UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

INTERCONECTIVIDAD LOCAL DE SISTEMA DE POSICIONAMIENTO AUTONOMO PARA UNA COMUNIDAD DE DISPOSITIVOS COMPACTOS DE ARQUITECTURA MODULAR

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELECTRICISTA

SEBASTIAN DANIEL BARCKHAHN FLORES

PROFESOR GUÍA:
EDUARDO VERA S.

MIEMBROS DE LA COMISIÓN:
NESTOR BECERRA YOMA.
JORGE SANDOVAL A.

SANTIAGO DE CHILE
ENERO 2011

RESUMEN DE LA MEMORIA

PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELECTRICISTA
POR: SEBASTIÁN DANIEL BARCKHAHN FLORES
PROF. GUÍA: SR. EDUARDO VERA SOBRINO
FECHA: 11 DE ENERO DE 2011

"Interconectividad local de sistema de posicionamiento autónomo para una comunidad de dispositivos compactos de arquitectura modular"

En el presente trabajo se busca desarrollar y montar una aplicación de posicionamiento autónomo sobre un dispositivo portátil y de arquitectura modular llamado *BUG*. El objetivo es poder llegar a obtener la información de posicionamiento relativo, es decir, el posicionamiento respecto de un punto inicial conocido, haciendo uso de un acelerómetro y una brújula, procesar y desplegar esta información en pantalla y transmitirla de manera inalámbrica a otro *BUG*.

El *BUG* es un microcomputador basado en Linux. Uno de los módulos con que cuenta se llama *BUGsensor,* y tiene una serie de sensores, entre los cuales se encuentran un acelerómetro y una brújula de tres ejes, y es el que fue utilizado para el desarrollo de la aplicación. Al eje Z de la brújula no se tiene acceso debido a un problema en la adquisición del servicio, el cual es entregado por otro chip, lo que limita la aplicación a funcionar solo entre los ángulos 90 y -90 grados de inclinación respecto a la horizontal inicial.

La idea es utilizar el vector N, que representa al Norte magnético, como referencia para obtener el vector G de aceleración gravitatoria. Una vez obtenidos, se leen los datos desde el acelerómetro y substrayendo vectorialmente el vector G a los datos, se obtiene el vector aceleración neta, el que luego se multiplica por el tiempo de muestreo para adquirir la velocidad desde la cual se obtiene finalmente la posición.

A través de diversos métodos de procesamiento de datos se logra mostrar la posición actual respecto a tres ejes fijos: Norte, Este y Arriba.

La aplicación no pudo ser corrida directamente en el *BUG,* por lo que fue necesario crear un simulador que reemplazó la adquisición de los datos. Este simulador fue ejecutado en el emulador del *BUG* con el que cuenta el entorno de trabajo *Dragonfly*. Las pruebas resultaron exitosas, por lo que se logró la validación de la aplicación.

Finalmente se propone como trabajo futuro solucionar el problema de funcionamiento de la aplicación, migrar está al *BUG* 2.0 que cuenta con una mayor capacidad de procesamiento y lograr la transmisión inalámbrica de los datos obtenidos.

Agradecimientos

Mis más sinceros agradecimientos a mi profesor guía, Sr. Eduardo Vera Sobrino, por las oportunidades que me ha abierto mientras he trabajado con él y al buen equipo de trabajo que formamos.

Gracias a mis padres Jorge y Patricia por apoyarme tanto en mi decisión de cambio de carrera, como con los recursos necesarios para costear finalmente ingeniería. También les agradezco su cariño incondicional y aunque se los digo muy poco, los quiero mucho. Agradezco además a mis hermanos menores, Pablo Ignacio y Javiera Paz, quienes solo por existir me ayudan a seguir adelante y les digo que los quiero mucho también.

Quiero agradecer además a mis compañeros y amigos de carrera. Primero le agradezco enormemente a Pamela Castillo cuyo apoyo fue fundamental para aprobar muchos de los ramos cursados y su amistad y cariño es y ha sido una de las más valiosas que he tenido en la vida y espero que seguirá siendo así, ojala durante muchísimo tiempo más. También le agradezco a Pablo Vásquez “Vaskular” con quien compartí muchos ramos, estudios y celebraciones, y su amistad fue muy importante durante toda la carrera y también fuera de ella. Gracias Nicolás López “Manwa” por ser tan optimista, siempre subir mi ánimo y convertirse en uno de mis mejores amigos. También quiero agradecer a Diego Cortes “Mandril”, Daniela Astaburuaga “Burawa” y Cristóbal Armstrong “Ass” a quienes conocí durante Bachillerato y también siguieron ingeniería. Ellos también se convirtieron en mis mejores amigos y son con quienes compartí muchas horas de estudio antes y durante la carrera y sé que seguirán siendo mis grandes amigos de la vida, incluso fuera de la carrera.

Por ultimo le agradezco al resto de mis mejores amigos, que aunque no participaron directamente de mis estudios universitarios, compartimos grandes experiencias durante la vida, con algunos la enseñanza básica y con otros además fuimos Scouts, instancia muy importante en mi vida y creo que también en la de ellos: Eduardo Pardo, Héctor Herrera, Ignacio Catrileo, Cristian Fuentes, Cristopher Orellana, Kreuza Alarcón, Juan Pablo Carrasco, Juan Pablo Uribe, Ivan Acuña “Paloon”, Nicolás Vargas, Daniel Romo, Manuel Salva e Israel Robinson. A todos ustedes les agradezco mucho su amistad.

Índice de Contenidos

Índice de Figuras

Índice de Tablas

Índice de Códigos

1 Introducción

1.1 Motivación

El uso de dispositivos electrónicos portátiles se ha convertido en una tendencia mundial que ha llegado a gran parte de la población ya sea en forma de celulares, PDAs, *Netbooks* u otros. Además, estos se han convertido en asistentes para el trabajo, donde muchas veces desempeñan papeles cruciales. En minería subterránea, por ejemplo, resulta sumamente importante poder estimar la posición de los trabajadores en todo momento y es ahí donde estos dispositivos pueden prestar valiosos servicios.

Por esto que nace la inquietud de crear una aplicación que pueda determinar la posición sin necesidad de elementos externos, mediante la utilización solo de sensores internos, lo cual puede prestar servicios en diversas áreas y en algunos casos extremos incluso puede marcar la diferencia entre la vida y la muerte de un operario.

1.2 Objetivo General

En esta memoria se busca explorar el manejo y procesamiento de la información obtenida desde sensores, utilizando dispositivos electrónicos portátiles que cuentan con una arquitectura modular. Esto último implica la existencia de diferentes módulos con diversas funcionalidades cada uno. Mediante el uso de alguno de estos módulos se espera lograr determinar la posición relativa del dispositivo solo mediante el uso sensores internos, lo que permitirá llegar a reemplazar la necesidad de referencias externas que usualmente se utilizan para este fin.

1.3 Objetivos Específicos

Se busca montar la aplicación de posicionamiento a ciegas, sobre equipos portátiles y modulares, llamados *BUGs*. Los dispositivos *BUG* son pequeños computadores portátiles con procesador ARM, que cuentan con diferentes interfaces de acceso (como USB, Ethernet, puertos seriales, etc.), batería interna, y lo principal es la capacidad de agregar diferentes tipos de módulos con diversas funcionalidades, como GPS, pantalla LCD, sonido y 3G, entre otros, pudiendo utilizar hasta cuatro módulos en

forma simultánea. El desarrollo de las aplicaciones, así como la arquitectura misma de los aparatos es totalmente abierta logrando así la colaboración de una comunidad creciente de usuarios y desarrolladores asociados a una nueva empresa, llamada *Bug Labs Inc*.

El *BUG* deberá programarse para lograr determinar su posición y si es posible transmitir ésta a otro dispositivo *BUG*. Esta información será desplegada en una pantalla LCD. Para esto será necesario contar con los módulos de LCD, el cual servirá de interface para el usuario ya que además es *Touch Screen* y el *BUGsensor,* que es un módulo que cuenta con varios tipos de sensores, como acelerómetro, brújula, luminosidad y movimiento entre otros. Mediante la programación que se realice sobre los *BUGs* estos deberán ser capaces de estimar su posición relativa respecto de un punto inicial conocido.

1.4 Estructura de la memoria

La memoria se divide en 6 capítulos incluido el presente correspondiente a la Introducción. El capítulo 2 prepara al lector con lo necesario para entender esta memoria, se revisa internet, OSGI y transmisiones de datos entre otros temas. El capítulo 3 muestra y explica el código de la implementación desarrollada explicando los métodos y registros de los servicios. El capítulo 4 aborda las pruebas realizadas, para la cual fue necesario crear un simulador que reemplazara la adquisición de datos. El capítulo 5 muestra las conclusiones y plantea el trabajo a futuro. El capítulo 6 muestra la bibliografía usada en esta memoria. Finalmente el capítulo 7 contiene los anexos, código en su mayoría y una tabla de resultados de una de las pruebas.

2 Marco Conceptual

2.1 Internet

La historia de Internet se remonta al temprano desarrollo de las redes de comunicación. La idea era crear una red de computadoras diseñada para permitir la comunicación general entre usuarios de varias computadoras, ya sea para desarrollos tecnológicos, como también para la fusión de la infraestructura de la red ya existente y los sistemas de telecomunicaciones.

Se suele considerar al correo electrónico como una aplicación posterior a la aparición de Internet, aunque realmente, el *e-mail* ya existía antes de Internet y fue una herramienta crucial en su creación. Comenzó en 1965 como una aplicación de ordenadores centrales a tiempo compartido para que múltiples usuarios pudieran comunicarse.

Las más antiguas versiones de internet aparecieron a finales de los años cincuenta. Implementaciones prácticas de estos conceptos empezaron a finales de los ochenta y a lo largo de los noventa. En la década de 1980, tecnologías que reconoceríamos como las bases de la moderna Internet, empezaron a expandirse por todo el mundo. En los noventa se introdujo la *World Wide Web* (WWW), que se hizo muy popular y muchas veces se confunde con el mismo Internet. La *World Wide Web* es un sistema de documentos de hipertexto o hipermedios enlazados y accesibles a través de Internet. Con un navegador web, un usuario visualiza sitios web compuestos de páginas web que pueden contener texto, imágenes, videos u otros contenidos multimedia, y navega a través de ellas usando hiperenlaces.

El punto decisivo para la *World Wide Web* comenzó con la introducción de *Mosaic* en 1993, un navegador web con interfaz gráfica desarrollado por un equipo en el *National Center for Supercomputing Applications* en la Universidad de Illinois en Urbana Champaign EE.UU, liderado por Marc Andreessen. *Mosaic* fue finalmente suplantado en 1994 por *Netscape Navigator* de Andreessen, que reemplazó a *Mosaic* como el navegador web más popular en el mundo. La competencia de *Internet Explorer* y una variedad de otros navegadores, como *Firefox* o *Chrome*, casi lo ha sustituido completamente en la actualidad.

A medida que Internet creció, se crearon los buscadores y los directorios web para localizar las páginas en la red y permitir a las personas encontrar lo que buscaban. El primer buscador web completamente de texto fue *WebCrawler* en 1994. Antes de *WebCrawler*, sólo se podían buscar los títulos de las páginas web.

Con el crecimiento de "la red", la complejidad de la misma creció junto con ella. Para esto se crea OSI, que es un modelo de referencia de Interconexión de Sistemas Abiertos (*Open System Interconnection*) que sirve de modelo de red descriptivo y fue creado por la Organización Internacional para la Estandarización en 1984. Este, es un marco de referencia para la definición de arquitecturas de interconexión de sistemas de comunicaciones.

2.1.1 OSI

El modelo propone siete capas que son:

2.1.1.1 Capa Física (Capa 1)

Es la encargada de las conexiones físicas de la computadora hacia la red, tanto en lo que se refiere al medio físico en si, como a la forma en la que se transmite la información. Define los medios físicos (pares trenzados, RS232, coaxial, etc), características de los materiales, características de las interfaces y el manejo de las señales eléctricas entre otras.

2.1.1.2 Capa de Enlace (Capa 2)

Es la encargada del direccionamiento físico, de la topología de la red, del acceso a la red, de la notificación de errores, de la distribución ordenada de tramas y del control de flujo. Aquí se hace un direccionamiento de los datos en la red y la notificación de errores.

2.1.1.3 Capa de Red (Capa 3)

Es la encargada de hacer que los datos lleguen desde el origen al destino, aún cuando ambos no estén conectados directamente. Los dispositivos que facilitan tal tarea se denominan *routers*. Los *routers* trabajan en esta capa, aunque pueden actuar como *switch* de capa 2 en determinados casos, dependiendo de la función que se le asigne. En este nivel se realiza el direccionamiento lógico y la determinación de la ruta de los datos hasta su receptor final.

2.1.1.4 Capa de Transporte (Capa 4)

Es la encargada de efectuar el transporte de los datos desde la máquina origen a la de destino, independizándolo del tipo de red física que se esté utilizando. El encabezado de la capa 4 se llama Segmento o Datagrama, dependiendo de si corresponde a UDP o TCP, los cuales son los protocolos de esta capa. El primero esta orientado a conexión, o sea que hay un seguimiento de errores y retransmisión en caso necesario, y el segundo no esta orientado a la conexión, con lo cual no se verifica si existen errores ni se retransmite.

2.1.1.5 Capa de Sesión (Capa 5)

Es la encargada de mantener y controlar el enlace establecido entre dos computadores que están transmitiendo datos de cualquier índole. Por esto, el servicio provisto por esta capa, es la capacidad de asegurar que, dada una sesión establecida entre dos máquinas, esta se pueda efectuar para las operaciones definidas de principio a fin, reanudándolas en caso de interrupción. En muchos casos, los servicios de la capa de sesión son parcial o totalmente prescindibles.

2.1.1.6 Capa de Presentación (Capa 6)

Es la encargada de la forma en que se representa la información, de manera que aunque distintos equipos puedan tener diferentes representaciones internas de caracteres, los datos lleguen de manera reconocible. Esta capa es la primera en

trabajar más el contenido de la comunicación que el cómo se establece la misma. Se tratan aspectos como la semántica y la sintaxis de los datos transmitidos, ya que distintas computadoras pueden tener diferentes formas de manejarlas. Esta capa también permite cifrar los datos y comprimirlos, por lo que se podría decir que actúa como “traductor”.

2.1.1.7 Capa de Aplicación (Capa 7)

Es la encargada de ofrecer a las aplicaciones la posibilidad de acceder a los servicios de las demás capas y define los protocolos que utilizan las aplicaciones para intercambiar datos, como correo electrónico (POP y SMTP), gestores de bases de datos y servidor de ficheros (FTP). Existen tantos protocolos como aplicaciones distintas y puesto que continuamente se desarrollan nuevas aplicaciones el número de protocolos crece aceleradamente. Cabe aclarar que el usuario normalmente no interactúa directamente con el nivel de aplicación, ya que este suele interactuar con programas que a su vez, interactúan con el nivel de aplicación pero ocultando la complejidad subyacente.

2.1.2 TCP

Uno de los protocolos más importantes de Internet es el protocolo TCP (*Transmission Control Protocol*) que fue creado entre los años 1973 y 1974 por Vint Cerf y Robert Kahn. Como se mencionó anteriormente, TCP es un protocolo de Capa 4 según el modelo OSI y está orientado a la conexión.

TCP es la capa intermedia entre el protocolo de Internet (IP) y la capa de aplicación. Habitualmente, las aplicaciones necesitan que la comunicación sea confiable y, dado que la capa IP aporta un servicio de datagramas no fiable (sin confirmación), TCP añade las funciones necesarias para prestar un servicio que permita que la comunicación entre dos sistemas se efectúe libre de errores, sin pérdidas y con seguridad.

Los servicios provistos por TCP corren en el anfitrión (*Host*) de cualquiera de los extremos de una conexión, no en la red misma. Por lo tanto, TCP es un protocolo para manejar conexiones de extremo a extremo. Tales conexiones pueden existir a través de

una serie de conexiones punto a punto, por lo que estas conexiones extremo-extremo son llamadas circuitos virtuales. Las características del TCP son:

2.1.3 Orientado a la conexión

Dos computadoras establecen una conexión para intercambiar datos. Los sistemas de los extremos se sincronizan con el otro para manejar el flujo de paquetes y adaptarse a la congestión de la red.

2.1.4 Operación Duplex y Full-Duplex

Una conexión TCP es un par de circuitos virtuales, cada uno en una dirección. La operación *Duplex* es bidireccional, pero no necesariamente en forma simultánea, mientras que la operación *Full Duplex* si se realiza en forma simultánea. Sólo los dos sistemas finales sincronizados pueden usar la conexión.

2.1.5 Error Checking

Utiliza una técnica de c*hecksum*, que es una forma de control de redundancia para proteger la integridad de los datos, verificando que no hayan sido corrompidos. El procedimiento se realiza sumando los bits de los datos enviados, almacenando y enviando luego este valor, el cual al ser recibido, se compara con uno nuevo calculado con los datos que se recibieron y si su *checksum* coincide con el recibido, probablemente los datos no hayan sido corrompidos

2.1.6 Acknowledgements

Es el aviso sobre el recibo de uno o más paquetes. El receptor regresa un reconocimiento (*Acknowledgement*) al transmisor, indicando que recibió los paquetes. Si los paquetes no son notificados, el transmisor puede reenviar los paquetes o terminar la conexión si es que el transmisor cree que el receptor no está más en la conexión.

2.1.7 Control de flujo

Si el transmisor está desbordando el buffer del receptor por transmitir demasiado rápido, el receptor descarta paquetes. Los *acknowledgement* fallidos que llegan al transmisor le alertan para bajar la tasa de transferencia o dejar de transmitir.

2.1.8 Servicio de recuperación de Paquetes

El receptor puede pedir la retransmisión de un paquete. Si el paquete no es notificado como recibido (ACK), el transmisor envía de nuevo el paquete.

Los servicios confiables de entrega de datos son críticos para aplicaciones tales como transferencias de archivos (FTP por ejemplo), servicios de bases de datos, proceso de transacciones y otras aplicaciones de carácter crítico, en las cuales la entrega de cada paquete debe ser garantizada.

Las conexiones TCP se componen de tres etapas: establecimiento de conexión, transferencia de datos y fin de la conexión. Para establecer la conexión se usa un procedimiento llamado negociación en tres pasos (*3-way handshake*). Una negociación en cuatro pasos (*4-way handshake*) es usada para la desconexión. Durante el establecimiento de la conexión, algunos parámetros como el número de secuencia, son configurados para asegurar la entrega ordenada de los datos y la robustez de la comunicación.

TCP usa el concepto de número de puerto para identificar a las aplicaciones emisoras y receptoras. Cada lado de la conexión TCP tiene asociado un número de puerto (de 16 bits sin signo, con lo que existen 65536 puertos posibles) asignado por la aplicación emisora o receptora.

Finalmente, se puede decir que Internet es la ventana que comunica al mundo entre sí, pero no es la única red que existe. De hecho, existen una gran cantidad de tipos de redes que brindan diferentes servicios y la tendencia actual es a integrar éstas. Este proceso es llamado redes convergentes.

