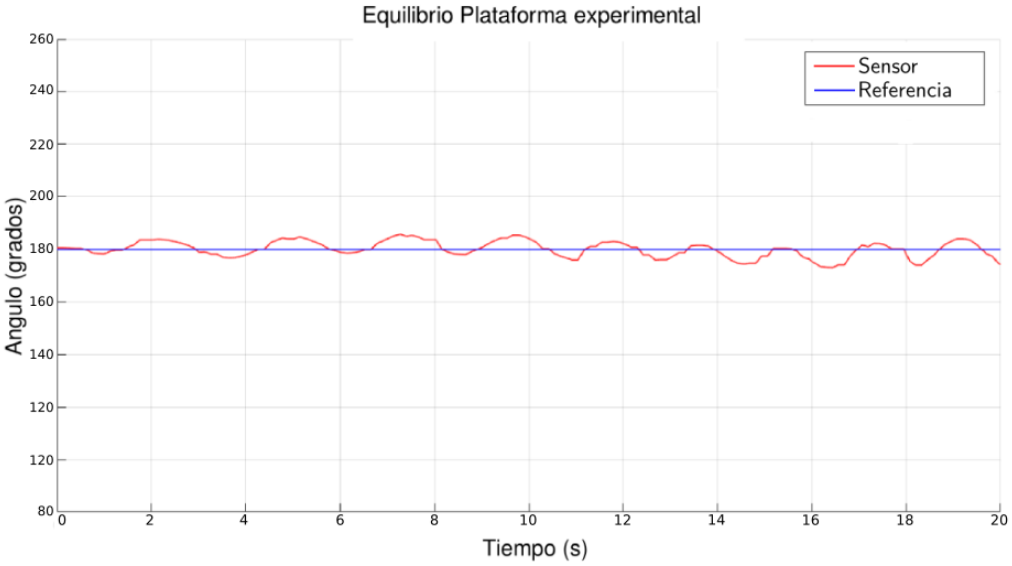


Figura 5.8: Respuesta de la plataforma para guardar el equilibrio.

Cuando la plataforma experimental inicia en una posición fuera de la región de equilibrio o se expone ante una perturbación más grande, el controlador trata de compensarlo aplicando un voltaje que logre que los motores desplacen la plataforma (hacia adelante o atrás, según sea el caso) para volver al punto de equilibrio y disminuir el error obtenido. Sin embargo, como se muestra en la gráfica de la Figura 5.9 el robot no alcanza a equilibrarse sobre la vertical y se mantiene avanzando tratando de compensar el error.

La inclinación medida por el dispositivo móvil no vuelve a acercarse a la referencia de los 180°, sin embargo, el voltaje aplicado a los motores hacen que la plataforma avance tratando de compensar el error y lo mantienen de pie sin caerse pero avanzando sin alcanzar la vertical.

La Figura 5.10 muestra el movimiento que tiene la plataforma experimental cuando trata de alcanzar el equilibrio ante respuesta de una perturbación que la obliga a salir de su zona de equilibrio.

Las oscilaciones que realiza la plataforma son entre cinco y seis oscilaciones cuyo ángulo medido va incrementando observándose oscilaciones bruscas. En la séptima ocilación el ángulo medido se ha vuelto tan grande que la plataforma tiende a avanzar sin alcanzar la referencia y llegar al punto de caer.

En este capítulo se pudo apreciar que se alcanzaron los objetivos deseados de controlar el balanceo de un péndulo móvil con ruedas mediante el uso de un dispositivo celular. La integración de los diferentes elementos produjo resultados satisfactorios, los cuales son susceptibles de ser mejorados en el futuro, sea mediante el uso de otras estrategias de control en la plataforma actual o mediante la modificación de la plataforma con el cambio de sensores, por ejemplo.