2.2 Redes Convergentes

Las redes convergentes, o redes de multiservicio, hacen referencia a la integración de los servicios de voz, dato y video sobre una sola red basada en IP como protocolo de nivel de red.

Tradicionalmente, estos servicios se han ofrecido en forma separada sobre redes especializadas. En la gran mayoría de corporaciones, por ejemplo, la red de voz se basa en uno o varios PBX (*Private Branch eXchange*) conectados a la PSTN (*Public Switched Telephone Network*) externa, mientras que la red de datos se basa en conmutadores y enrutadores IP interconectando redes LAN (*Local Area Network*) y permitiendo el acceso a Internet. Sin embargo, cada vez es mayor la necesidad de una red única en la que tanto la voz, como los datos y el video converjan naturalmente y permitan además, reducir costos de administración, mantenimiento y manejo de la información, así como aumentar la productividad y disminuir los tiempos de atención a los clientes.

En efecto, la aplicación del protocolo IP para la transmisión integrada de voz y datos es un concepto que ha revolucionado a la industria de las telecomunicaciones, elevando la posición de la Internet a un plano de competencia comercial.

Una red de convergencia basada en IP se construye sobre tres elementos claves:

- Tecnologías que permitan ofrecer múltiples servicios sobre una red de datos.
- Una red multipropósito, construida sobre una arquitectura de red funcionalmente distribuida y basada en IP.
- Un sistema abierto de protocolos estándares, maduro e internacionalmente aceptado.

La clave para las redes de convergencia basadas en IP es la división de las principales funciones de red en componentes lógicos que pueden implementarse en equipos de propósito específico. Así se pueden construir soluciones escalables e interoperables para satisfacer las diferentes necesidades de los distintos proveedores de servicios a bajo costo y permitiendo que los mismos servicios se puedan ofrecer

uniformemente a lo largo de toda la red. De esta manera los proveedores pueden acelerar el desarrollo de sus soluciones mediante la adquisición de elementos de red estándar. La competencia entre fabricantes de equipos se promueve a través de estos estándares abiertos, la separación de los elementos de control y de multimedios permite el rápido desarrollo de nuevas aplicaciones.

La infraestructura de la Internet y la madurez y aceptación universal de sus protocolos, hacen del modelo de redes convergentes basadas en IP la forma más adecuada para el rápido desarrollo de estas nuevas redes de convergencia, así como el desarrollo y perfeccionamiento de las distintas técnicas que aseguren los niveles de calidad de servicio.

Aún falta camino que recorrer para alcanzar una red eficaz multiservicios que responda a la multitud de necesidades que se presentan.

2.3 Transmisión Inalámbrica de Datos

La transmisión inalámbrica de datos designa a la forma de transmitir información sin necesidad de una conexión física (cables) si no que por medio de ondas electromagnéticas que se desplazan por el aire y que se reciben y transmiten a través de diferentes tipos de antenas las cuales varían en tamaño y forma según la frecuencia que se utilice.

Dentro de la gama de protocolos de transmisión inalámbrica, solo dos de ellos serán revisados ya que son candidatos de ser usados dentro de esta memoria, ellos son WiFi y ZigBee.

2.3.1 WiFi

Es una marca de la *WiFi Alliance* que es la organización comercial que adopta, prueba y certifica que los equipos cumplan los estándares 802.11 relacionados con redes inalámbricas de área local. El estándar IEEE 802.11 define el uso de las dos capas inferiores de la arquitectura OSI (capas física y de enlace de datos), especificando las normas de funcionamiento dentro de una red inalámbrica local. El estándar original es el 802.11, que permite un ancho de banda de entre 1Mbps a 2Mbps, pero este se ha modificado para optimizar el ancho de banda o para especificar

componentes de mejor manera. Así por ejemplo el estándar 802.11a provee ocho canales sobre la banda de los 5GHz y admite un ancho de banda teórico de 54Mbps. Por otro lado el estándar 802.11b es el más utilizado actualmente y cuenta con tres canales sobre la banda de 2,4GHz y permite un ancho de banda teórico de 11Mbps. Sin duda la principal ventaja que diferencia WiFi de otras tecnologías propietarias, como LTE, UMTS y Wimax, son que las tres tecnologías mencionadas, únicamente están accesibles a los usuarios mediante la suscripción a los servicios de un operador que está autorizado para uso de espectro radioeléctrico, mediante concesiones de ámbito nacional. La principal desventaja frente a las tecnologías propietarias es el alcance limitado que tiene, pero por lo mismo no requiere de autorización para el uso del espectro radioeléctrico.

2.3.2 ZigBee

ZigBee es el nombre de la especificación de un conjunto de protocolos de alto nivel de comunicación inalámbrica para su utilización con radiodifusión digital de bajo consumo, basada en el estándar IEEE 802.15.4 de redes inalámbricas de área personal WPAN (*Wireless Personal Area Network*). Se definen tres tipos distintos de dispositivo ZigBee según su papel dentro de la red.

2.3.2.1 Coordinador ZigBee (ZigBee Coordinator, ZC)

Es el tipo de dispositivo más completo. Debe existir uno por red. Sus funciones son las de encargarse de controlar la red y los caminos que deben seguir los dispositivos para conectarse entre ellos.

2.3.2.2 Router ZigBee (ZigBee Router, ZR)

Interconecta dispositivos separados en la topología de la red, además de ofrecer un nivel de aplicación para la ejecución de código de usuario.

2.3.2.3 Dispositivo final (ZigBee End Device, ZED)

Posee la funcionalidad necesaria para comunicarse con su nodo padre (coordinador o *router*), pero no puede transmitir información destinada a otros dispositivos. De esta forma, este tipo de nodo puede estar dormido la mayor parte del tiempo, aumentando la vida media de sus baterías. Un ZED tiene requerimientos mínimos de memoria y es por tanto significativamente más barato.

Su principal objetivo son las aplicaciones que requieren comunicaciones seguras con baja tasa de envío de datos y maximización de la vida útil de sus baterías. Una red ZigBee puede constar de un máximo de 65535 nodos distribuidos en subredes de 255 nodos. En términos exactos, ZigBee tiene un consumo de 30mA transmitiendo y de 3uA en reposo. Este bajo consumo se debe a que el sistema ZigBee se queda la mayor parte del tiempo "dormido". Tiene una velocidad máxima de 25kbps.

2.4 Sistemas Embebidos

Un sistema embebido, es un sistema de computación diseñado para realizar una o algunas pocas funciones dedicadas, frecuentemente realizadas en tiempo real. Los sistemas embebidos se utilizan para usos muy diferentes a los computadores de propósito general. Como están diseñados para la realización de tareas específicas, los diseñadores pueden optimizarlos reduciendo su tamaño y costo e incrementando su confiabilidad y performance.

Físicamente, los sistemas embebidos van desde dispositivos portátiles como relojes y reproductores de música, a grandes instalaciones estáticas, como controles de luces de tráfico, controladores de fábricas y hasta sistemas de control de plantas nucleares. Su complejidad varía desde integrar un microcontrolador, hasta contar con varias unidades de procesamiento y periféricos en una misma unidad.

En general estos sistemas se escapan del alcance de su propio nombre, ya que muchos tienen elementos que permiten su ampliación y/o programación o son parte de un sistema más grande y de propósito más general. Generalmente las instrucciones de estos (firmware) se encuentran en memorias solo de lectura de tipo Flash.

Las principales ventajas de estos sistemas son el precio y el consumo frente sistema de cómputo de propósito general. Su principal desventaja es que ante el fallo en un elemento, generalmente existe la necesidad de reparar la placa en forma íntegra.

2.5 Dispositivos Electrónicos Portátiles

Los dispositivos Electrónicos Portátiles, son aparatos de pequeño tamaño, que obtienen su fuente de alimentación de baterías y que puede incluir capacidades de procesamiento, de conexión permanente o intermitente a una red, de memoria o bien estar diseñados específicamente para una función en particular.

Los primeros dispositivos portátiles en ser comercializados de manera masiva, fueron los reproductores de audio. Inicialmente se almacenaba la música en *cassetes* (cintas magnéticas), luego de manera digital en discos ópticos (CD) y hoy en día, la música se almacena mediante memorias de estado sólido o discos duros, con lo que se puede agregar y quitar cuantas veces se quiera el audio de dichos aparatos.

Dentro del mundo de los videojuegos, en 1989 hizo su aparición *Game Boy*, que pese a no ser la primera consola portátil, marco una referencia para la época al incorporar cartuchos para los juegos, con los que se pudo jugar diferentes juegos en la misma plataforma. Este, batió récords de venta y con él, se abrieron las puertas para una serie de diferentes consolas portátiles, con la variedad de interfaces y unidades de procesamiento que existen actualmente.

Las cámaras digitales, por su parte, cambiaron la manera en que se toman las fotografías. Reemplazando el rollo y el revelado fotográfico, disminuyó notablemente el costo por fotografía para el usuario final, lo cual brindó finalmente una mayor cantidad de fotografías, las que además pueden ser almacenadas en menos espacio que la fotografía tradicional.

A continuación los *Handhelds* se convirtieron en equipos robustos, principalmente para su utilización en la gestión de situaciones empresariales, como por ejemplo, archivar variedad de tareas, digitalización de notas, gestión de archivos, capturas de firmas, gestión, escaneo de partes de código de barras, etc. Luego aparecieron *Handhelds* que incorporaban telefonía, lo que desencadenó en la integración de todas las aplicaciones de comunicación y procesamiento en un mismo dispositivo, el teléfono móvil.

El teléfono móvil, más que un teléfono, actualmente es un terminal multifuncional, que permite la conexión a través de diferentes redes inalámbricas, brindando servicios de mail, procesamiento de texto, telefonía, GPS, hojas de cálculo, agenda, Internet, mensajería, etc. Aunque la mayoría de las funciones están incorporadas hoy en día a los terminales móviles, aún no se ha reemplazado el poder de procesamiento de los PC's de escritorio, para el cálculo de diferentes tareas que requieran mayor poder de procesamiento. Los computadores de escritorio cuentan además con el teclado como herramienta de interface, que es la mejor opción actual para ingresar texto en cuanto a rapidez (aunque se está intentando la incorporación de teclados proyectados, que se emiten desde un proyector que se encuentra en el teléfono).

Los *Laptop*s simplemente mejoraron el transporte de los PC, ya que no se puede decir que son cien por ciento portátiles, pues para su correcta utilización se requiere estar sentado, a diferencia del resto de los dispositivos móviles.

Dado el variado número de niveles de funcionalidad asociado con dispositivos móviles, en 2005, T38 y DuPont *Global Mobility Innovation Team* propusieron los siguientes estándares para la definición de dispositivos móviles, la mayoría de ellos aplicada a los terminales de telefonía móvil:

2.5.1 Dispositivo Móvil de Datos Limitados (*Limited Data Mobile Device*)

Son dispositivos que tienen una pantalla pequeña, principalmente diseñada para texto con servicios de datos generalmente limitados a SMS y acceso WAP. Un típico ejemplo de este tipo de dispositivos son los teléfonos móviles de primera y segunda generación.

2.5.2 Dispositivo Móvil de Datos Básicos (*Basic Data Mobile Device*)

Son dispositivos que tienen una pantalla de mediano tamaño, (entre 120 x 120 pixeles y 240 x 240 pixeles), menú o navegación basada en íconos por medio de una "rueda" o cursor, y que ofrece acceso a e-mail, lista de direcciones, SMS, y un navegador web básico. Un típico ejemplo de este tipo de dispositivos son las BlackBerry y los teléfonos inteligentes.

2.5.3 Dispositivo Móvil de Datos Mejorados (*Enhanced Data Mobile Device*)

Estos son dispositivos que tienen pantallas de medianas a grandes (por encima de los 240 x 120 pixels), navegación de tipo *stylus*, y que ofrecen las mismas prestaciones que los dispositivos móviles de datos básicos más aplicaciones nativas, como Microsoft Office Mobile (Word, Excel, PowerPoint) y aplicaciones corporativas usuales, en versión móvil, como SAP, portales de intranet, etc. Este tipo de dispositivos incluyen los Sistemas Operativos como Windows Mobile, como en las Pocket PC o Iphone OS de Mac para iPhone y iPod Touch.

2.6 Dispositivos Electrónicos Portátiles: BUG

El *BUG* es un micro-computador de arquitectura modular basado en Linux, al cual se le pueden unir 4 módulos diferentes de manera simultánea, ampliando sus funciones como se guste, mediante componentes actualizables y disponibles para otros usuarios vía web (el usuario A puede usar, incluso, el GPS de B), lo que permite la creación de dispositivos electrónicos a la medida del desarrollador.

La base versión 1.3 cuenta con las siguientes características:

Figura 2.1: Base BUG.

- *Microprocesador ARM 11 de 532Mhz.*
- *128 MB SDRAM y 32 MB de almacenamiento en memoria flash*
- *Interfase de tarjeta MicroSD (hasta 16GB)*

- *802.11b/g WLAN integrado*
- *Bluetooth 2.0 + EDR*
- *4 interfases BUGmodule*
- *Mini-LCD con joystick y botones configurables*
- *Hardware acelerador gráfico y codificación/decodificación MPEG4*
- *Poky Linux 1.4.x (kernel 2.6.29)*
- *Batería interna de Li-ION de carga rápida (1100 mAh)*
- *USB 2.0 OTG*
- *4 links UART seriales*
- *Interfaz SPI de 4 canales*
- *10/100 Ethernet MAC*
- *Dimensiones: 5.10" x 2.55" x 0.765"*

2.6.1 Módulos

Los módulos que se venden en la página principal de *Bug Labs Inc.* hasta hoy, son*: BUGbee, BUGmotion, BUGvonHippel, BUGlocate, BUGview, y BUGsound*.

2.6.1.1 BUGbee

Es un módulo de radio de bajo consumo basado en el protocolo 802.15.4 ZigBee, que resulta ideal para realizar redes caseras de comunicación, entre *BUG*s, así como automatización de hogares o aplicaciones en las que trabajen varios *BUG*. Además es compatible aplicaciones Xbee ZB.

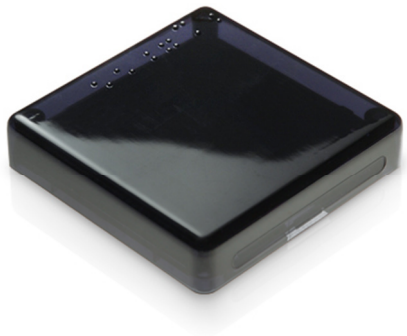

Figura 2.2: BUGbee.

2.6.1.2 BUGmotion

Es un módulo que cuenta con dos componentes: un detector de movimiento y un acelerómetro. El detector de movimiento tiene un rango de 2m y detecta movimientos de hasta 30 centímetros. El acelerómetro tiene una sensibilidad que varia entre 2.5g y 10g. Es ideal para creación de aplicaciones en las que se incluya movimiento de algún tipo.

Figura 2.3: BUGmotion.

2.6.1.3 BUGvonHippel

Es un módulo en el que se pueden agregar sensores, con conexiones directa mediante cables de tamaño estándar, o lo que se quiera unir mediante USB. Cuenta además con puerto serial y SPI. Es una interfaz ideal para cualquier tipo de electrónica que se quiera ampliar, lo que extiende la funcionalidad del *BUG* a cualquier nivel de electrónica pudiendo operar por ejemplo, como controlador de otro dispositivo.

Figura 2.4: BUGvonHippel.

2.6.1.4 BUGlocate

Es un módulo que provee conexión GPS, lo cual abre la posibilidad de desarrollo de aplicaciones de ubicación. Tiene una antena pasiva interna e incluye un conector para una antena activa externa. Utiliza implementación chipset SiRF.

Figura 2.5: BUGlocate.

2.6.1.5 BUGview

Es una pantalla LCD de 2,46 pulgadas, resolución de 320x240 y que puede ser usado como *touch-sensitive-screen*. Este módulo puede ser utilizado como *display* o como dispositivo de entrada de datos, lo que extiende las capacidades de los *BUG* al despliegue de información visual y entrada táctil.

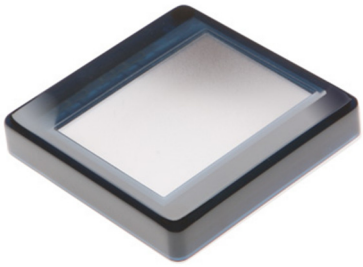

Figura 2.6: BUGview.

2.6.1.6 BUGsound

Es un módulo de audio, provisto con un parlante de 20 milímetros y un micrófono omnidireccional, con *hardware* de codecs stereo y 4 *plugs* stereo, para 3 entradas simultaneas, además de salidas, audífonos y micrófonos. Puede ser usado como reproductor de música, procesador de audio, entre otras funciones.

Figura 2.7: BUGsound.

2.6.1.7 BUGsensor

Además de los módulos nombrados anteriormente, en esta memoria se trabajara con un módulo que no se encuentra disponible comercialmente. Este es llamado *BUGsensor* y cuenta con acelerómetro de tres ejes, brújula de tres ejes, la cual se implementa con un circuito integrado que entrega los ejes X e Y y a través de otro chip se obtiene el eje Z, sensor de proximidad, sensor de luminosidad y sensor de humedad. Todos estos sensores son análogos y mediante conversores análogo-digital las señales son obtenidas en forma digital. Mediante el uso de la brújula y el acelerómetro se realizará la implementación de posicionamiento que trata esta memoria.

Como soluciones empresariales de innovación, los *BUG* se presentan como opciones ideales, ya que reduce proceso de creación de un nuevo dispositivo electrónico, pasando desde el prototipo desarrollado en el *BUG* al modelo Beta de forma inmediata, evitando el desarrollo del prototipo, el piloto y la versión Alfa del nuevo producto.

En educación, nuevamente los *BUGs* resultan soluciones ideales para acercar la electrónica a los estudiantes y brinda la posibilidad de realizar una infinidad de pruebas para el aprendizaje y familiarización según desee el diseñador/educador en cuestión. Además mediante la colaboración de la comunidad, cada vez existen más aplicaciones disponibles en línea.

Para los desarrolladores independientes, los *BUGs* abren las puertas para que mediante buenas ideas, se pueda lograr la materialización de un nuevo dispositivo, lo que simplifica el proceso de creación mismo de un producto electrónico nuevo, donde tradicionalmente se requiere de gran capital humano y material, pasando a estar disponible prácticamente a cualquier usuario que lo desee.

Las aplicaciones van desde sensores inteligentes, monitoreo de salud remoto, cámaras de seguridad inteligentes (que solo transmiten cuando detectan movimiento), hasta juegos interactivos, que son posibles de montar sobre los *BUGs*, lo que hace de estos dispositivos una herramienta sumamente versátil para el desarrollo de nuevos dispositivos electrónicos o simplemente para la satisfacción de un usuario que quiera tener un *gadget* a su medida.

Además los *BUGs* pueden ser programados en lenguajes de alto nivel, como Java, Python y Ruby, así como en C que es de más bajo nivel.