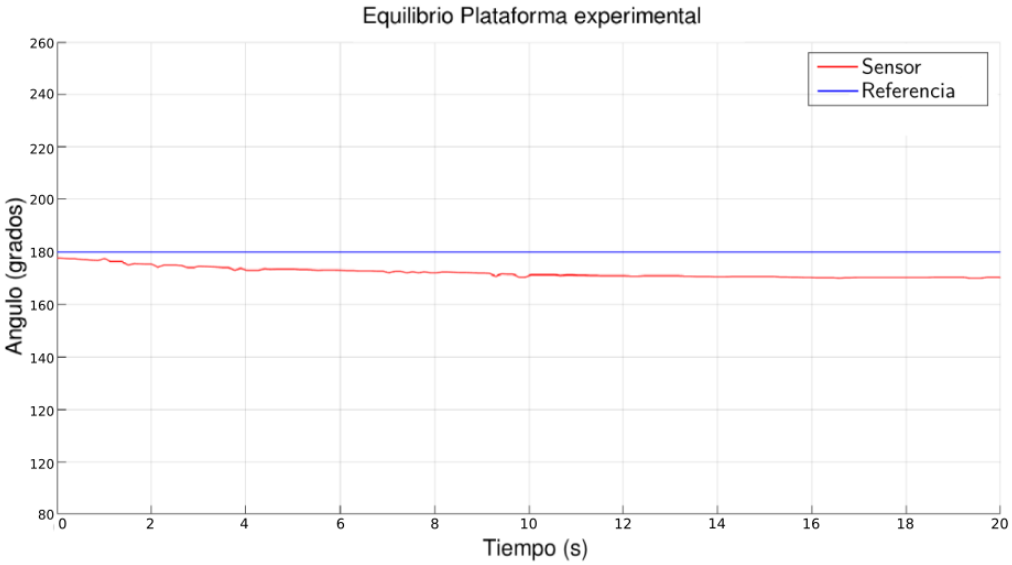


Figura 5.9: Respuesta de la plataforma experimental con posición inicial fuera de la vertical.

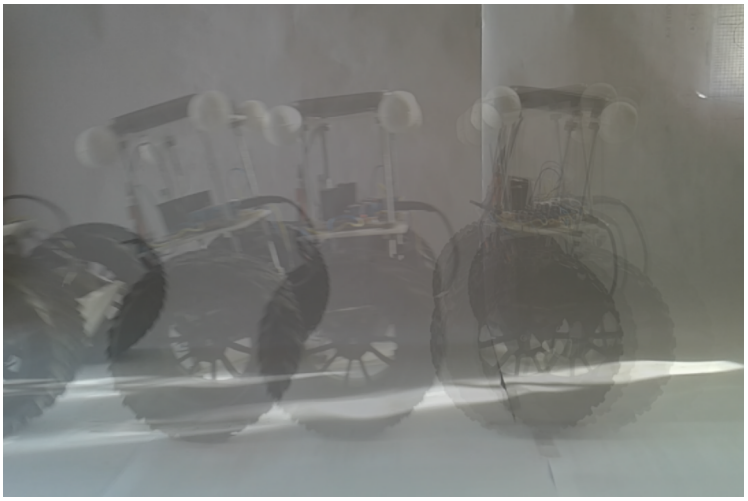


Figura 5.10: Movimiento de la plataforma tratando de guardar equilibrio.

En el siguiente capítulo se presentan las conclusiones del presente trabajo de tesis con base en el análisis de los resultados. Se presentan también propuestas de trabajo futuro del trabajo desarrollado.

Capítulo 6

Conclusiones y trabajos futuros

6.1. Conclusiones

En este trabajo se logró diseñar un inclinómetro usando los sensores de un dispositivo móvil basado en Android® para el balanceo de un robot de transporte tipo péndulo invertido móvil, a través de una aplicación para dispositivos móviles Android® capaz de actuar como alternativa a los sensores inerciales.

Estas son algunas conclusiones que se derivan de este trabajo:

- El desarrollo e implementación de un prototipo requiere de la investigación de diferentes áreas de estudio, en la cual ninguno es independiente.
- El rediseño de la estructura mecánica influyó en el comportamiento satisfactorio de la plataforma experimental.
- Identificar cuando son apropiados las diferentes técnicas de fusión es parte importante para la utilidad de los datos que proporcionan los sensores en el desarrollo de la aplicación.
- El controlador implementado en la plataforma tiene resultados satisfactorios en una región pequeña cercana a la vertical.
- En condiciones de perturbación la respuesta es capaz de mantener el equilibrio cuando el sistema tiene un error acotado de $\pm 1.7\,\%(\pm 3^\circ)$.
- La plataforma alcanza un ángulo de 176.2° con desplazamiento constante antes de que el error incremente y la plataforma caiga.
- El uso de dos tipos de sensores para medir la orientación del dispositivo requiere mayor procesamiento de datos y los resultados pueden no ser mejores a los de un sensor sintético de Android®.
- Una de las desventajas que presenta al dispositivo móvil como sensor de orientación es que todos los recursos del dispositivo limitan su funcionamiento a la ubicación en la que el dispositivo móvil se encuentra.
- Siempre que sea posible, es preferible usar sensores sintéticos en lugar de usar datos de magnetómetro y acelerómetro crudos, para no enfocarse en algoritmos ya implementados y probados.

- Las técnicas patentadas de fusión de sensores (como sensores sintéticos en Android®) podrían ser utilizadas en una llamada a una API existente de Android y puede evitar que el desarrollador implemente o use una técnica de fusión.

6.2. Trabajo futuro

Se proponen las siguientes mejoras:

- Aplicar otros algoritmos de control, como retroalimentación de todos los estados.
- Mejorar la estructura mecánica de la plataforma.
- Diseñar una tarjeta electrónica con componentes específicos para la plataforma experimental.
- Abundar en el estudio de *Sensor Fusion* para aprovechar mejor los recursos del dispositivo móvil.
- Una tarea importante es llevar los trabajos realizados a una aplicación específica como vehículo de transporte.
- Implementación de control remoto del desplazamiento de la plataforma desde dispositivo móvil.
- Implementación de otro tipo de sensores, como encoders, para tener una plataforma más completa.
- Una tarea importante es llevar los trabajos realizados a una aplicación específica como vehículo de transporte.

Bibliografía

[1] Z.Li, C.Yang y L.Fan, *Advanced Control of Wheeled Inverted Pendulum Systems.*, Springer, London, 2013.

[2] Hyungjik, Lee y Seul Jung., *Balancing and navigation control of a mobile inverted pendulum robot using sensor fusion of low cost sensors,* Mechatronics, ELSEVIER (2012), págs. 95-105.

[3] Juang Hau-Shiue y Kai-Yew Lum. "Desing and Control of a Two-Wheel Self-Balancing Robot using Arduino Microcontroller Board". En: *10th IEEE International Conference on Control and Automation (ICCA)*. Hangzhou, China, 2013.

[4] J.Z, Sasiadek y P. Hartana, "Sensor data fusion using Kalman filter". En: *III International Conference on Information Fusion*. Vol 2. 2000, págs. 19-25.

[5] Ammar, Wasif y Danish Raza, "Design and Implementation of a Two Wheel Self Balancing Robot with a Two Level Adaptive Control". En: *VIII International Conference on Digital Information Management (ICDIM)*. 2013.

[6] Miseon, Han y KyungHwan Kim y DoYoun Kim y Jaesung Lee, "Implementation of Unicycle Segway Using Unscented Kalman Filter in LQR control". En: *X International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*. 2013.

[7] Castro, Arnoldo y Adams Christopher y Singhose William, "Dynamic Response Characteristics of a Two-Wheeled Inverted-Pendulum Transporter". En: *52nd IEEE Conference on Decision and Control*. 2013.

[8] Volker Kempe, *Inertial MEMS: Principles and Practice*. 1ra ed. Cambridge University Press, 2011.

[9] Péndulo invertido en tableta Android. Accedido Febrero 2018. Online. Disponible en: https://ichiro-maruta.blogspot.com/2013/08/inverted-pendulum-android.html

[10] Corredor, Óscar F y Pedraza Luis F, "Tecnología Bluettoth: Alternativa para redes celulares de voz y datos,Revista Visión Electrónica, año 3, pp. 73-84, Noviembre de 2009.

[11] Servín Cardozo, Gary Stephen y Sandoval Vera Luis Miguel, "Prototype for a Self-Balanced Personal Transporter," *Workshop on Engineering Applications (WEA)*, Bogota, Colombia, 2012.

[12] Rich Chi Ooi, "Balancing a Two-Wheeled Autonomous Robot ," M. Eng. thesis, University of Western Australia, 2003.

[13] Segway. Accedido Noviembre 2017. Online. Disponible en: http://www.segway.com/about/our-story

[14] Android Developers. Accedido Noviembre 2017. Online. Disponible en: https://developer.android.com

[15] Milette, Greg y Adam Stroud, *Professional Android Sensor Programming*. John Wiley and Sons, Inc., 2012.

[16] Control Tutorials for MATLAB and SIMULINK. Accedido Diciembre 2017. Online. Disponible en: http://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum

[17] H. Eren y S. Makinist y E. Akin, "Estimating Driving Behavior by a Smartphone". En: *Alcalá de Hemares, España*. 2012.

[18] Improving the Beginner's PID. Accedido Enero 2018. Online. Disponible en: http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/

[19] Ramírez A. Jorge Alonso y Hernández H. José Luis y Orejel S. Luis Ricardo y Olguin B. Ubaldo Jair, "Dispositivo mecatrónico móvil autónomo para la exploración interna de tuberías," Tesis, Instituto Politécnico Nacional, 2015.