2.7 OSGI

Como el uso de dispositivos móviles y embebidos ha pasado a ser una tendencia mundial, el manejo y configuración del software se ha convertido en un asunto critico. OSGI es un estándar para el manejo del ciclo de vida de los componentes de software de Java. Está orientado a dar servicios a arquitecturas donde las unidades funcionales puedan ser removidas y funcionar de manera independiente entre si. Es ideal para el ambiente de hoy donde la movilidad requiere de una constante adaptación en la adición y reemplazo de los diferentes módulos tanto de *software* como de *hardware*.

Hasta hace poco tiempo atrás las tareas requeridas al realizar un cambio en el *hardware*, implicaban necesariamente el rediseño del *software* de forma *ad-hoc* y de lo cual se encargan los propietarios de los sistemas, trabajando cada uno por su parte. OSGI responde a la demanda de una plataforma estandarizada de manejo de *software* con módulos dinámicos.

Cualquier marco de trabajo que implemente el estándar OSGI provee un ambiente para la modularización de las aplicaciones en pequeños *bundles*. Cada *bundle* es una colección de clases, archivos jar y archivos de configuración que declaran explícitamente sus dependencias externas si es que existen. Un archivo *jar* contiene varios archivos dentro de uno que generalmente se usa para la distribución de librerías o aplicaciones en forma de clases, *metadata* y recursos como fotografías, texto, etc.

Conceptualmente OSGI está dividido en las siguientes áreas (Tabla 2.7.1):

Bundles	Un *bundle* es un grupo de clases Java y recursos adicionales con un detallado archivo *MANIFIEST.MF* en todos sus contenidos así como también servicios adicionales necesarios para incluir grupos de clases de Java de comportamiento más sofisticado, extiendo todo el conjunto para ser percibido como una sola unidad.
Services	La capa de servicio conecta los *bundles* de manera dinámica ofreciendo un modelo público de búsqueda para los antiguos objetos de Java.
Services Registry	Es la API para el manejo de los servicios. Una API (*Application Programming Interface)* es una interfaz implementada por un programa que permite interactuar con otro software, lo que facilita la interacción entre diferente software. Las tres APIs que brinda esta capa son: - ***Service Registration:*** Es la API que es devuelta cuando el método *BundleContext.registerService* ha sido invocado exitosamente. Este objeto es de uso privado para el registro de un *bundle* y no se comparte con otros *bundles*. - ***ServiceTracker:*** Esta API se construye con un criterio de búsqueda y un objeto llamado *ServiceTrackerCustomizer.* El objeto *ServiceTracker* puede usar al objeto *ServiceTrackerCustomizer* para modificar el servicio que se está buscando y ser abierto para comenzar a buscar todos los servicios que concuerden con el criterio de búsqueda. - ***Service Reference:*** Esta API puede ser compartida entre diferentes *bundles* y ser usada para examinar las propiedades del servicio y obtener el objeto *service.* Cada servicio registrado en OSGI tiene un único objeto *ServiceRegistration* y puede tener

	múltiples y/o distintos objetos *ServiceReference* referidos a él.
Life Cycle	Es la API encargada del manejo del "ciclo de vida" de los *bundles.* Esta capa añade *bundles* que pueden ser manejados dinámicamente. Esta parte dinámica introducida no es normalmente parte de la aplicación en sí. Así, los estados que puede tener un *bundle* pueden ser: - ***Installed:*** El *bundle* ha sido exitosamente instalado. - ***Resolved:*** Todas las clases de Java que necesita el *bundle* se encuentran disponibles. Este estado indica que el *bundle* o está listo para ser inicializado o se ha detenido. - ***Starting:*** El *bundle* ha sido inicializado, el método *BundleActivator.start* será llamado, pero no retornado aun. Cuando el *bundle* tiene una política de activación, el *bundle* permanecerá en estado *STARTING* hasta que el *bundle* sea activado de acuerdo a su propia política de activación. - ***Active:*** El *bundle* ha sido activado exitosamente y está ejecutándose. Su método *Bundle Activator* ha sido llamado y retornado. - ***Stopping:*** El *bundle* ha comenzado a detenerse. El método *BundleActivator.stop* ha sido llamado aún no ha sido retornado. - ***Uninstalled:*** El *bundle* ha sido desinstalado. No puede ser movido a otro estado.
Modules	Esta capa define la encapsulación y declaración de las dependencias, es decir como un *bundle* puede importar y exportar código.
Security	Esta capa maneja los aspectos de seguridad limitando la funcionalidad del *bundle* a capacidades predefinidas.

Execution Environment	Define que métodos y clases estarán disponibles para una plataforma específica. No existe una lista fija de los ambientes de ejecución pues está sujeta a los cambios realizados por la *Java Community Process,* la que crea nuevas versiones y ediciones de Java.

Tabla 2.1: Capaz de arquitectura OSGI.

La figura 2.8 muestra un esquema de la arquitectura OSGI.

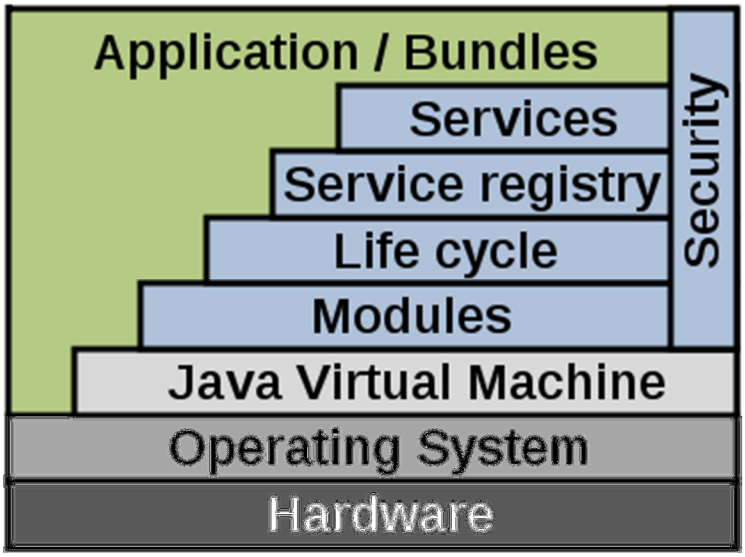

Figura 2.8: Arquitectura OSGI.

2.8 Eclipse y Dragonfly

Eclipse SDK (*Software Development Kit)* es un entorno de desarrollo de software multiplataforma que incluye un entorno de desarrollo integrado (*Integrated Development Environment,* IDE) y un sistema de *plug-in* extensible. Un *plug-in* es un conjunto de componentes de *software* que agrega capacidades específicas a una aplicación de *software* de mayor tamaño. Eclipse generalmente se programa en Java y puede ser utilizado para el desarrollo de aplicaciones tanto en Java como en otros lenguajes (Ada, C, C++, COBOL, Perl, PHP, Python, Ruby, Scala y Scheme) por medio de la adición de diferentes *plug-ins*.

Eclipse utiliza los *plug-ins* para proveer la funcionalidad sobre el sistema de ejecución (*runtime system)*, en contraste con otras aplicaciones donde la funcionalidad es típicamente código duro. Con excepción de una pequeña parte del Kernel que corre en tiempo real, todo en Eclipse es un *plug*-in. Esto implica que cualquier *plug-in* desarrollado, se integra a Eclipse de la misma forma que el resto de los *plug-in*, por lo que todas sus características son tratadas de la misma forma.

Eclipse SDK incluye *Java Development Tools (JDT),* ofrece un IDE que contiene un compilador de Java incremental y un completo modelo de los archivos fuente de Java (Java *Source Files)* lo que permite técnicas avanzadas de *refactoring* y análisis de código. *Code Refactoring* es el proceso de cambio del código fuente de un programa sin alterar el comportamiento funcional externo, con el fin de mejorar algunos atributos o funcionales del programa.

Eclipse está basado en Equinox que es un aplicación compatible con OSGI y es *software* abierto, por lo que no requiere de licencia para su uso.

Dragonfly es un *plug-in* construido sobre Eclipse SDK que fue desarrollado por *Bug Labs Inc.* y que permite manejar de manera gráfica las aplicaciones diseñadas para el *BUG*, es decir, cargar, borrar, copiar, crear, modificar, subir y bajar aplicaciones, desde y hacia el *BUG*. Además permite descargar y correr aplicaciones creadas por otros desarrolladores mediante una conexión directa entre Eclipse y la comunidad *BUGnet*. Todo lo anterior en conjunto con la utilización de las herramientas de desarrollo propias de Eclipse.

2.9 Sensores

Básicamente, lo que se pretende lograr es crear un dispositivo llamado *Inertial Measurement Unit* (IMU) los cuales utilizan tradicionalmente combinaciones de acelerómetros y giroscopios. La diferencia es que aquí se pretende establecer la posición a partir de un acelerómetro y una brújula.

2.9.1 Acelerómetro

Para entender el funcionamiento del acelerómetro, es conveniente imaginar a este como una caja en forma de cubo con una bola dentro de ella (figura 2.9).

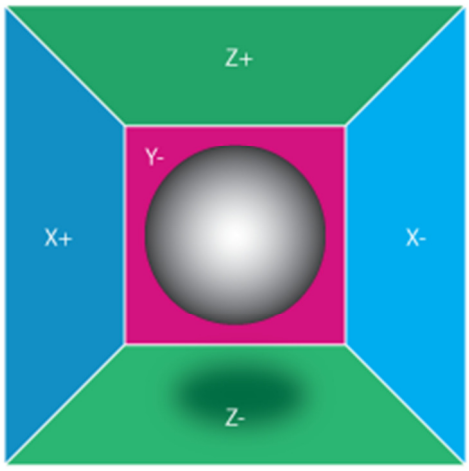

Figura 2.9: Imagen conceptual del acelerómetro.

Así, al someter al sensor bajo una aceleración de 1g en dirección del eje +X, la bola se desplazara en dirección contraria ejerciendo presión contra la pared y generando una señal proporcional a la presión ejercida en ella, como muestra la figura 2.10. Siempre se obtendrá la aceleración resultante a la que está sometido el sensor, y dado que en la tierra existe un campo gravitatorio, este será constantemente medido.

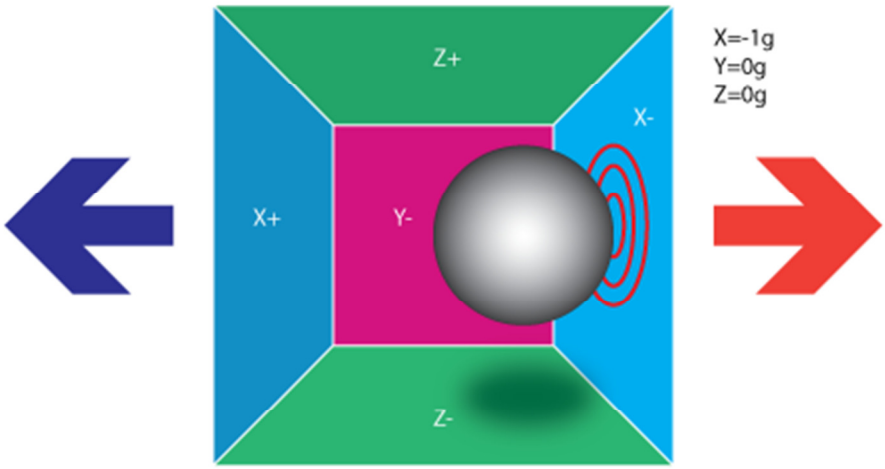

Figura 2.10: Acelerómetro bajo aceleración de 1g.

Si la aceleración no se encuentra solo en un eje del sensor, se obtendrán lecturas en los ejes a los que afecte esta, como se muestra en la figura 2.11, donde bajo una aceleración de 1g, el sensor se encuentra inclinado en cuarenta y cinco

grados, por lo que se obtiene en cada eje una lectura equivalente al coseno de 45 grados por g (0.71g).

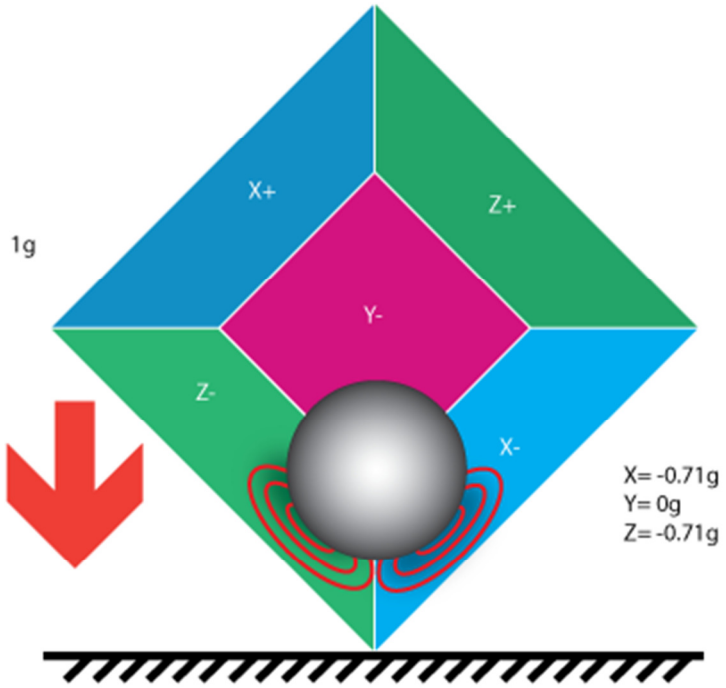

Figura 2.11: Acelerómetro bajo aceleración de 1g.

2.9.2 Brújula

La brújula se comporta de manera muy similar al acelerómetro, solo que en vez de ser sensible a la aceleración responde a los campos magnéticos.

De esta forma, como en la tierra existe un campo magnético que está aproximadamente alineado con el eje de giro de ella (aunque no es exactamente igual) las lecturas de cada eje de la brújula, mostrarán, al igual que con el acelerómetro, hacia donde está el norte magnético terrestre (o el campo magnético resultante al que esté sometido el dispositivo).

Mezclando la información de ambos sensores se espera obtener el vector de aceleración instantánea, desde el cual se pueda inferir el desplazamiento realizado por el dispositivo.

3 Implementación

3.1 Activator

Toda aplicación creada dentro de Dragonfly (el *kit* de desarrollo del *BUG*) debe incluir una clase llamada *Activator*. Cuando el entorno de trabajo OSGI inicia, el método *Start* es llamado y al ser cerrado OSGI, se llama al método *Stop*. Esto es equivalente a los métodos "*public static void main(Strings[] args)"* de la mayoría de los programas desarrollados en Java. Cualquier programa o método llamado desde acá, correrá durante toda la vida del *bundle.*

Esta clase es por lo general común para la mayoría de las aplicaciones. Se comienza por importar las distintas interfaces, métodos y objetos que serán utilizados.

Primero, se importa la interfaz *BundleActivator* que será implementada en esta clase la cual será llamada cuando un *bundle* sea iniciado o detenido. Si el método *BundleActivator.start* se ejecuta correctamente, se garantiza que el método *BundleActivator.stop* de la instancia será llamado cuando el *bundle* se detenga.

Luego, se importa la interfaz *BundleContext* que es utilizada para garantizar el acceso a otros métodos para que el *bundle* pueda interactuar con el entorno de trabajo (*Framework).* Este permite que un *bundle* sea capaz de:

- Subscribirse a los eventos que son publicados por el entorno de trabajo.
- Registrar objetos de servicio (*service objects)* mediante el registro de servicios (*service registry)* del entorno de trabajo.
- Obtener referencias de servicios (*ServiceReference*) desde el registro de servicios del entorno de trabajo.
- Instalar nuevos *bundles* en el entorno de trabajo.
- Obtener el objeto *bundle* requerido para un *bundle*.

En este caso la interfaz será usada para generar un objeto filtro, el que a su vez será utilizado para hacer coincidir un objeto de referencia de servicios (*ServiceReference)*.

A continuación se importa la interfaz *Filter* la cual a diferencia de las anteriores puede ser usada varias veces para determinar si el argumento buscado calza con el argumento recibido en forma de *String* que se usa para crear el filtro. Este puede ser

creado llamando al método de la interfaz anterior *BundleContext.createFilter(String)*. Este será usado básicamente para registrar los servicios que serán utilizados para cada módulo.

El objeto *ServiceTracker* será utilizado ya que simplifica el uso de los servicios del registro de servicios (*service registry*) del entorno de trabajo. Este objeto es construido con un criterio de búsqueda y con el objeto *ServiceTrackerCustomizer.* El último es usado para personalizar los objetos de servicio que son buscados. Es iniciado para comenzar a buscar todos los servicios en el registro de servicios (del *Framework*) que coincidan con el criterio de búsqueda y maneja correctamente todos los detalles que tienen que ver con escuchar los eventos de los servicios, así como obtener y liberar los servicios solicitados.

También es necesario importar otra clase que deberá ser creada, llamada *ServiceTrackerCustom.* Aquí es donde se especificaran los servicios que serán utilizados por la aplicación misma y la cual será llamada por la clase *Activator.*

Finalmente el *ServiceFilterGenerator* es el encargado de la generación del filtro en sí. Si no se deseara utilizar un filtro, se pueden especificar manualmente los servicios que serán utilizados.

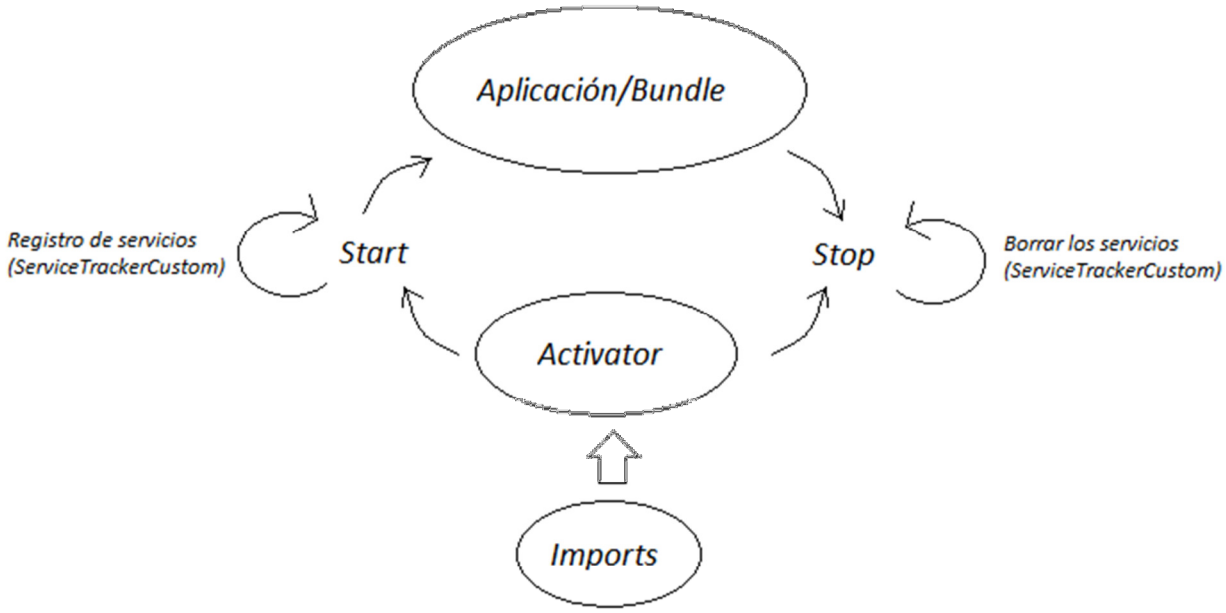

Figura 3.1: Esquema de funcionamiento de la clase Activator.