[20] Tesón M. Eva, "Desarrollo de una práctica docente de estimación de la orientación basada en fusión de sensores de Smartphones," Trabajo fin de Máster, Escuela Técnica Superior de Ingeniería Universidad de Sevilla, 2017.

[21] Hoja de datos motores Namiki. Accedido Enero 2018. Online. Disponible en: http://www.namikisingapore.com.sg/product/motor/pdf

[22] Ogata Katsuhiko, *Ingeniería de control moderna*. 5ed. Pearson, 2010.

[23] Bjorn Carlsson y Per Orback, "Mobile Inverted Pendulum Control of an unestable process using Open Source Real-Time Operating System," Maestría en Ciencias Tesis, Chalmers University of Technology, 2009.

Apéndice A

Código en Android® para el uso de sensores

```
1  package com.example.carolina.segway_vf;
2
3  //Se importan algunos paquetes
4  import android.app.Activity;
5  import android.hardware.Sensor;
6  import android.hardware.SensorEvent;
7  import android.hardware.SensorEventListener;
8  import android.content.pm.ActivityInfo;
9  import java.math.BigDecimal;
10 import android.content.Intent;
11 import android.os.Handler;
12 import android.widget.TextView;
13 import android.widget.Toast;
14 import java.util.ArrayList;
15
16 public class MainActivity extends Activity implements SensorEventListener {
17   //Elementos para manipular en la interfaz
18   TextView textViewX;
19   TextView textoEstado;
20   //Elementos para los datos obtenidos
21   // Valores para magnetometro
22   final private float[] magnitudeValues = new float[3];
23
24   //Valores acelerometro
25   final private float[] accelerometerValues = new float[3];
26
27   //Valores inclinometro
28   final float[] inclinationValues = new float[16];
29   float axisoX, axisoY, axisoZ;
30   //Para acceder a los recursos (SENSORES)
31   private SensorManager mSensorManager;    //  SensorManager
32   private Sensor acelerometro, magnetometro, gravedad, rotacion; //nombre de
         cada uno de los sensores a utilizar
33
34   float[] rotationV = new float[9];
35   float[] adjustedRotationMatrix = new float[9];
36   float[] orientation = new float[3];
37   float[] orientation2 = new float[3];
```

```
38
39  private static int ref = 180;
40
41  //Sobre escribimos algunos metodos del Activity para el desarrollo de la
        aplicacion
42  @Override
43  public void onCreate(Bundle savedInstanceState) {
44  //el metodo onCreate para la inicializacion de los recursos de la
        aplicacion
45  super.onCreate(savedInstanceState);
46  setContentView(R.layout.activity_main); //layout
47  //para bloquear el giro automatico de la pantalla
48  setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
49
50  //Registro del listener para el sensor
51  mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
52  //"Alta" de los sensores que se utilizan
53  acelerometro = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
54  magnetometro = mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
55  rotacion = mSensorManager.getDefaultSensor(Sensor.TYPE_ROTATION_VECTOR);
56
57  }
58
59  @Override
60  public void onResume() {
61      super.onResume();
62      textViewX = (TextView) findViewById(R.id.voX);
63
64      //Manager del sensor
65      //A cada sensor se le asigna un listener con delay definido
66      mSensorManager.registerListener(this, acelerometro, SensorManager.
            SENSOR_DELAY_UI);
67      mSensorManager.registerListener(this,magnetometro,SensorManager.
            SENSOR_DELAY_UI);
68      mSensorManager.registerListener(this,rotacion,SensorManager.
            SENSOR_DELAY_GAME);
69  }
70
71  @Override
72  public void onPause()
73  {
74      super.onPause();
75      }
76  //para cada que el valor del sensor cambie
77  @Override
78  public final void onSensorChanged (SensorEvent event) {
79      float [] data;
80      try {
81              if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD) {
82              System.arraycopy(event.values, 0, magnitudeValues, 0,
                    magnitudeValues.length);
83      } else if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
84              System.arraycopy(event.values, 0, accelerometerValues, 0,
                    accelerometerValues.length);
85      }
86
87      if (magnitudeValues != null && accelerometerValues != null) {
88          // Fusiona valores de acelerometro y magnetometro
```

```
89              SensorManager.getRotationMatrix(currentOrientationRotationMatrix.
                    matrix, inclinationValues, accelerometerValues,
90                      magnitudeValues);
91              // Transforma la matriz de rotacion a cuaternion
92              currentOrientationQuaternion.setRowMajor(
                    currentOrientationRotationMatrix.matrix);
93          }
94
95          if (event.sensor.getType() == Sensor.TYPE_ROTATION_VECTOR) {
96                          //Convertir el vector de rotacion a matriz 4x4
97              SensorManager.getRotationMatrixFromVector(rotationV, event.
                    values);
98              //Ajuste de coordenadas
99              SensorManager.remapCoordinateSystem(rotationV, SensorManager.
                    AXIS_Y, SensorManager.AXIS_X, adjustedRotationMatrix);
100             //Obtencion de la orientacion
101             SensorManager.getOrientation(adjustedRotationMatrix,
                    orientation);
102             orientation2[0] = (float) Math.toDegrees(orientation[0]);
103             orientation2[1] = (float) Math.toDegrees(orientation[1]);
104             orientation2[2] = (float) Math.toDegrees(orientation[2]);
105             showResults(orientation2[1]);
106         }
107     }catch(Exception e){
108         System.out.println(e.getMessage());
109     }
110 }
111
112 @Override
113 public void onAccuracyChanged(Sensor sensor, int i) {
114
115 }
116
117 public static float round(float value, int decimalPlace) {
118     BigDecimal bd = new BigDecimal(Float.toString(value));
119     bd = bd.setScale(decimalPlace, BigDecimal.ROUND_HALF_UP);
120     return bd.floatValue();
121 }
122
123 public void showResults (float valor3){
124     float data;
125     data = valor3+180;
126     data = round(data,1);
127     try {
128         nuevo_dato(data);
129         textViewX.setText(""+(data));
130         if (comunicar.isChecked()&& (data>100)) {
131             //se va a enviar el error
132             mConnectedThread.write("@");
133             mConnectedThread.write(Float.toString(data));
134         }
135     }
136
137     catch (Exception e){System.out.println(e.getMessage());}
138     //el valor 3 es el que nos interesa en esta configuracion
139 }
140}
```