En resumen, se crea el método *start* (implementación del *BundleActivator*) donde se registran los servicios, usando la clase *ServiceTrackerCustom,* la que se detallará a

continuación y el método *stop* (implementación del *BundleActivator*) simplemente cierra los objetos *ServiceTracker* y *ServiceTrackerCustom*.

El código para esta clase puede ser visto en el Anexo A.1

3.2 ServiceTrackerCustom

Esta clase se encarga de registrar los servicios que utilizara la aplicación. Esta clase es hija de la clase *AbstractServiceTracker* y hereda los siguientes métodos:

- *canStart*: Método que determina si la aplicación puede ser iniciada.
- *doStart*: Método que es invocado cuando se han registrado todos los servicios.
- *doStop*: Método que es invocado cuando todos los servicios han sido removidos del registro.
- *getService*: Método que ayuda a obtener un servicio del mismo tipo que la clase. Retorna una referencia a la instancia del servicio disponible.

Los métodos anteriores son los que serán usados en esta clase[1].

Se deben importar además los servicios que serán utilizados en la aplicación misma, en este caso se utilizaran:

- *IModuleDisplay*: Este servicio proporciona el soporte de los *widgets*[2] de la interfaz gráfica usuario (*Graphic User Interface,* GUI). El principal método de este servicio es *getFrame()* el cual retorna el marco base (*base frame*) usado para crear los controles en la pantalla, que será usado en la aplicación misma.
- *ISensorAccelerometer*: Este servicio proporciona todos los métodos relacionados con el acelerómetro que consistan en la obtención de la lectura de cada eje.
- *ISensorCompass*: Este servicio proporciona todos los métodos relacionados con la brújula que consisten en obtener la lectura de cada eje.

[1] Si se quiere una lista de todos los métodos de la clase *AbstractServiceTracker* revisar http://bugcommunity.com/development/javadoc/r1.4/bug/com/buglabs/application/AbstractServiceTracker.html

[2] *Widget* es un elemento GUI que despliega información que puede ser cambiada por el usuario, como una ventana o un sector donde ingresar texto (*text box*).

- *ISensorModuleControl*: Este servicio proporciona todos los métodos relacionados con el manejo del módulo en sí, que incluyen manejo de los leds y revisar si los sensores están presentes.

Además de lo anterior es necesario importar también la clase que contiene la aplicación misma, ya que será usada para el registro de sus servicios. La figura 3.2 muestra un esquema del funcionamiento de esta clase.

La clase comienza heredando el método *canStart()* de la clase padre *AbstractServiceTracker*. Si este método retorna el *boolean true* se llama al método *doStart()* desde donde se obtienen los servicios usados por la aplicación, además de crear el objeto de la aplicación.

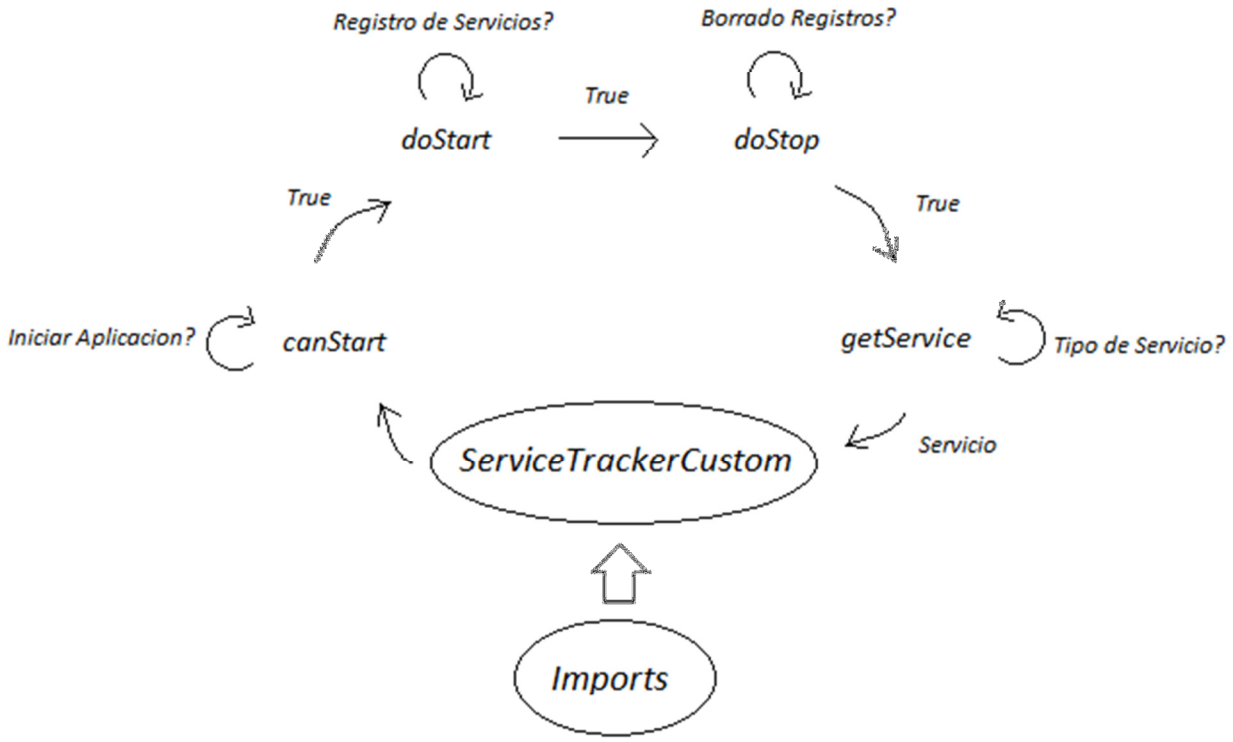

Figura 3.2: Esquema de funcionamiento de la clase ServiceTrackerCustom.

Luego se le agrega al método *stop()* que arroje un mensaje al ser llamado, además de terminar el GUI que utiliza la aplicación. Este método es llamado cuando existe un servicio del que depende la aplicación que aún no ha sido registrado.

Finalmente el método *initServices()* se encarga de agregar los servicios que serán utilizados en la aplicación, los que se detallarán posteriormente.

3.3 PositionApp

En esta clase se creara la aplicación misma. El objetivo de la aplicación es obtener los datos desde el *BUGSensor* procesarlos y entregar la posición relativa desde un punto inicial al usuario a través de una Interfaz de Usuario. La figura 3.3 muestra un esquema del funcionamiento de la aplicación.

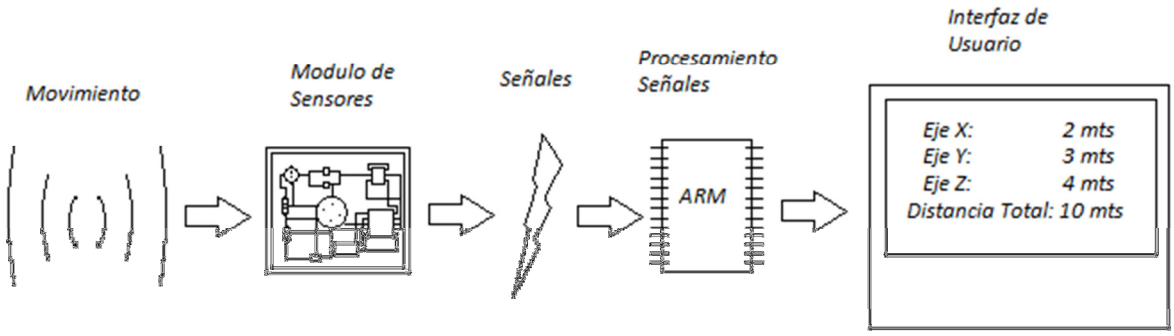

Figura 3.3: Esquema de la Aplicación.

Nuevamente se requiere comenzar importando todo lo necesario para que funcione la aplicación que será desarrollada. Para esto inicialmente se requerirán de los siguientes servicios (el servicio *IModuleDisplay* ya se explicó, por lo que no será comentado ahora)

3.3.1 Imports

- *java.awt.event.*ActionListener: Interfaz encargada de recibir los eventos de acción (*action event*). Una vez que un evento ocurre, se invoca al objeto *actionPerformed* y con él se maneja la acción a realizar. Este será usado cuando se presiona el botón *Start* de la aplicación.
- *java.lang.Math:* Clase que contiene métodos para realizar operaciones numéricas básicas, como potencias, raíces y funciones trigonométricas. Esta clase prestará las operaciones de raíces y funciones trigonométricas que serán usadas en la aplicación.

- *java.lang.System*: Clase que contiene diferentes campos y métodos, como entrada estándar, salida estándar y salidas de errores entre otras. De esta clase se utilizará el método *currentTimeMillis()* el cual retorna el tiempo transcurrido desde que se inició la aplicación.
- *ISensorAccelerometer*: Mediante este servicio se crea un objeto desde el cual se obtienen las lecturas de la aceleración recibida para cada uno de los tres ejes. Los métodos *readX(), readY()* y *readZ()* entregan las lecturas de las aceleraciones para los ejes X, Y y Z respectivamente. Estos ejes pueden ser vistos sobre la placa del módulo mismo para saber hacia dónde apuntan. Los datos son entregados en unidades de g (9,8m/s^2), o sea una lectura de 1.5 en alguna dirección significa una aceleración de 14,7m/s^2 (9,8m/s^2 x 1.5). El eje Z tiene una desviación (*offset*) de 0.2g respecto de la aceleración real, por lo que debe ser sumada para su corrección.
- *ISensorCompass*: Mediante este servicio se crea un objeto desde el cual se obtienen las lecturas del campo magnético recibido para cada uno de los tres ejes. Los métodos *getHx(), getHy()* y *getHz()* entregan las lecturas para los ejes X, Y y Z respectivamente. Además el método *getHeading()* entrega el ángulo de separación en grados entre el eje +X y el campo magnético resultante. Este ángulo solo tiene sentido cuando el sensor (o dispositivo) se encuentra en forma paralela al plano que forma el campo magnético resultante que se está midiendo. Este sensor se compone de dos circuitos integrados, uno encargado de las lecturas de los ejes X e Y y el otro encargado de la lectura del eje Z. Este último no fue correctamente implementado y arroja datos erróneos que parecen aleatorios por lo que inicialmente no será utilizado.
- *ISensorModule:* Este servicio crea un objeto desde el cual se pueden encender y apagar los *LEDs* con que cuenta el módulo (por ejemplo, *LEDGreenON()*) y verificar si se encuentran presentes los sensores, mediante los siguientes métodos asociados:

    1. *isADCPresent()*: Función que retorna un *boolean true* si se encuentra presente el conversor análogo digital

2. *isCompassPresent()*:Función que retorna un *boolean true* si se encuentra presente la brújula.
3. *isProximityPresent()*:Función que retorna un *boolean true* si se encuentra presente el sensor de proximidad.
4. *isTemperaturePresent()*:Función que retorna un *boolean true* si se encuentra presente el sensor de temperatura.
5. *isHumidityPresent()*:Función que retorna un *boolean true* si se encuentra presente el sensor de humedad.
6. *isSoundPressurePresent()*:Función que retorna un *boolean true* si se encuentra presente el sensor de presión de sonido.
7. *isMotionPresent()*:Función que retorna un *bolean true* si se encuentra presente el sensor de movimiento.
8. *isAccelerometerPresent()*:Función que retorna un *boolean true* si se encuentra presente el acelerómetro.
9. *isLightPresent()*:Función que retorna un *boolean true* si se encuentra presente el sensor de luminosidad.

- *java.io.File*: Clase principal para el manejo de archivos y directorios. Ambos son representados por objetos de archivos (*File Objects)*. Este es requerido ya que será necesario leer desde un registro del módulo de sensores, el que será explicado más adelante.
- *java.io.FileInputStream*: Clase que obtiene una entrada en *bytes* desde un archivo. Que archivos están disponibles depende del ambiente en que se trabaje. Será utilizado al igual que el anterior para la lectura del registro del módulo de sensores.
- *java.io.IOException*: Clase necesaria que arroja excepciones de entrada/salida (*I/O exceptions*) en caso de ocurrir algún error o interrupción en la lectura o escritura de datos. También será utilizado en la lectura de los registros del módulo de sensores.
- *java.util.Properties*: Clase que representa un conjunto de propiedades que pueden ser guardadas o cargadas desde un *stream*[3]. Son valores de

[3] Un *stream* es un flujo de datos de cualquier tipo.

configuración manejados como pares de llaves y valores. La llave se usa para identificar y así obtener el valor asociado a está. En este caso nuevamente será utilizado al leer los registros del módulo de sensores.

- *java.awt.Frame*: Clase que despliega una ventana de alto nivel con un título y un borde. Es necesaria para la representación gráfica de cualquier tipo de información.
- *java.awt.Button*: Clase que crea un botón con etiqueta. Se puede realizar una acción cuando el botón es presionado.
- *java.awt.GridLayout*: Clase encargada del manejo de componentes gráficos de forma rectangular. Se pueden agregar componentes (botones y *labels* entre otros), que son ordenados de izquierda a derecha y de arriba hacia abajo según el orden de ingreso.
- *java.awt.Label*: Objeto al que se le puede introducir texto dentro del contenedor el cual no puede ser alterado.
- *java.awt.Panel*: Clase que provee el espacio en donde una aplicación puede insertar cualquier otro componente, incluido otros paneles.
- *java.awt.event.ActionEvent*: Clase que genera un evento que indica que una acción definida por algún componente ha ocurrido (como un botón al ser presionado).
- *java.awt.BorderLayout*: Clase que genera un contenedor, el cual maneja los tamaños de los componentes que son agregados a él para calzar en cinco regiones: Norte, sur, este, oeste y centro. Cada región puede tener solo un componente, el que puede ser un panel que a su vez incluye más componentes.
- *java.awt.Color*: Clase usada para encapsular los colores. Cada color contiene un valor implícito alfa de 1.0 o uno explicito definido en el constructor. Esta clase pintara la interface que será mostrada en pantalla.
- *java.text.DecimalFormat*: Subclase de *NumberFormat* que da formato a los números decimales. Incluye diversos métodos para adaptarse a los distintos tipos de representaciones decimales (como la Occidental, Arabica e India). En este caso será usada para entregar una cierta cantidad de decimales.

Una vez importados todos los elementos necesarios procedemos con la creación de la clase. El código de todo lo que se importó se encuentra en el Anexo A.3.

3.3.2 Constructor

Se crea el constructor de la clase (Anexo A.4). Este incluye los objetos asociados a la pantalla, acelerómetro, brújula, control del módulo de sensores y el *BundleContext* todos los cuales son necesarios de especificar para su registro en el OSGI.

3.3.3 Interfaz de Usuario (*User Interface, UI)*

createUI() creará la interfaz de usuario (*User Interface*) donde se realizará toda la ejecución y está definida a continuación del constructor. Primero, se asignan los objetos necesarios para el despliegue gráfico de los elementos hacia el *frame*, al cual se le entrega el manejo de la pantalla a través del objeto *display* (servicio del *IModuleDisplay*), definiendo su color de fondo y título, en este caso "PositionApp" y blanco respectivamente. Luego se crea el único botón que será utilizado en la aplicación, *Start,* al cual es necesario agregar el *ActionListener* para poder ser escuchado al momento de ser presionado.

A continuación se inicializan todos los *labels* que serán usados. Se crean ocho en total, los que son separados en grupos de cuatro. Se crea un panel para cada grupo de *labels* (dos en total) y uno para el botón. A cada panel se le aplica el método *setLayout(new GridLayout(4,1))* el cual divide al panel en columnas y filas, en este caso cuatro columnas y una fila para agregar los *labels* y un panel de uno por uno para agregar el botón *Start*. Finalmente, se agrega cada panel mediante el método *BorderLayout()* al *frame*. Un grupo de *labels* irá al centro, *BorderLayout.CENTER,* un grupo irá a la izquierda, *BorderLayout.WEST*, y el botón *Start* ira abajo, *BorderLayout.SOUTH*. Para que sea desplegada toda la parte gráfica es necesario agregar *frame.show()* al final de *createUI()*. También es necesario crear un método que destruya la parte gráfica al terminar la aplicación, la que será usada por la clase *ServiceTrackerCustom()* en el método *Stop().* Por esto se creó el método *destroyUI()* donde simplemente mediante *dispose()* se destruye al *frame.*

El código para la interfaz de usuario se puede encontrar en el Anexo A.5.

3.3.4 Métodos

La aplicación utiliza diversos métodos para la corrección de los datos y el cálculo mismo de la posición. Estos son presentados a continuación.

- *loadCalibrationData(int axe)*: Método que es utilizado para obtener los valores de desfase (*offset*) de la brújula una vez que se ha realizado la calibración de esta. Estos valores deberán de ser restados a los datos que se están leyendo desde el sensor, para obtener una lectura coherente. Simplemente se realiza la lectura de un archivo llamado *COMPASS_CALIBRATION_FILE* desde el cual se obtienen los datos buscados. El método primero verifica si existen los datos en el archivo, en caso negativo retorna un cero. Luego estos datos son leídos, convertidos a *double* y finalmente retornados. Se debe ingresar un número *int* al llamar al método, según el eje que se desee obtener, cero para el eje X y uno para el eje Y. Dado que el eje Z no se encuentra bien implementado, no se tendrá esta información dentro del archivo. El código es mostrado en el Anexo A.6.
- *GetCompassAxes()*: Método que realiza las lecturas de los tres ejes de la brújula, más una cuarta lectura que indica el ángulo en grados de separación entre el eje X y el Norte magnético. Retorna un arreglo o vector de cuatro elementos con estos datos. Utiliza el método creado anteriormente para la corrección de los datos, restándoles a los valores leídos los desfases obtenidos. Las lecturas mismas de los ejes se realizan simplemente aplicándoles el método *getHx()* (para el eje X) al objeto *ISensorCompass* y para la lectura del ángulo se aplica el método *getHeading()* al mismo objeto. El código es mostrado en el Anexo A.7.
- *GetAccelAxes()* Método equivalente al anterior, pero para el acelerómetro. De la misma forma, la lectura de los ejes se realiza aplicando el método *readX()* (para el eje X) al objeto *ISensorAccelerometer* y retorna un arreglo de *double* de tres elementos con estos datos. Al eje z se le debe sumar 0,2 ya que tiene ese desfase implícito. Cada valor luego es multiplicado por 9,8 para pasar el valor a m/s^2. Cabe destacar que para la lectura de los ejes, a diferencia de la brújula, el método debe estar dentro de un *try* en caso de existir un error. El código se muestra en el Anexo A.8.