Apéndice B

Código en Android® para mostrar dispositivos enlazados

```
1  package com.example.carolina.segway_vf;
2
3  import android.app.Activity;
4  import java.util.Set;
5  import android.bluetooth.BluetoothAdapter;
6  import android.bluetooth.BluetoothDevice;
7  import android.content.Intent;
8  import android.os.Bundle;
9  import android.util.Log;
10 import android.view.View;
11 import android.widget.AdapterView;
12 import android.widget.ArrayAdapter;
13 import android.widget.Button;
14 import android.widget.ListView;
15 import android.widget.TextView;
16 import android.widget.Toast;
17 import android.widget.AdapterView.OnItemClickListener;
18
19
20 //Para la lista de dispositivos enlazados
21
22
23 public class DeviceListActivity extends Activity {
24   // Debugging for LOGCAT
25   private static final String TAG = "DeviceListActivity";
26   private static final boolean D = true;
27   TextView textView1;
28   // cadena extra para enviar al main
29   public static String EXTRA_DEVICE_ADDRESS = "device_address";
30   //adaptador que necesitamos para la comunicacion bluetooth
31   private BluetoothAdapter mBtAdapter;
32   //array de los disspositivos enlazados
33   private ArrayAdapter<String> mPairedDevicesArrayAdapter;
34   //funcion para verificar el estado del bluetooth
35   private void checkBTState() {
36       //Se obtiene el adaptador por default
37       try{
38           mBtAdapter=BluetoothAdapter.getDefaultAdapter();
```

```
39          if(mBtAdapter==null) { //si el adaptador es nulo, el dispositivo no
                lo soporta
40              Toast.makeText(getBaseContext(), "El dispositivo no soporta
                    Bluetooth", Toast.LENGTH_SHORT).show();//muestra mensaje
41          } else {
42              if (mBtAdapter.isEnabled()) {
43                  Log.d(TAG, "...Bluetooth Activado...");
44              } else {
45                  //Intenta habilitar el bluetooth
46                  Intent enableBtIntent = new Intent(BluetoothAdapter.
                        ACTION_REQUEST_ENABLE);
47                  startActivityForResult(enableBtIntent, 1);
48              }
49          }
50      }catch (Exception e){
51          System.out.println(e.getMessage());
52      }
53
54  }
55
56  @Override
57  protected void onCreate(Bundle savedInstanceState) {
58      super.onCreate(savedInstanceState);
59      //cuando inicie, el layout que se muestra esel de los dispositivos
            enlazados
60      setContentView(R.layout.device_list);
61
62  }
63
64  @Override
65  public void onResume()
66  {
67      super.onResume();
68      try{
69          //Verificamos la compatibilidad
70          checkBTState();
71          textView1 = (TextView) findViewById(R.id.connecting);
72          textView1.setTextSize(20);
73          textView1.setText(" ");
74          mBtAdapter = BluetoothAdapter.getDefaultAdapter();
75
76          // Inicializacion del array donde se guardaran los dispositivos
                sincronizados
77          mPairedDevicesArrayAdapter = new ArrayAdapter<String>(this, R.
                layout.device_name);
78          // Lista para los dispositivos emparejados
79          ListView pairedListView = (ListView) findViewById(R.id.
                paired_devices);
80          pairedListView.setAdapter(mPairedDevicesArrayAdapter);
81          pairedListView.setOnItemClickListener(mDeviceClickListener);
82          // Obtener el adaptador Bluetooth
83
84
85          //Se obtienen los dispositivos que se han sincronizado con el movil
86          Set<BluetoothDevice> pairedDevices = mBtAdapter.getBondedDevices();
87          // se agregan a una lista de dispositivos
88          if (pairedDevices.size() > 0) {
```

```
89                  findViewById(R.id.title_paired_devices).setVisibility(View.
                        VISIBLE);
90                  for (BluetoothDevice device : pairedDevices) {
91                      mPairedDevicesArrayAdapter.add(device.getName() + "\n" +
                            device.getAddress());
92                  }
93              } else {
94                  String noDevices = getResources().getText(R.string.app_name).
                        toString();
95                  mPairedDevicesArrayAdapter.add(noDevices);
96              }
97
98          }catch (Exception e){
99              System.out.println(e.getMessage());
100         }
101     }
102     //Activar los listeners on-click listener para la lista de dispositivos (
            nicked this - unsure)
103     private OnItemClickListener mDeviceClickListener = new OnItemClickListener
            () {
104
105         public void onItemClick(AdapterView<?> av, View v, int arg2, long arg3)
                {
106             try{
107                 // La direccion MAC son los ultimos 17 caracteres en el view
108                 String info = ((TextView) v).getText().toString();
109                 //obtenemos la direccion MAC del dispositivo
110                 int inicio = info.length();
111                 String address = info.substring(inicio-17 );
112                 String name = info.substring(0,inicio-18);
113                 textView1.setText("Conectando con: "+ name);
114
115                 // Make an intent to start next activity while taking an extra
                        which is the MAC address.
116                 Intent i = new Intent(DeviceListActivity.this, MainActivity.
                        class);
117                 i.putExtra(EXTRA_DEVICE_ADDRESS, address);
118                 startActivity(i);
119             }catch(Exception e ){
120                 System.out.println(e.getMessage());
121             }
122
123
124         }
125     };
126 }
```