- *GetCompassMaxMin()*: Método que obtiene las máximas y mínimas lecturas de la brújula, que luego serán usadas para normalizar los valores de ésta. El método realiza quinientas lecturas usando el método creado anteriormente *GetCompassAxes(),* compara estas y rescata los máximos y mínimos valores para los ejes X, Y y Z. Retorna un arreglo de *double* de seis elementos, dos para cada eje indicando mínimo y máximo respectivamente. La idea es que mientras se está ejecutando este método se esté rotando lentamente el dispositivo en una superficie plana, con el lado mayor paralelo a la superficie, de la misma forma que se realiza la calibración de la brújula. El código se muestra en el Anexo A.9.
- *Abs(double [] data)*: Método que calcula el módulo de un vector o arreglo ingresado en él, retornando el valor de éste. Esto se logra tomando cada elemento del arreglo, multiplicándolo por sí mismo, sumándolo a una variable auxiliar y luego sacando la raíz cuadrada de esta variable, la cual es finalmente retornada. El método será usado por otro más adelante. El código se muestra en el Anexo A.10.
    - *NormCompassAxes(double [] absCompass)*: Método que normaliza las lecturas de la brújula mediante el método *GetCompassAxes()*. Se evalúan las combinaciones para X e Y mayor o menor que cero (cuatro en total) y según eso se divide cada eje por su máximo o su mínimo según corresponda. Por ejemplo, si X es mayor que cero e Y es menor que cero, X se debe dividir por su máximo e Y por su mínimo. *AbsCompass* representa al vector que contiene los máximos y mínimos valores de la brújula los cuales deben ser ingresados al momento de llamar al método. Ya que la aplicación estará llamando a las funciones una y otra vez, en vez de leer los valores desde un archivo, es menos costoso a nivel de recursos ser ingresados de manera manual. El código se muestra en el Anexo A.11.
- *GetNorthAngle(double [] absCompass)*: Método que retorna un arreglo de *double* con el ángulo entre el eje X y la proyección del vector Norte magnético sobre el plano XY (figura 3.4, ángulo b) como primer elemento y el ángulo entre el vector Norte magnético y el eje Z (figura 3.4 ángulo a). Al igual que el método anterior se le debe entregar el vector que contiene los máximos y mínimos de la brújula.

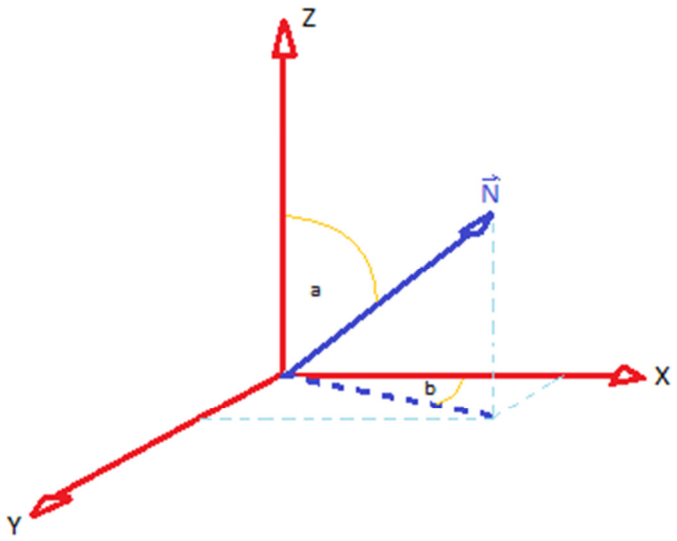

Figura 3.4: Ángulos entregados por el método GetNorthAngle(double [] absCompass).

Para obtener dichos ángulos, se tiene:

$$N_z = Ncos(a)$$

y despejando para a

$$a = arcos(N_z)$$

Luego para b se tiene:

$$tan(b) = \frac{N_y}{N_x}$$

$$b = arctan(\frac{N_Y}{N_x})$$

El código para el método se muestra en el Anexo A.12.

- *getNVector(double [] absCompass)*: Método que retorna el vector Norte magnético haciendo uso del método anterior para obtener los ángulos y con estos obtener los componentes del vector. Aunque ya se tenían los valores que entrega el método, sirve como comprobación del mismo. El código es mostrado en el Anexo A.13.

- *getGVector(double [] absCompass)*: Este método calcula el vector de aceleración gravitatoria a partir del vector Norte magnético. Esto es posible ya que ambos vectores son perpendiculares entre sí. Para la primera y segunda componente (X e Y) del vector aceleración, hay que imaginar los vectores mirados forma perpendicular al plano XY como muestra la figura 3.5. Tomando las componentes X e Y del vector magnético con el signo invertido, se tiene la dirección de la proyección del vector aceleración en el plano XY. Para la magnitud de las componentes, la diferencia entre el módulo de la proyección del vector Norte magnético sobre el plano XY y uno entrega la magnitud del vector aceleración. Así se tiene:

$$G_x = -(1 - |\vec{N}|)\cos(b)$$
$$G_y = -(1 - |\vec{N}|)\,\mathrm{sen}(b)$$

  Para la componente Z del vector aceleración, se requiere tomar el seno del ángulo a (figura 3.5) con el signo invertido y ya que el vector G tiene módulo uno, por lo que entregará su valor real simplemente con la función trigonométrica, como muestra la figura 3.6. Finalmente se tiene:

$$G_z = -sen(a)$$

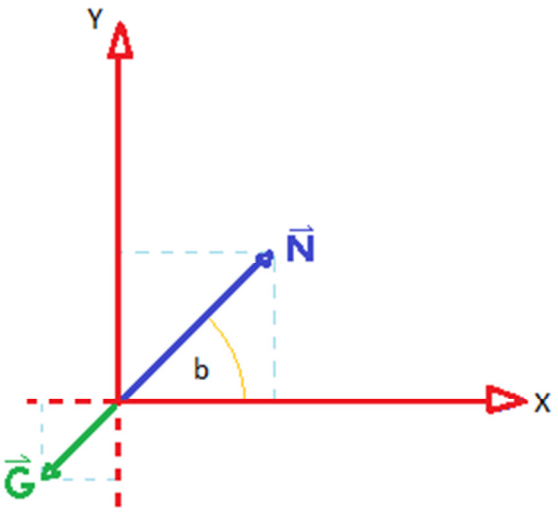

Figura 3.5: Relación entre las componentes X e Y del vector Norte magnético y las componentes X e Y del vector aceleración gravitatoria.

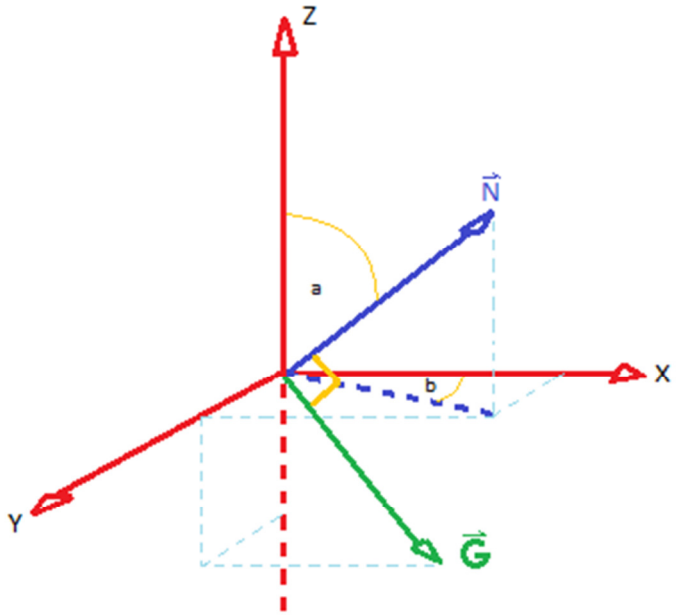

Figura 3.6: Relación entre la componente Z del vector Norte magnético y la componente Z del vector aceleración gravitatoria.

El código para este método se muestra en el Anexo A.14.

- *getAVector(double [] absCompass)*: Método que retorna la aceleración que experimenta el dispositivo. Para lograr esto se deben tomar los datos del acelerómetro y restarle a ellos la aceleración gravitatoria que entrega el método anterior, los cuales son multiplicados por 9.8 para llevarlos a unidades de m/s^2 con lo que se obtiene solo la aceleración que sufre el dispositivo, descartando la aceleración gravitatoria. El código es mostrado en el Anexo A.15.
- *dotProduct(double [] a, double [] b)*: Método que recibe dos vectores o arreglos y retorna el producto punto[4] entre ellos. Al comienzo se verifica si el tamaño de los vectores es el mismo, en caso contrario retorna cero. El código es mostrado en el Anexo A.16.

- *PosVect(double [] Pos, double [] absCompass)*: Método que retorna la posición relativa respecto al Norte, Este y Arriba como un vector de tres componentes respectivamente. El método recibe la posición relativa, la cual se obtiene multiplicando el vector velocidad por el tiempo (este será explicado en el siguiente método). Dado que la posición que es entregada con el vector velocidad está dada respecto a los ejes que se encuentran en el dispositivo, estos no apuntarán en la misma dirección todo el tiempo, por lo que se hace necesario trasladar estos respecto al Norte, el cual es el eje fijo con que se cuenta, y mediante el cual se pueden obtener Norte, Este y Arriba (los nuevos ejes). Para esto, se calculan inicialmente los mismos ángulos que se obtuvieron para el vector Norte magnético del método *GetNorthAngle(double [] absCompass)*, pero para el vector aceleración y mediante la diferencia entre estos ángulos (figura 3.7) se calcula la posición como sigue:

$$Norte = \left|\vec{P}\right| cos(A - a) cos(B - b)$$

$$Este = \left|\vec{P}\right| cos(A - a) sen(B - b)$$

$$Arriba = \left|\vec{P}\right| sen(A - a)$$

[4] Producto punto vectorial es la multiplicación entre las mismas componentes de los vectores las que luego son sumadas

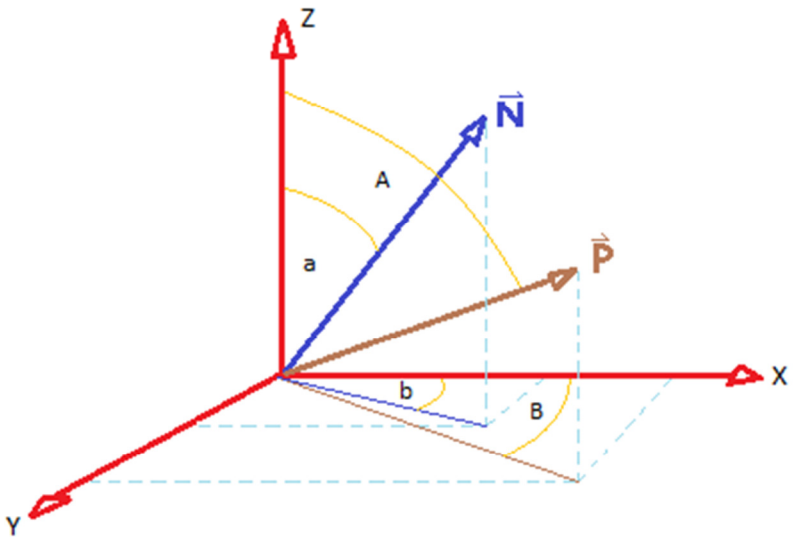

Figura 3.7: Vector aceleración y Norte magnético.

El código se muestra en el Anexo A.17.

- *actionPerformed(ActionEvent event)*: Método que realiza la acción que se ejecutará al presionar el botón. La aplicación desplegará la información en pantalla de la distancia que se desplazado el dispositivo respecto de la posición inicial en cada eje (ejemplo: dos metros hacia el Norte, menos un metro hacia el Este, cero metros hacia arriba) y estos datos serán mostrados en pantalla. Se comienza definiendo las variables que guardarán la posición y velocidad. Al presionar el botón se resetea la brújula (que borra los registros internos) y luego inicia su calibración. Esto toma 28 segundos, tiempo durante el cual se debe rotar el dispositivo en 360 grados en una superficie lo más horizontal posible. Luego durante 30 segundos más se debe seguir rotando el dispositivo para obtener los máximos y mínimos valores de la misma, los cuales son guardados en una variable. A continuación se entra en un ciclo infinito (*while(true)*) donde se calcula la velocidad, la cual se obtiene multiplicando el tiempo entre lecturas por la aceleración y se suma por componentes con la velocidad anterior. Debido a que si existe aceleración cero, no significa que no exista movimiento, si no que se tiene una velocidad constante. Se calcula la posición relativa y se entrega junto con los máximos y mínimos como entrada para el método *PosVect(double*

[] Pos, double [] absCompass) el cual retorna finalmente la posición que será usada. El código se muestra en el Anexo A.18.

4 Pruebas

La aplicación no pudo ser corrida en el *BUG* directamente, por lo que fue necesario desarrollar un simulador, de forma de obtener una validación de la aplicación (uno para la brújula y otro para el acelerómetro), que lo que hace es reemplazar los valores que son entregados por los métodos para la adquisición de datos de la brújula y el acelerómetro, pero no se alteran los métodos creados anteriormente como lo muestra la figura 4.1. Este simulador corre solo dentro de *DragonFly* y utiliza el emulador del *BUG* con el que cuenta.

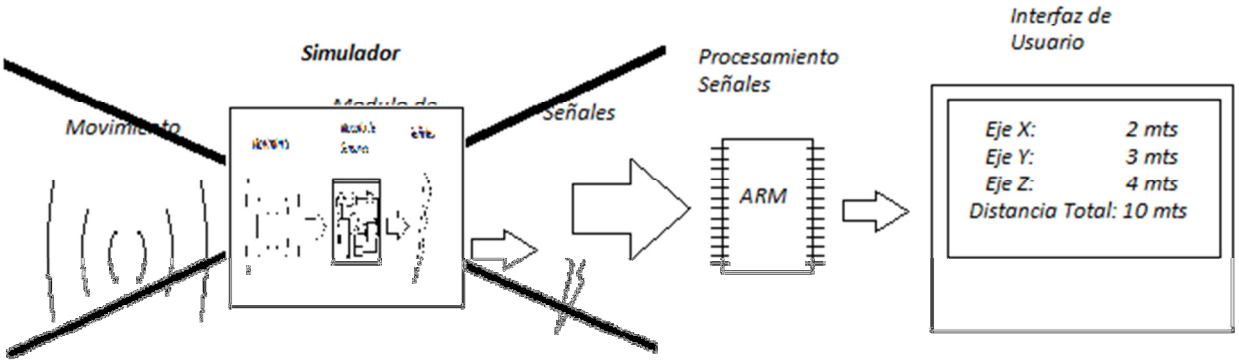

Figura 4.1: Esquema de la aplicación corriendo con el simulador.

4.1 CompassSimulator(int I)

Este método simula los datos que son entregados por la brújula y retorna un arreglo de 3 componentes que indican los ejes X, Y y la última componente simula el ángulo entre el eje X y el Norte magnético. El comportamiento de este varía según el entero (I) que se ingrese según la siguiente la siguiente lista de comportamiento para cada entero:

- *I=1:* Con este valor el simulador busca realizar el proceso de calibración de la brújula, para lo cual genera 500 muestras cuyos máximos valores, en modulo, son 2 y -2. La idea es entregar valores mayores que 1 y -1 ya que luego existe

un método que normaliza estos datos, de modo de probar el funcionamiento de este.

- *I=2:* Con este valor se simula que la brújula se encuentra fija mirando hacia el norte, por lo que solo retorna el valor constante 2 en la componente X, dejando las otras componentes en cero.
- *I=3:* Con este valor se simula que la brújula mire al norte durante las primeras 250 muestras y las siguientes 250 muestras simula que el dispositivo se encuentre mirando hacia abajo y al Norte. Esto se logra gracias a que la brújula entrega solo 2 componentes (X e Y), si estos valores en forma vectorial son menores que 1 (en modulo y normalizados por otro método) se asume que el dispositivo apunta hacia abajo en forma proporcional a la disminución del módulo de estas componentes.
- *I=4:* Con este valor se obtiene un comportamiento similar al caso anterior. Las primeras 250 muestras simula estar mirando hacia el sur y las siguientes 250 muestras simula estar mirando hacia abajo y al Norte (todas las componentes son 0).
- *I=5:* Con este valor se simula durante las primeras 150 lecturas estar mirando al Sur. Luego las siguiente 100 lecturas simula estar mirando hacia el Este. Durante las últimas 250 lecturas simula estar mirando hacia el Oeste.

El código puede ser visto en el Anexo A.19.

4.2 AccelSimulator(String A, int a)

Este método simula los datos que son entregados por el acelerómetro y retorna un arreglo de 3 componentes (X, Y y Z) según los valores ingresados para el *String* "A" y entero "a". El valor de "a" representa un cuarto de "g" de modo que si se ingresa un 4, se representa una aceleración de 9.8m/s^2.

El valor del *String* "A" representa la dirección a la que será aplicada la aceleración representada por el entero "a". Estas direcciones se indican mediante el nombre de cada eje en mayúscula. Así, por ejemplo, *AccelSimulator("X",5)* entrega un valor de 1,25g en la dirección del eje X. Además para cada eje se puede direccionar en

ángulos de 45° al ingresar el *String "*A" como mezcla de ejes. Esto quiere decir, si se ingresa *AccelSimulator("XY",4)* representa una fuerza de módulo 1 en 45° de inclinacion en el plano XY.

4.3 Validación

Para la realización de las pruebas, primero se debió alterar la forma en que los métodos reciben los datos. De esta forma, el método que adquiere los desfases de la brújula no se utiliza, y a los siguientes es necesario agregar los enteros y el *String* A que utilizaran los simuladores de forma de decidir cual se estará utilizando.

4.3.1 Prueba Simulador Brújula

En esta prueba se realiza una lectura en pantalla para cada valor entero de entrada para el simulador. Cabe destacar que las unidades no son metros como se muestra en algunas imágenes, sino que lo importante es solo el valor numérico de ellos:

- *I=1:* Para esta prueba se obtienen los valores de calibración. Algunos de estos valores son mostrados en la tabla 4.1 (para ver todos los valores ver Anexo A.21). Estos datos se compararon con una tabla realizada en Excel con la misma función que realiza el simulador y se obtuvieron los mismos valores.

Contador	X	Y
1	1.467243	1.372149
2	0.152803	1.241661
3	-1.24304	-1.72012
4	-1.97665	-0.31179
5	-1.65718	-1.29144
6	-0.45484	-1.60057
7	0.989821	0.326464
8	1.90715	1.263795
9	1.808432	0.113842
10	0.74626	-1.88143
11	-0.71349	0.078281
12	-1.79312	-1.60836
13	-1.91745	-0.18687

14	-1.02025	-1.88762
15	0.420495	-1.72497
16	1.637222	-0.85517
17	1.981708	0.863181
18	1.270426	-1.01707
19	-0.11768	-0.88717
20	-1.4431	-1.6851

Tabla 4.1: Prueba de calibración para el simulador de la brújula.

- *I=2:* Dado que para esta y las siguientes pruebas no se requiere saber cada uno de los datos (que son iguales para muchas lecturas) es que serán mostradas en imágenes solo los momentos cuando se cambian estos valores. En este caso se obtiene una lectura constante (North=2) que se muestra en la figura 4.2.

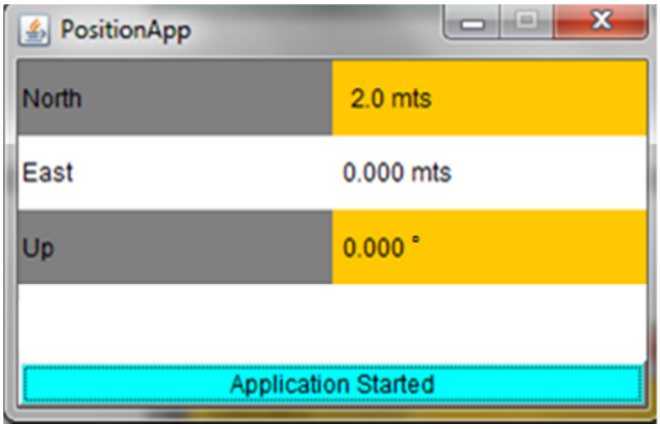

Figura 4.2: Prueba de calibración para el simulador de la brújula I=2.

- *I=3:* Se muestran a continuación dos imágenes para las que varía la salida del simulador para este caso. La primera figura 4.3 muestra la salida simulando estar mirando hacia el norte

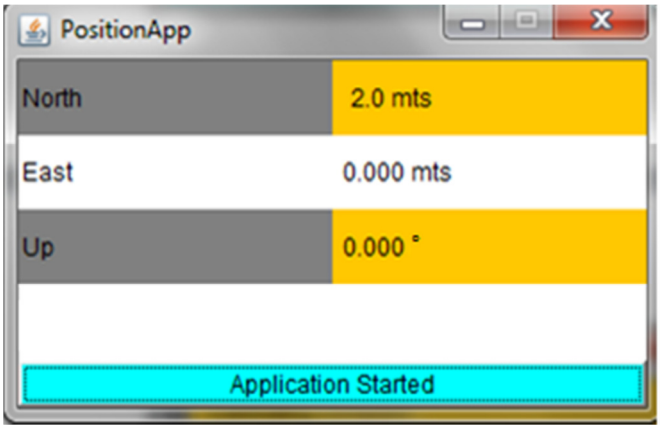

Figura 4.3: Prueba de calibración para el simulador de la brújula I=3, mirando al Norte.

La figura 4.4 muestra cuando se realiza el cambio simulando mirar al Sur.

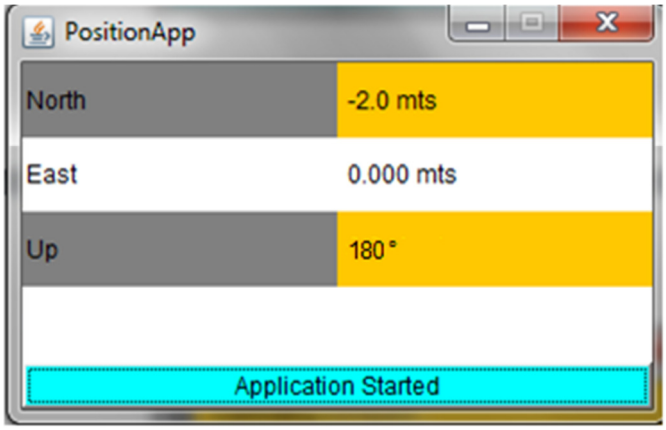

Figura 4.4: Prueba de calibración para el simulador de la brújula I=3, mirando al Sur.

- *I=4:* Se muestra en la figura 4.5 el momento en que el simulador está mirando al Sur.

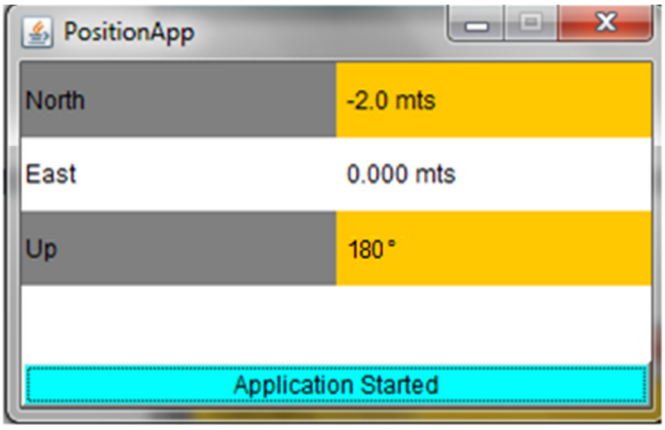

Figura 4.5: Prueba de calibración para el simulador de la brújula I=4, mirando al Sur.

La figura 4.6 muestra al simulador mirando hacia abajo, en dirección al norte.

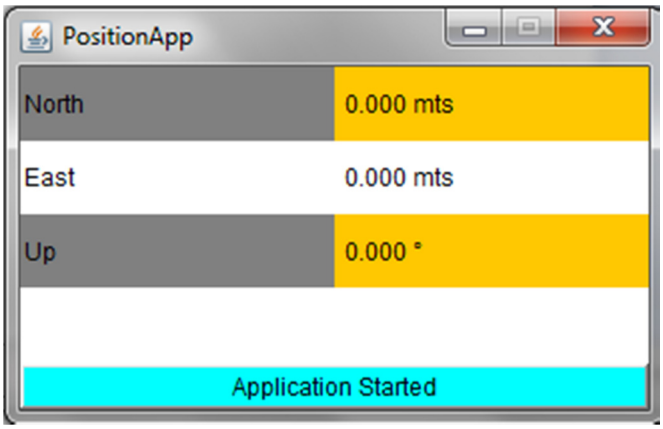

Figura 4.6: Prueba de calibración para el simulador de la brújula I=4, mirando hacia abajo y al Norte.

- *I=5:* En las siguientes 3 figuras se muestran los momentos en que se cambia la salida para los 3 casos. La figura 4.7 muestra la salida mirando al Sur.

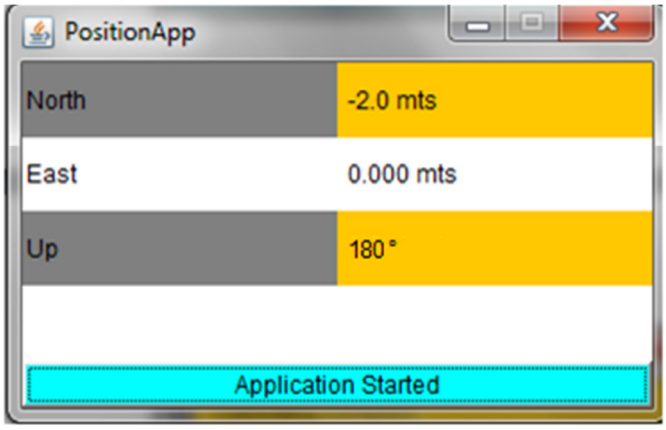

Figura 4.7: Prueba de calibración para el simulador de la brújula I=5, mirando al Sur.

La figura 4.8 muestra la salida para el dispositivo mirando hacia el Este.

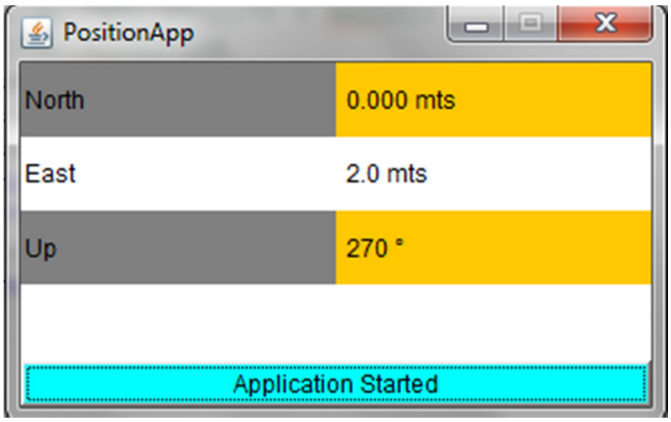

Figura 4.8: Prueba de calibración para el simulador de la brújula I=5, mirando al Este.

La figura 4.9 muestra la salida para el dispositivo mirando hacia el Oeste.

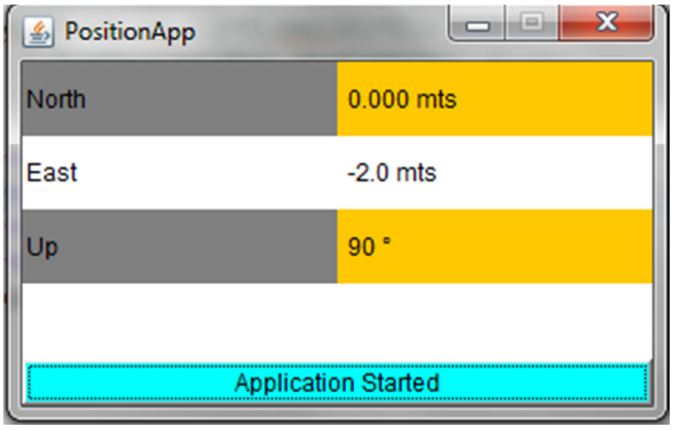

Figura 4.9: Prueba de calibración para el simulador de la brújula I=5, mirando al Oeste.

4.3.2 Prueba Simulador Acelerómetro

Como prueba para el simulador del acelerómetro se verán solo 3 casos. El primer caso será ingresando a la función A=”X”, a=6, con lo que se debiera obtener como resultado 1.5 en X, 0 en Y y 0 en Z. La figura 4.10 muestra el resultado de esta prueba.

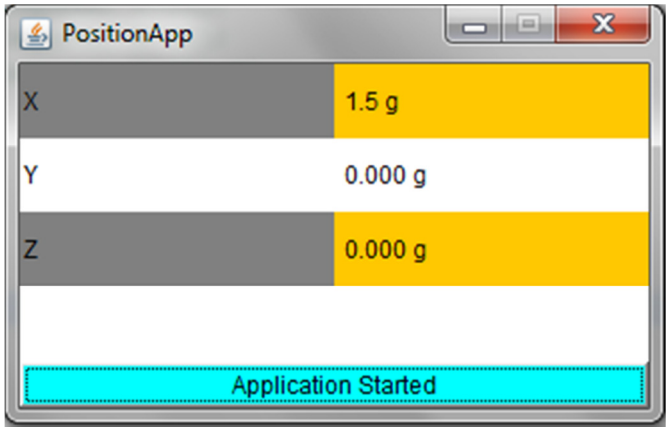

Figura 4.10: Prueba de calibración para el simulador del acelerómetro con A=”X” y a=6.

La segunda prueba tendrá como entrada A=”YZ”, a=7, con lo que se debe tener un valor de:

$$X = 0$$

$$Y = cos(45°) \times 7 \div 4 = 1.2374$$
$$Z = sen(45°) \times 7 \div 4 = 1.2374$$

$$Z = 0$$

La figura 4.11 muestra el resultado de esta prueba.

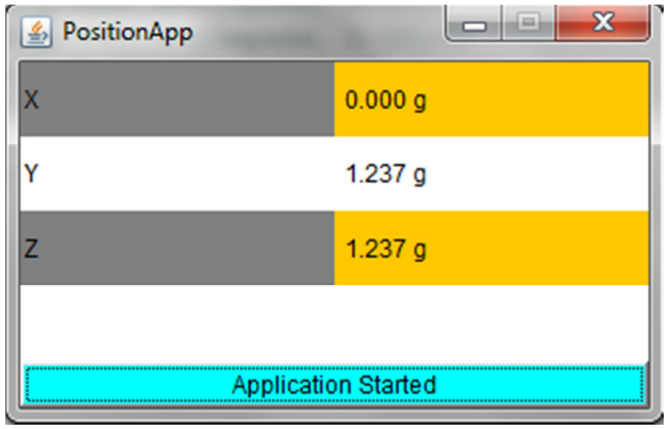

Figura 4.11: Prueba de calibración para el simulador del acelerómetro con A="YZ" y a=7.

Como prueba final se tendrá una entrada de A="XY", a=4, con lo que se debe obtener:

$$X = cos(45°) \times 4 \div 4 = 0.70716$$
$$Y = sen(45°) \times 4 \div 4 = 0.70716$$
$$Z = 0$$

La figura 4.12 muestra el resultado de esta prueba.

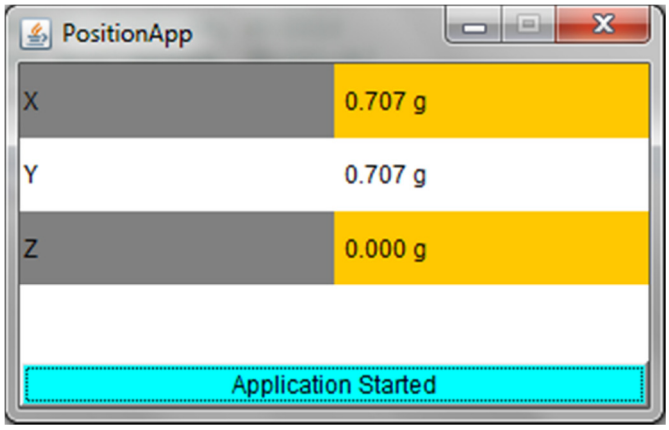

Figura 4.12: Prueba de calibración para el simulador del acelerómetro con A="XY" y a=4.

4.3.3 Prueba Aplicación

Como prueba de validación de la aplicación, primero se tendrá una aceleración de 1 g hacia adelante (que también será el Norte virtual) durante 3 segundos. Luego se detendrá y girara de forma de quedar mirando hacia abajo y al Norte durante 1 segundo. Finalmente se aplicara una aceleración de 1 g hacia el eje Y positivo, que también será el Este virtual durante 2 segundos.

Como se quiere una aceleración de 1 g efectiva hacia adelante, se debe entregar una descomposición vectorial de esta, ya que se debe incluir la aceleración gravitacional apuntando hacia abajo más la aceleración que se quiere. Como se busca tener 1 g hacia abajo y 1 g hacia el eje X, se debe ingresar como parámetros del simulador del acelerómetro A="XZ", a=5,6568. El valor de "a" corresponde a raíz de 2 multiplicada por 4, de forma de obtener raíz de 2 como módulo de aceleración (que equivale a la suma vectorial de 1 más 1 en forma perpendicular) y así, dada la inclinación del dispositivo se obtendrá lo buscado.

Como parámetro del simulador de la brújula se debe ingresar I=2 (Norte fijo). Luego se ingresa para el simulador del acelerómetro A=0, a=0 y para la brújula I=3, de modo de cumplir con el giro sin aceleración. Finalmente se ingresa, similar a los primeros segundos, A="YZ", a=5,6568, para el simulador del acelerómetro e I=4 para el simulador de la brújula de modo de entregar el Sur como dirección.

Durante los primeros 3 segundos, se tendrá (siendo a, v, d y t aceleración, velocidad, desplazamiento y tiempo respectivamente) la fórmula de movimiento

rectilíneo uniformemente acelerado. Tomando velocidad inicial y posición inicial como cero se tiene:

$$d = \frac{1}{2} a t^2$$

Reemplazando para t=3 seg, la distancia resulta:

$$d = 44.1\, m$$

Como luego de 3 segundos la aceleración será 0, se seguirá con la última velocidad de manera constante hacia el Norte sin importar si el dispositivo se encuentre girado. Luego de 3 segundos la velocidad final será:

$$v = at$$
$$v = 29.4 m/s$$

Así, a la distancia que se tenía antes se le debe sumar 29.4m (que es la que se recorre en 1 segundo a velocidad constante) con lo que se tendrá después de 4 segundos una distancia total de:

$$d = 73.5m$$

Luego se tendrá una aceleración de 1 g hacia el eje Y durante 2 segundos. La velocidad constante anterior (que apunta hacia el Norte) se mantendrá durante 2 segundos más, así en el eje Norte se seguirá recorriendo una distancia de:

$$d_{norte} = 29.4 m/s \times 2 seg$$

$$d_{norte} = 58.8m$$

Finalmente, luego de 2 segundos de aceleración hacia el eje Y (Este) se recorrerá una distancia total de:

$$d_{este} = \frac{1}{2} \times 9.8m/s^2 \times (2s)^2$$

$$d_{este} = 19.6m$$

Y como posición final respecto del origen se obtendrá finalmente:

$$d_{origen} = \sqrt{(58.8m + 73.5)^2 + (19.6m)^2}$$

$$d_{origen} = 133.74m$$

Como esta distancia será mostrada en pantalla, tendrá una componente Norte, Este y arriba, que representaran las componentes de X, Y y Z, fijas a los puntos cardinales terrestres.

Finalmente, las siguientes imágenes (4.13 a 4.18) muestran la prueba para la aplicación explicada anteriormente, corriendo cada 1 segundo de ejecución:

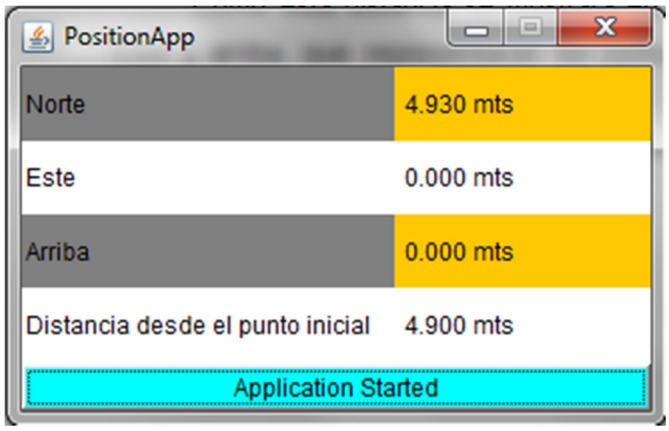

Figura 4.13: Prueba de simulación de la aplicación después de 1 segundo.

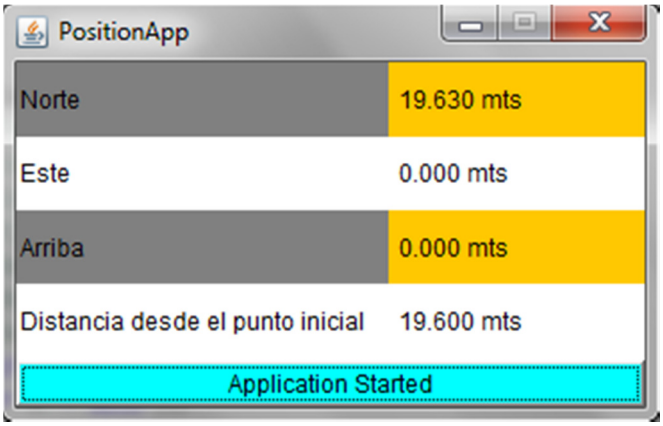

Figura 4.14: Prueba de simulación de la aplicación después de 2 segundos.