Apéndice C

Código en Android® para iniciar comunicación Bluetooth®

```
1  package com.example.carolina.segway_vf;
2
3  //Se importan algunos paquetes
4  import android.app.Activity;
5  import android.os.Bundle;
6  import android.view.View;
7  import java.util.UUID;
8  import android.bluetooth.BluetoothAdapter;
9  import android.bluetooth.BluetoothDevice;
10 import android.bluetooth.BluetoothSocket;
11 import android.content.Intent;
12 import android.os.Handler;
13 import java.io.IOException;
14 import java.io.InputStream;
15 import java.io.OutputStream;
16 import java.util.ArrayList;
17
18 public class MainActivity extends Activity {
19
20   Handler bluetoothIn;
21   final int handlerState = 0;//para identificar handler message
22   private StringBuilder recDataString = new StringBuilder();
23   //elementos necesarios para iniciar una conexion Bluetooth
24   private BluetoothSocket BTSocket; //socket Bluetooth
25   private BluetoothAdapter BTAdapter; //adaptador Bluetooth
26   private static final UUID BTMODULEUUID = UUID.fromString("
         00001101-0000-1000-8000-00805F9B34FB"); //Se define un valor UUID para
         establecer la comunicacion Bluetooth
27   private static String address = null; //Direccion MAC del dispositivo
28   //Para metodos del ConnectedThread
29   private ConnectedThread mConnectedThread; // Se declara una variable tipo
         ConnectedThread para acceder a los metodos necesarios en la
         comunicacion Bluetooth
30   //Crea el socket del Bluetooth
31   private BluetoothSocket createBluetoothSocket(BluetoothDevice device)
         throws IOException {
32       return  device.createRfcommSocketToServiceRecord(BTMODULEUUID);
33       //creates secure outgoing connecetion with BT device using UUID
```

```
34  }
35
36  private static final String TAG = "DeviceListActivity";
37  private static final boolean D = true;
38  TextView textView1;
39  // cadena extra para enviar al main
40  public static String EXTRA_DEVICE_ADDRESS = "device_address";
41  //adaptador que necesitamos para la comunicacion bluetooth
42  private BluetoothAdapter mBtAdapter;
43  //array de los disspositivos enlazados
44  private ArrayAdapter<String> mPairedDevicesArrayAdapter;
45
46  //funcion para verificar el estado del bluetooth
47  private void checkBTState() {
48      //Se obtiene el adaptador por default
49      try{
50          mBtAdapter=BluetoothAdapter.getDefaultAdapter();
51          if(mBtAdapter==null) { //si el adaptador es nulo, el dispositivo no
                lo soporta
52              Toast.makeText(getBaseContext(), "El dispositivo no soporta
                    Bluetooth", Toast.LENGTH_SHORT).show();//muestra mensaje
53          } else {
54              if (mBtAdapter.isEnabled()) {
55                  Log.d(TAG, "...Bluetooth Activado...");
56              } else {
57                  //Intenta habilitar el bluetooth
58                  Intent enableBtIntent = new Intent(BluetoothAdapter.
                        ACTION_REQUEST_ENABLE);
59                  startActivityForResult(enableBtIntent, 1);
60              }
61          }
62      }catch (Exception e){
63          System.out.println(e.getMessage());
64      }
65
66
67
68  //Sobre escribimos algunos metodos del Activity para el desarrollo de la
        aplicacion
69  @Override
70  public void onCreate(Bundle savedInstanceState) {
71      //el metodo onCreate para la inicializacion de los recursos de la
            aplicacion
72      super.onCreate(savedInstanceState);
73      setContentView(R.layout.activity_main); //layout
74      //Creamos el Adaptador Bluetooth
75      BTAdapter = BluetoothAdapter.getDefaultAdapter();
76      //Verificamos si es compatible con nuestro dispositivo
77      checkBTState();
78  }
79
80  @Override
81  public void onResume() {
82      super.onResume();
83      textViewX = (TextView) findViewById(R.id.voX);
84      //Se va a obtener la direccion MAC del dispositivo que se va a enlazar
85      Intent intent = getIntent();    //Intent
```

```
86        address = intent.getStringExtra(DeviceListActivity.EXTRA_DEVICE_ADDRESS
              );//Se obtiene la direccion MAC del dispositivo
87        BluetoothDevice device = BTAdapter.getRemoteDevice(address);     //Se
              crea un dispositivo remoto con la direccion MAC proporcionada
88        try {
89            BTSocket = createBluetoothSocket(device);    //Crea un socket del
                  dispositivo Bluetooth
90        } catch (IOException e) {
91            Toast.makeText(getBaseContext(), "La creaccion del Socket fallo",
                  Toast.LENGTH_LONG).show();     //Si el intento es fallido
92        }
93        try
94        {
95            BTSocket.connect(); //Establecer la conexion del socket
96        } catch (IOException e) {
97            try
98            {
99                Toast.makeText(getBaseContext(), "Salga y vuelva a entrar a la
                      aplicacion", Toast.LENGTH_LONG).show();
100               BTSocket.close();    //Si no se puede, intenta cerrar el socket
101           } catch (IOException e1) {
102               Toast.makeText(getBaseContext(), "no conecta el socket", Toast.
                      LENGTH_LONG).show();
103           }
104       }
105
106       //Inicio de la conexion con el dispositivo
107       mConnectedThread = new ConnectedThread(BTSocket);    //Se declara una
              variable tipo ConnectedThread del socket Bluetooth
108       mConnectedThread.start();                            //Inicia la
              conexion
109   }
110
111   @Override
112   public void onPause()
113   {
114       super.onPause();
115       try
116       {
117           BTSocket.close();}
118       catch (IOException e2) { }
119   }
120
121  }
122
123}
```

Apéndice D

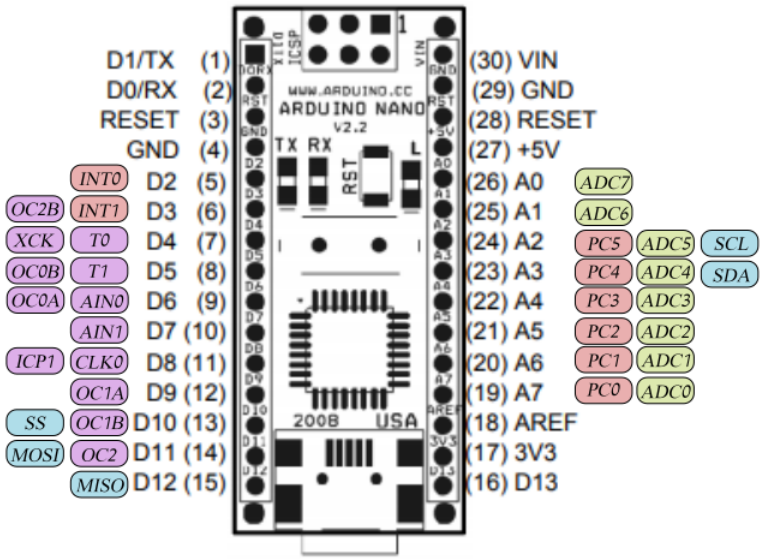
Programa en Matlab para simulaciones

```
% Par\'ametros del motor de CD NAMIKI 22CL-3501PG
%-----------------------------------------------
R = 6;
L = 104.3e-6;
J = 3.54e-3;
B = 7.55994e-3;
k = 802.8724e-3;
%-----------------------------------------------
%radio de las llantas
%-----------------------------------------------
r = .135/2;
%-----------------------------------------------
%caja reduccion de los motores
%80:1
%-----------------------------------------------
%masa total 1.315
%par\'ametros de la plataforma carro-pendulo
M = .620;     %masa del carro
m = 1.200;    %masa del pendulo
g = 9.81;     % gravedad
l = 0.065;     %distancia al centro de masa del pendulo
%b = 0.1;     %friccion entre las llantas y el piso
%Para la inercia del pendulo
hp = 0.18;
lp = 0.15;
dp = 0.09;
I = (hp^2+lp^2)*m/12 + (m*dp^2);  %Inercia del pendulo
%Para la inercia de las llantas
mwheel = 0.310*2;   % Masa de las llantas
Iw = ((1/2)*mwheel*r^2);
a = (I + m*l^2);
b = (2*M + (2*Iw/(r^2)) + m);
%-----------------------------------------------
%para el espacio de estados
beta = (2*M) + (2*Iw/r^2) + m;
alpha = (I*beta) + (2*M*L^2*(m + (Iw/r^2)));

Ae = [0 1 0 0;
```

```
  0 (2*k*k*(m*l*r-I-m*l^2))/(R*r^2*alpha)  (m^2*g*l^2)/alpha 0;
  0 0 0 1;
  0 (2*k*k*(r*beta - m*l))/(R*r^2*alpha) (m*g*l*beta)/alpha 0];

Be = [0;
  2*k*((I+ m*l^2 - m*l*r))/(R*r*alpha);
  0;
  (2*k*(m*l - r*beta))/(R*r*alpha)];

C = [0 0 1 0];

D = 0;

Ce = [1 0 0 0;
  0 0 1 0];

De = [0;
  0];
%-----------------------------------------------
%Funcion de transferencia
[num1, den1] = ss2tf(Ae,Be,C,D);
sys = tf(num1,den1)
%sys es la funcion en lazo abierto
%-----------------------------------------------
%Analisis
%-----------------------------------------------
rlocus(num1,den1)
title('Lugar geometrico de las raices pendulo invertido movil con llantas');
xlabel('Eje Real');
ylabel('Eje Imag');
figure;
%Respuesta al impulso
impulse(num1,den1,T);
title('Respuesta del sistema al impulso en lazo abierto');
ylabel('Posicion[m], Angulo[rad]');
xlabel('Tiempo[s]');
legend('Posicion del robot','Angulo de inclinacion');
%-------------------------------------------------------------
Kp = -205.1423;
Ki = -5.5765;
Kd = -58.5039;
Tf = 1;
C = pid(Kp, Ki, Kd, Tf)
TFC = tf(C);
R = feedback(TFC*sys,1);
time = 0:0.01:2;
impulse(R,time);
figure;
rlocus(R);
```

Apéndice E

Arduino® Nano Pines



N. Pin	Nombre	Tipo	Descripción
1-2, 5-16	D0-D13	I/O	Input/Output digital puertos 0 a 13
3, 28	RESET	Input	Reset
4, 29	GND	PWR	GND
17	3V3	Output	Salida +3.3V (De FTDI)
18	AREF	Input	Referencia ADC
19-26	A7-A0	Input	Entrada analógica de 0 a 7
27	+5V	Output or Input	Salida +5V (regulador de la tarjeta) +5V (entrada de fuente externa)
30	VIN	PWR	Fuente de voltaje

Apéndice F

Placa PCB