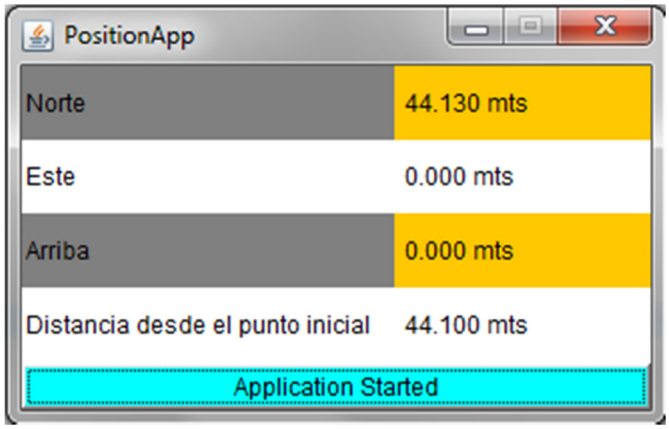

Figura 4.15: Prueba de simulación de la aplicación después de 3 segundos.

Figura 4.16: Prueba de simulación de la aplicación después de 4 segundos.

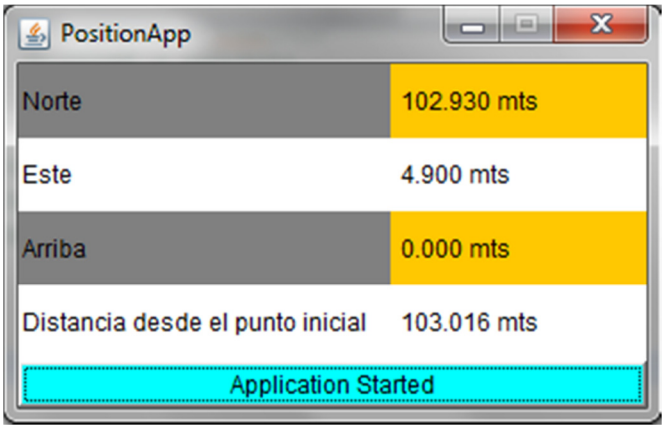

Figura 4.17: Prueba de simulación de la aplicación después de 5 segundos.

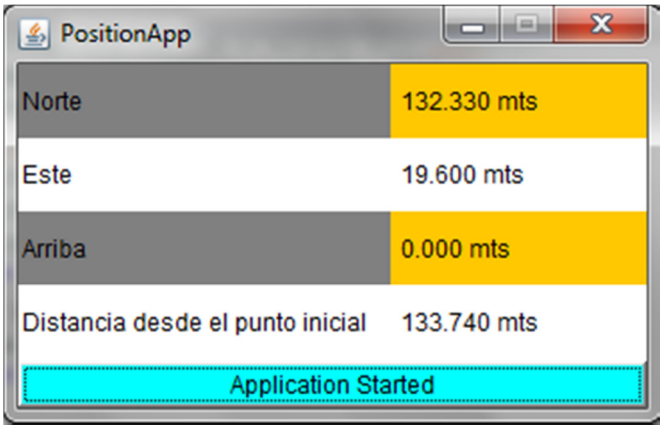

Figura 4.18: Prueba de simulación de la aplicación después de 6 segundos.

5 Conclusión y trabajo a futuro

Una aplicación de posicionamiento que no requiere de medios externos, como satélites o faros de algún tipo, presenta una ventaja de autonomía que la hace ideal para desenvolverse, tanto en los diversos ambientes de trabajo donde no se tenga conexiones inalámbricas, como en aplicaciones turísticas y de entretenimiento en general.

La aplicación realizada dentro de esta memoria no pudo ser corrida directamente en el *BUG*, y aún no se encuentra la causa de ello. Esta situación es extraña ya que la aplicación se comenzó a realizar durante una pasantía de un mes realizada a *New York,* en particular a la empresa *Bug Labs Inc.* creadora de los *BUG*s. Mientras se trabajó allí con sus *BUG*s, la aplicación aún no terminada corría perfectamente, o sea los sensores eran leídos sin ningún problema. Al llegar a Chile y migrar la aplicación al *BUG* que fue facilitado por el profesor guía, bajo el mismo código la aplicación no corrió más.

Como no se pudo correr la aplicación creada directamente en el *BUG*, se desarrollaron 2 simuladores que reemplazaron a los sensores de aceleración y brújula respectivamente y fueron ejecutados en el emulador del *BUG* con que cuenta el entorno de trabajo *Dragonfly*.

Las primeras pruebas fueron para los simuladores y resultaron exitosas, por lo que se pudo realizar una prueba final para la aplicación. Se calculó inicialmente una serie de valores de posición en forma manual, para ciertos valores de aceleración y dirección (de la brújula) y luego se ingresaron estos mismos parámetros al simulador.

La prueba realizada para la aplicación, mostro los mismos resultados que se esperaban para cada uno de los segundos de ejecución de la prueba. Esto significa que si los sensores entregaran datos coherentes la aplicación funcionaría correctamente.

Pese a que aún no se pudo probar la precisión de la aplicación desarrollada, se estima que se obtendrá una baja tasa de muestreo para el caminar humano (menor a dos muestras por segundo), lo que impactará en la precisión de la aplicación. Esto se sabe debido a que se logró leer desde los sensores los datos sin procesar y el tiempo de muestreo era menor a 2 muestras por segundo, por lo que una vez procesados los datos, este tiempo será aún mayor. Debido a que el caminar humano es bastante

errático, este tiempo es demasiado alto, por lo que el error acumulativo inherente a este tipo de aplicación ira creciendo con cada muestra de forma muy rápida y requerirá realizar una recalibración cada poco tiempo.

Otro de los problemas que se encontró, fue en la implementación del sensor brújula, la cual no fue realizada al cien por ciento por parte de *Bug Labs Inc.*, lo que hizo perder la lectura del eje Z, dejando a la aplicación validez solo para inclinaciones de +90 y -90 grados respecto de la posición horizontal inicial, ya que no se puede estimar si el dispositivo se encuentra en posición normal o invertida.

Debido a los problemas que se encontraron durante el desarrollo de la memoria, no fue posible transmitir los datos de manera inalámbrica, pero dado que el *BUG* cuenta con WiFi incorporado y un módulo ZigBee, debiera ser relativamente simple transmitir esta información a través de alguno de estos métodos y se deja como trabajo futuro.

Dada la cantidad de cálculos, el alto nivel del lenguaje con el que se trabajó y la capacidad de procesamiento del *BUG*, la aplicación funcionará de mejor manera sobre objetos que no varíen su vector velocidad tan abruptamente, como automóviles, aviones o barcos. La idea de trabajar en Java y no en C es que brinda la posibilidad de migrar la aplicación al *BUG* 2.0 que está pronto a salir y el cual cuenta con una mayor capacidad de procesamiento entre otras características, lo que repercutirá en la rapidez de la adquisición de datos.

Fuera de las dificultades que se encontraron durante el trabajo, se logró adquirir, manejar y procesar los datos de los sensores simulados. Se trabajó con ellos en forma de vectores tridimensionales aplicando diferentes operaciones para convertir la información en algo legible por el usuario. Se logró además trabajar y desarrollar la interfaz gráfica del *BUG* a través de la pantalla LCD *Touch Screen* que permitió el desarrollo de botones, con lo que se adquirió mayor conocimiento sobre interfaces de usuario.

El trabajo con el *BUG*, requiere siempre superar pequeños y diversos problemas que hacen más lento el avance del proyecto, problemas de conexión, errores del Eclipse o de la versión portable de Linux con que cuenta el *BUG* entre otros. Estos problemas son esperables por tratarse de una plataforma de innovación y lograron ser superados exitosamente, gracias a lo que se adquirió una mayor cantidad de aprendizaje.

Como trabajo futuro se propone encontrar y corregir lo que hace que no corra la aplicación en el *BUG,* junto con exportarla al nuevo *BUG* 2.0 y transmitir esta información de manera inalámbrica ya sea a otro *BUG,* servidor o dispositivo electrónico haciendo uso de WiFi o ZigBee. Esto permitirá obtener una aplicación totalmente funcional de posicionamiento autónomo para personas y prácticamente cualquier vehículo y llevar un seguimiento de ello.

6 Revisión Bibliográfica

[1] Aparatos Eléctricos Electrónicos <http://www2.uca.es/grup-invest/cit/mas_AEE.htm>

[2] Accelerometer and Gyro Tutorial IMU <http://www.instructables.com/id/Accelerometer-Gyro-Tutorial/>

[3] Layered Model of Regulation <http://www.cybertelecom.org/broadband/layers.htm>

[4] OSGI Specification License V1.0 <http://www.osgi.org/Main/OSGiSpecificationLicense>

[5] Pagina Web de Bug labs Inc.<http://www.buglabs.net>

[6] Consierge: Aservice Platform for Resource-Constrained Devices, Jan S. Rellermeyer, Gustavo Alonso, Department of Computer Science ETH Zurich

[7] "QuickStudy: Application Programming Interface (API)". Computerworld. Orenstein, David

[8] OSGI the footings of the foundation of the platform". The Eclipse Foundation. Retrieved

[9] Class AbstractServiceTracker http://bugcommunity.com/development/javadoc/r1.4/bug/com/buglabs/application/AbstractServiceTracker.html

[10] Bug Wiki http://wiki.buglabs.net/index.php/Welcome_to_BUG_Wiki

[11] SVN Code Recopilation http://svn.buglabs.net/svn/!tree/12256

Anexo A: Códigos de la aplicación desarrollada

```
package positionapp;
import service.tracker.*;
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.Filter;
import org.osgi.util.tracker.ServiceTracker;
import com.buglabs.util.ServiceFilterGenerator;

public class Activator implements BundleActivator {
      private ServiceTrackerCustom stc;
      private ServiceTracker st;
public void start(BundleContext context) throws Exception {
      stc = new ServiceTrackerCustom(context);
      Filter f = context.createFilter(ServiceFilterGenerator
      .generateServiceFilter(stc.getServices()));
      st = new ServiceTracker(context, f, stc);
      st.open();
}
public void stop(BundleContext context) throws Exception {
      stc.stop();
      st.close();
      }
}
```

Anexo A.1: Código clase Activator.

```
package service.tracker;
import position.app.PositionApp;
import org.osgi.framework.BundleContext;
import com.buglabs.application.AbstractServiceTracker;
import com.buglabs.bug.module.lcd.pub.IModuleDisplay;
import com.buglabs.bug.module.sensor.pub.ISensorAccelerometer;
import com.buglabs.bug.module.sensor.pub.ISensorCompass;
import com.buglabs.bug.module.sensor.pub.ISensorModuleControl;

public class  ServiceTrackerCustom extends AbstractServiceTracker {
```

```
    private IModuleDisplay display;
    private PositionApp app;
    private ISensorAccelerometer isa;
    private ISensorCompass isc;
    private ISensorModuleControl ismc;
    private BundleContext bundlecontext;
public  ServiceTrackerCustom(BundleContext context) {
    super(context);
}
public boolean canStart() {
    return super.canStart();
}
public void doStart() {
    System.out.println("PositionApp: start");
    display = (IModuleDisplay) getService(IModuleDisplay.class);
    isa = (ISensorAccelerometer) getService(ISensorAccelerometer.class);
    isc = (ISensorCompass) getService(ISensorCompass.class);
    ismc = (ISensorModuleControl) getService(ISensorModuleControl.class);
    app = new PositionApp(display, isa, isc, ismc, bundlecontext);
}
public void doStop() {
    System.out.println("PositionApp: stop");
    app.destroyUI();
}
public void initServices() {
    getServices().add("com.buglabs.bug.module.lcd.pub.IModuleDisplay");
    getServices().add("com.buglabs.bug.module.sensor.pub.ISensorAccelerometer
");
    getServices().add("com.buglabs.bug.module.sensor.pub.ISensorCompass");
    getServices().add("com.buglabs.bug.module.sensor.pub.ISensorModuleControl
");
    }
}
```

Anexo A.2: Codigo clase ServiceTrackerCustom.

```
    package position.app;

    import java.awt.event.ActionListener;
    import java.lang.Math;
```

```
import java.lang.System;
import com.buglabs.bug.module.lcd.pub.IModuleDisplay;
import com.buglabs.bug.module.sensor.pub.ISensorAccelerometer;
import com.buglabs.bug.module.sensor.pub.ISensorCompass;
import com.buglabs.bug.module.sensor.pub.ISensorModuleControl;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;
import org.osgi.framework.BundleContext;

import java.awt.Frame;
import java.awt.Button;
import java.awt.GridLayout;
import java.awt.Label;
import java.awt.Panel;
import java.awt.event.ActionEvent;
import java.awt.BorderLayout;
import java.awt.Color;
import java.text.DecimalFormat;
public class PositionApp implements ActionListener{
```

Anexo A.3: Código de las clases importadas y comienzo del constructor.

```
public PositionApp(IModuleDisplay display,ISensorAccelerometer isa,
ISensorCompass isc, ISensorModuleControl ismc, BundleContext bundlecontext) {
      super();
      this.display = display;
      this.isa = isa;
      this.isc = isc;
      this.ismc = ismc;
      this.bundlecontext = bundlecontext;
      createUI();
}
```

Anexo A.4: Código del constructor de la Clase PositionApp.

```
private void createUI() {
      threePlaces = new DecimalFormat("0.000");
```

```
frame = display.getFrame();
frame.setTitle("PositionApp");
frame.setBackground(Color.WHITE);

buttonStart = new Button("Start");
buttonStart.setBackground(Color.cyan);
buttonStart.addActionListener(this);

label1 = new Label("    PositionApp 1.0");
label1.setBackground(Color.orange);
label2 = new Label("        2010");
label2.setBackground(Color.white);
label3 = new Label("     Autor:");
label3.setBackground(Color.orange);
label4 = new Label("Sebastian Barckhahn");
label4.setBackground(Color.white);

labeli1 = new Label("");
labeli1.setBackground(Color.gray);
labeli2 = new Label("");
labeli2.setBackground(Color.white);
labeli3 = new Label("");
labeli3.setBackground(Color.gray);
labeli4 = new Label("");
labeli4.setBackground(Color.white);

centerPanel = new Panel();
centerPanel.setLayout(new GridLayout(4,1));
centerPanel.add(label1);
centerPanel.add(label2);
centerPanel.add(label3);
centerPanel.add(label4);

leftPanel = new Panel();
leftPanel.setLayout(new GridLayout(4,1));

leftPanel.add(labeli1);
leftPanel.add(labeli2);
```

```
        leftPanel.add(labeli3);
        leftPanel.add(labeli4);

        bottomPanel = new Panel();
        bottomPanel.setLayout(new GridLayout(1,1));
        bottomPanel.add(buttonStart);

        frame.setLayout(new BorderLayout());
        frame.add(centerPanel, BorderLayout.CENTER);
        frame.add(leftPanel, BorderLayout.WEST);
        frame.add(bottomPanel, BorderLayout.SOUTH);
        frame.show();
      }
public void destroyUI(){
        frame.dispose();
}
```

Anexo A.5: Código de la Interfaz de Usuario (UI).

```
private double loadCalibrationData(int axe) {
        final File f = bundlecontext.getDataFile(COMPASS_CALIBRATION_FILE);
        final Properties cp = new Properties();
        try {
        cp.load(new FileInputStream(f));
                } catch (final IOException e) {
                    return 0;
                }
        final String xOffStr = cp.getProperty(X_OFF_KEY);
        final String yOffStr = cp.getProperty(Y_OFF_KEY);
              if ((xOffStr == null) || (yOffStr == null)) {
                    return 0;
        }
              xOff = Double.parseDouble(xOffStr);
              yOff = Double.parseDouble(yOffStr);
        switch(axe){
               case 0:
                    return xOff;
               case 1:
```

```
                return yOff;
            default:
                return 0;
        }
}
```

Anexo A.6: Código del método para obtener los desfases de los ejes de la brújula.

```
public double [] GetCompassAxes(){
    final double[] axes = {(isc.getHx() - loadCalibrationData(0)),
(isc.getHy() - loadCalibrationData(1)), (isc.getHz()),isc.getHeading()};
     return axes;
}
```

Anexo A.7: Código del método para obtener las lecturas de la brújula.

```
    public double [] GetAccelAxes(){
            double [] axes = {0,0,0};
            try{
                axes[0] = isa.readX()*9.8;
                axes[1] = isa.readY()*9.8;
                axes[2] = (isa.readZ() + 0.2)*9.8;
            }catch (Exception e) {
                e.printStackTrace();
            }
            return axes;
     }
```

Anexo A.8: Código del método para obtener las lecturas del acelerómetro.

```
    public double [] GetCompassMaxMin(){
     double[] axes = {0,0,0,0,0,0};
     double minX = 0;
     double maxX = 0;
     double minY = 0;
     double maxY = 0;
     double minZ = 0;
```

```
        double maxZ = 0;
        int count = 0;
              while(count<=500){
                    minX = Math.min(minX,GetCompassAxes()[0]);
                    maxX = Math.max(maxX,GetCompassAxes()[0]);
                    minY = Math.min(minY,GetCompassAxes()[1]);
                    maxY = Math.max(maxY,GetCompassAxes()[1]);
                    minZ = Math.min(minZ,GetCompassAxes()[2]);
                    maxZ = Math.max(maxZ,GetCompassAxes()[2]);
                    count = count + 1;
        }
        axes [0] = minX;
        axes [1] = maxX;
        axes [2] = minY;
        axes [3] = maxY;
        axes [4] = minZ;
        axes [5] = maxZ;
return axes;
}
```

Anexo A.9: Código del método que obtiene los valores mínimos y máximos de la brújula.

```
public double Abs(double [] data){
        double aux = 0;
        for(int i = 0; i < data.length; i++){
              aux = aux + data[i]*data[i];
        }
        return Math.sqrt(aux);
}
```

Anexo A.10: Código del método para obtener el módulo de un vector.

```
    public double[] NormCompass(double [] absCompass){
              double [] aux = {0,0,0,0};
                    if(GetCompassAxes()[0]<0 && GetCompassAxes()[1]<0){
                          aux [0] = (GetCompassAxes()[0])/(absCompass[0]);
                          aux [1] = (GetCompassAxes()[1])/(absCompass[2]);
                          aux [2] = (GetCompassAxes()[2])/(absCompass[4]);
```

```
                aux [3] = GetCompassAxes()[3];
            }
            if(GetCompassAxes()[0]>0 && GetCompassAxes()[1]>0){
                aux [0] = (GetCompassAxes()[0])/(absCompass[1]);
                aux [1] = (GetCompassAxes()[1])/(absCompass[3]);
                aux [2] = (GetCompassAxes()[2])/(absCompass[4]);
                aux [3] = GetCompassAxes()[3];
            }
            if(GetCompassAxes()[0]>0 && GetCompassAxes()[1]<0){
                aux [0] = (GetCompassAxes()[0])/(absCompass[1]);
                aux [1] = (GetCompassAxes()[1])/(absCompass[2]);
                aux [2] = (GetCompassAxes()[2])/(absCompass[4]);
                aux [3] = GetCompassAxes()[3];
            }
            if(GetCompassAxes()[0]<0 && GetCompassAxes()[1]>0){
                aux [0] = (GetCompassAxes()[0])/(absCompass[0]);
                aux [1] = (GetCompassAxes()[1])/(absCompass[3]);
                aux [2] = (GetCompassAxes()[2])/(absCompass[4]);
                aux [3] = GetCompassAxes()[3];
            }
            if(GetCompassAxes()[0]==0 && GetCompassAxes()[1]==0){
                aux [0] = 0;
                aux [1] = 0;
                aux [2] = (GetCompassAxes()[2])/(absCompass[4]);
                aux [3] = GetCompassAxes()[3];
            }
        return aux;
    }
```

Anexo A.11: Código del método para obtener los valores normalizados de la brújula.

```
public double [] GetNorthAngle(double [] absCompass){
    double[] aux = {NormCompass(absCompass)[0],NormCompass(absCompass)[1]};
    double[] angles = {Math.atan(aux[1]/aux[0]),Math.acos(Abs(aux))};
return angles;
}
```

Anexo A.12: Código del método para obtener los ángulos del Norte magnético.

```
public double [] getNVector(double [] absCompass){
      double [] nvector = {Math.cos(GetNorthAngle(absCompass)[0]),
            Math.sin(GetNorthAngle(absCompass)[0]),
            Math.cos(GetNorthAngle(absCompass)[1])};
return nvector;
}
```

Anexo A.13: Código del método para obtener las componentes del vector Norte magnético.

```
public double [] getGVector(double [] absCompass){
     double [] aux = {getNVector(absCompass)[0],getNVector(absCompass)[1]};
     double [] gvector ={-(1-abs(aux))*Math.cos(GetNorthAngle(absCompass)[0]),
      -(1-Abs(aux))*Math.sin(GetNorthAngle(absCompass)[0]),
      -Math.sin(GetNorthAngle(absCompass)[1])};
return gvector;
}
```

Anexo A.14: Código del método para las obtener el vector aceleración gravitatoria.

```
public double [] getAVector(double [] absCompass){
      double [] accel = {GetAccelAxes()[0] - 9.8*getGVector(absCompass)[0],
            GetAccelAxes()[1] - 9.8*getGVector(absCompass)[1],
            GetAccelAxes()[2] - 9.8*getGVector(absCompass)[2]};
return accel;
}
```

Anexo A.15: Código del método para las obtener el vector aceleración.

```
public double dotProduct(double [] a, double [] b){
      double aux = 0;
      if (a.length != b.length){
            return aux;
      }

      else if (a.length == b.length){
      for(int i = 0; i < a.length; i++){
            aux = aux + a[i]*b[i];
```

```
        }
        }
        return aux;
      }
```

Anexo A.16: Código del método producto punto vectorial.

```
public double [] PosVect(double [] Pos, double [] absCompass){
      double [] nangles = GetNorthAngle(absCompass);
      double [] posangles = {Math.atan(Pos[0]/Pos[1]),Math.acos(Abs(Pos))};
      double     []     posVect     =     {Abs(Pos)*Math.cos(posangles[1]     -
nangles[1])*Math.cos(posangles[0] - nangles[0]),
      Abs(Pos)*Math.cos(posangles[1]   -   nangles[1])*Math.sin(posangles[0]   -
nangles[0]),
      Abs(Pos)*Math.sin(posangles[1] - nangles[1])};
return posVect;
}
```

Anexo A.17: Código del método para obtener la posición respecto al Norte magnético.

```
public void actionPerformed(ActionEvent event) {
      double [] posFin = {0,0,0};
      double [] vel = {0,0,0};
if ( event.getSource() == buttonStart ) {
      try{
      label1.setText("    Calibrating Compass ..");
      label2.setText("  Please slowly rotate");
      label3.setText("   the bug in a plane");
      label4.setText("       surface ");

      isc.reset();
      Thread.sleep(250);
      isc.startCalibration();

      label1.setText("   Calibrating Compass ....");
      Thread.sleep(6000);
      label1.setText("   Calibrating Compass ......");
      Thread.sleep(6000);
      label1.setText("   Calibrating Compass ....");
      Thread.sleep(6000);
```

```
label1.setText("   Calibrating Compass ..");
Thread.sleep(6000);
label1.setText("   Calibrating Compass");
Thread.sleep(4000);
isc.stopCalibration();
Thread.sleep(500);

double [] absCompass = GetCompassMaxMin();
label1.setText("   Calibrating Compass ..");
Thread.sleep(6000);
label1.setText("   Calibrating Compass ....");
Thread.sleep(6000);
label1.setText("   Calibrating Compass ......");
Thread.sleep(6000);
label1.setText("   Calibrating Compass ....");
Thread.sleep(6000);
label1.setText("   Calibrating Compass ..");
Thread.sleep(6000);

buttonStart.setLabel("Application Started");
labeli1.setText("North");
labeli2.setText("East");
labeli3.setText("Up");
labeli4.setText("Distance from inicial Point");
label1.setText(" ");

while(true){

Thread.sleep(250);
time0 = System.currentTimeMillis();
      vel[0] = getAVector(absCompass)[0]*T + vel[0];
      vel[1] = getAVector(absCompass)[1]*T + vel[1];
      vel[2] = getAVector(absCompass)[2]*T + vel[2];
      double [] posRel = {vel[0]*T,vel[1]*T,vel[2]*T};
      posFin[0] = posFin[0] + PosVect(posRel,absCompass)[0];
      posFin[1] = posFin[1] + PosVect(posRel,absCompass)[1];
      posFin[2] = posFin[2] + PosVect(posRel,absCompass)[2];
      label1.setText(" " + threePlaces.format(posFin[0]) + " mts");
      label2.setText(" " + threePlaces.format(posFin[1]) + " mts");
```

```
            label3.setText(" " + threePlaces.format(posFin[2]) + " mts");
                  T = (time0 - time1)/1000;
                  time1 = time0;
                  frame.show();
                  }
      } catch (Exception e) {
            e.printStackTrace();
            }
      }
}
```

Anexo A.18: Código de la acción que se ejecutará al presionar el botón.

```
public double[] CompassSimulator(int I){
      double [] aux = {0,0,0};
      if(count2  >= 500)
      count2 = 0;

switch(I){
      case 1:
            aux[0] = 2*Math.cos(180*count2/Math.PI);
            aux[1] = 2*Math.sin(180*count2/Math.PI);
            if(aux[2] > 360)
                  aux[2] = aux[2] - 360;
            aux[2] = count2;

      case 2: //Heading North
            aux[0] = 2;
            aux[1] = 0;
            aux[2] = 0;

      case 3: //North then Down heading north
            if(count2 < 250){
                  aux[0] = 2;
                  aux[1] = 0;
                  aux[2] = 0;
                  }
            if(count2 < 250 && count2 >= 250){
                  aux[0] = 0.3;
                  aux[1] = 0.3;
```

```
                aux[2] = 0;
                }
        case 4: //South then Down heading north
                if(count2 < 250){
                aux[0] = -2;
                aux[1] = 0;
                aux[2] = 180;
        }
                if(count2 < 250 && count2 >= 250){
                aux[0] = 0;
                aux[1] = 0;
                aux[2] = 0;
}
        case 5: //South then east, then West
                if(count2 < 150){
                aux[0] = -2;
                aux[1] = 0;
                aux[2] = 180;
}
                if(count2 > 150 && count2 <= 250){
                aux[0] = 0;
                aux[1] = 2;
                aux[2] = 270;
}
                if(count2 > 250 && count2 <= 500){
                aux[0] = 0;
                aux[1] = -2;
                aux[2] = 90;
                }
                }
                count2 = count2 + 1;
        return aux;
}
```

Anexo A.19: Código del simulador de los servicios entregados por la brújula.

```
public double[] AccelSimulator(String A, double a){
        double [] aux = {0,0,0};
        if(A == "X"){
                aux[0] = a/4;
```

```
            aux[1] = 0;
            aux[2] = 0;
        }
        if(A == "XY" || A == "YX"){
            aux[0] = Math.cos(Math.PI/4)*a/4;
            aux[1] = Math.sin(Math.PI/4)*a/4;
            aux[2] = 0;
        }
        if(A == "Y"){
            aux[0] = 0;
            aux[1] = a/4;
            aux[2] = 0;
        }
        if(A == "YZ" || A == "ZY"){
            aux[0] = 0;
            aux[1] = Math.cos(Math.PI/4)*a/4;
            aux[2] = Math.sin(Math.PI/4)*a/4;
        }
        if(A == "Z"){
            aux[0] = 0;
            aux[1] = 0;
            aux[2] = a/4;
            }
        if(A == "ZX" || A == "XZ"){
            aux[0] = Math.cos(Math.PI/4)*a/4;
            aux[1] = 0;
            aux[2] = Math.sin(Math.PI/4)*a/4;
        }
        if(A == "0"){
            aux[0] = 0;
            aux[1] = 0;
            aux[2] = 0;
        }
        return aux;

}
```

Anexo A.20: Código del simulador de los servicios entregados por el acelerómetro.

Anexo B: Prueba de calibración del simulador de brújula

Contador	X	Y
0	2	1.994154
1	1.467243	1.372149
2	0.152803	1.241661
3	-1.24304	-1.72012
4	-1.97665	-0.31179
5	-1.65718	-1.29144
6	-0.45484	-1.60057
7	0.989821	0.326464
8	1.90715	1.263795
9	1.808432	0.113842
10	0.74626	-1.88143
11	-0.71349	0.078281
12	-1.79312	-1.60836
13	-1.91745	-0.18687
14	-1.02025	-1.88762
15	0.420495	-1.72497
16	1.637222	-0.85517
17	1.981708	0.863181
18	1.270426	-1.01707
19	-0.11768	-0.88717
20	-1.4431	-1.6851
21	-1.99969	-1.99113
22	-1.49094	1.131578
23	-0.18788	1.946824
24	1.215277	0.985536
25	1.970982	-0.33495
26	1.676634	1.940014
27	0.489047	0.502495
28	-0.95908	1.9993
29	-1.89625	-1.93148
30	-1.82318	1.41831
31	-0.7788	-1.19396
32	0.680494	1.921876
33	1.777251	1.926109
34	1.927166	-0.89295
35	1.05037	-0.944
36	-0.38602	0.25142
37	-1.61675	1.998077
38	-1.98615	-1.28967

39	-1.29741	1.74645
40	0.082528	-1.99974
41	1.418503	-0.79193
42	1.998761	1.978294
43	1.514166	-1.87066
44	0.222889	0.405652
45	-1.18713	1.779901
46	-1.9647	1.007708
47	-1.69556	-0.47666
48	-0.5231	1.984042
49	0.928046	0.463531
50	1.884771	1.845679
51	1.837372	-1.99908
52	0.8111	1.212316
53	-0.64729	1.149137
54	-1.76083	-0.6997
55	-1.93628	1.666564
56	-1.08016	1.618806
57	0.351421	1.919097
58	1.595781	-0.63938
59	1.989978	1.591332
60	1.324001	0.890616
61	-0.04735	-0.83165
62	-1.39347	1.927211
63	-1.99721	-1.94448
64	-1.53693	-0.1896
65	-0.25783	-1.60938
66	1.158622	-0.79865
67	1.957815	-1.59448
68	1.713969	-1.45391
69	0.556994	0.954469
70	-0.89672	-1.79405
71	-1.8727	-0.93066
72	-1.85099	1.378014
73	-0.84315	1.853017
74	0.613886	-1.15486
75	1.743869	-1.15319
76	1.944794	-1.99041
77	1.109617	1.355344
78	-0.31672	1.293146
79	-1.57432	-1.57217
80	-1.99319	-1.78611
81	-1.35018	-1.84942

82	0.01215	1.282535
83	1.368005	0.316371
84	1.995046	1.871558
85	1.559213	1.96053
86	0.292699	-1.74712
87	-1.12975	-1.89374
88	-1.95032	1.952366
89	-1.73184	1.92904
90	-0.59071	-1.30682
91	0.865122	-1.28481
92	1.860058	-0.47659
93	1.864036	-0.02537
94	0.874936	-0.26975
95	-0.58029	-1.93198
96	-1.72636	1.997822
97	-1.95271	1.875042
98	-1.13873	-1.3322
99	0.281912	-0.85985
100	1.552363	1.660266
101	1.995783	1.899663
102	1.375936	-0.58256
103	0.02305	1.937756
104	-1.34212	-1.99491
105	-1.99226	-1.73574
106	-1.58102	-0.99482
107	-0.32747	0.173188
108	1.100532	0.444364
109	1.942222	-1.94011
110	1.74918	-0.6106
111	0.624251	-1.87084
112	-0.83325	1.158791
113	-1.84683	1.681037
114	-1.8765	-1.29098
115	-0.90645	-1.99007
116	0.546517	-0.20522
117	1.708326	-0.94225
118	1.960013	-1.43022
119	1.16749	-1.58982
120	-0.24702	-1.99974
121	-1.52993	0.602495
122	-1.99776	-1.95816
123	-1.40127	1.969038
124	-0.05824	0.388443

125	1.315812	-0.01539
126	1.988859	1.510414
127	1.602328	-1.28922
128	0.362146	1.892658
129	-1.07097	1.989782
130	-1.93352	1.471782
131	-1.76598	-1.21366
132	-0.65759	0.043416
133	0.801125	1.880101
134	1.833039	-1.95273
135	1.88839	1.964727
136	0.937688	-0.62636
137	-0.51257	1.776912
138	-1.68976	-1.08483
139	-1.96671	0.802555
140	-1.19589	1.122066
141	0.212054	-0.80928
142	1.507022	-1.99772
143	1.999115	1.983846
144	1.426166	0.063871
145	0.093417	-1.60415
146	-1.2891	1.998936
147	-1.98484	-1.1714
148	-1.62314	1.896952
149	-0.39671	1.346364
150	1.041079	0.081582
151	1.924223	-0.57991
152	1.782224	1.99985
153	0.690734	1.906932
154	-0.76875	-0.12758
155	-1.81868	1.011076
156	-1.89969	-1.7926
157	-0.96863	1.734873
158	0.478471	1.515629
159	1.670666	1.990719
160	1.972803	-0.12781
161	1.223916	1.693924
162	-0.17702	1.31539
163	-1.48365	0.365623
164	-1.99985	-1.99279
165	-1.45062	-1.98109
166	-0.12856	-1.76668
167	1.261989	-0.10007

168	1.980208	0.705102
169	1.643458	-0.16882
170	0.431145	-0.83364
171	-1.01086	-1.95965
172	-1.91433	-0.53881
173	-1.79792	-1.22498
174	-0.72366	1.165294
175	0.736136	-1.94549
176	1.80375	0.639021
177	1.910404	0.954532
178	0.999278	1.297295
179	-0.44422	-0.62764
180	-1.65106	-0.6875
181	-1.97828	-0.49481
182	-1.25156	-1.04058
183	0.141933	1.923125
184	1.459814	1.85053
185	1.99997	1.993891
186	1.474629	0.653614
187	0.163669	0.094417
188	-1.23449	-1.99797
189	-1.97496	-0.11933
190	-1.66326	-1.73504
191	-0.46545	-1.99876
192	0.980335	-0.74102
193	1.903839	1.533422
194	1.81306	-0.41346
195	0.756361	-1.2039
196	-0.70329	-1.03666
197	-1.78826	-1.87295
198	-1.92053	0.164549
199	-1.02961	-1.2849
200	0.409832	-1.99356
201	1.630937	-1.43737
202	1.983149	1.009069
203	1.278826	-1.6987
204	-0.1068	0.326445
205	-1.43553	-1.07652
206	-1.99947	-1.98858
207	-1.49818	1.700434
208	-0.19872	1.84944
209	1.206602	0.036443
210	1.969103	-0.54492

211	1.682551	1.667872
212	0.499609	-0.68803
213	-0.9495	1.678051
214	-1.89276	-1.99611
215	-1.82764	1.728331
216	-0.78883	-1.87425
217	0.670234	1.292292
218	1.772226	1.693855
219	1.930052	-1.17536
220	1.05963	-1.70645
221	-0.37532	-0.93608
222	-1.61031	1.831941
223	-1.9874	-1.39584
224	-1.30569	1.10872
225	0.071636	-1.64162
226	1.410798	-1.5007
227	1.998348	1.970782
228	1.521265	-1.43814
229	0.233718	1.46868
230	-1.17834	1.999096
231	-1.96263	1.204859
232	-1.70132	0.17802
233	-0.53361	1.492007
234	0.918377	1.417756
235	1.881097	1.64394
236	1.84165	-1.92457
237	0.821052	0.161911
238	-0.63697	1.866636
239	-1.75564	-0.11944
240	-1.93898	1.817065
241	-1.08932	0.812672
242	0.340685	1.242413
243	1.589187	0.104798
244	1.991038	1.661948
245	1.332151	1.60142
246	-0.03645	-1.73807
247	-1.38563	1.503875
248	-1.99661	-1.9271
249	-1.54388	-0.94701
250	-0.26864	-0.62164
251	1.14972	0.198198
252	1.955559	-1.7368
253	1.71956	-1.81245

254	0.567455	1.779552
255	-0.88697	-1.05217
256	-1.86885	-0.52028
257	-1.85509	1.002678
258	-0.85302	1.967747
259	0.603503	-0.04134
260	1.738506	-1.59341
261	1.947309	-1.99787
262	1.11867	1.906155
263	-0.30595	1.937432
264	-1.56757	-1.92221
265	-1.99406	-1.82869
266	-1.3582	-1.31979
267	0.00125	0.143116
268	1.360033	1.154896
269	1.994249	1.837445
270	1.566016	1.963706
271	0.303477	-1.98808
272	-1.12074	-1.96444
273	-1.94788	1.993811
274	-1.73727	1.675024
275	-0.60112	-0.23131
276	0.855281	-1.90506
277	1.856024	-0.9088
278	1.867958	0.42096
279	0.884725	0.82568
280	-0.56985	-1.88776
281	-1.72084	1.869342
282	-1.95503	1.765876
283	-1.14767	-0.42966
284	0.271116	0.346495
285	1.545468	1.103152
286	1.996461	1.922518
287	1.383826	-1.35982
288	0.033949	1.861504
289	-1.33401	-1.71999
290	-1.99127	-1.67677
291	-1.58767	-0.27802
292	-0.33822	-1.00955
293	1.091414	-0.58791
294	1.939592	-1.84521
295	1.754439	-0.01786
296	0.634597	-1.94667

297	-0.82333	0.099021
298	-1.84262	1.891246
299	-1.88025	-1.5865
300	-0.91616	-1.58536
301	0.536023	-1.29457
302	1.702632	-0.32723
303	1.962152	-1.24853
304	1.176323	-1.97876
305	-0.2362	-1.6463
306	-1.52289	1.302978
307	-1.99824	-1.96873
308	-1.40902	1.627046
309	-0.06914	1.461855
310	1.307583	-0.92205
311	1.98768	1.418521
312	1.608827	-1.75718
313	0.372861	1.174704
314	-1.06175	1.820236
315	-1.93071	1.235148
316	-1.77107	-1.61949
317	-0.66788	-1.07512
318	0.791126	1.949632
319	1.828652	-1.78385
320	1.891952	1.999743
321	0.947302	-1.52786
322	-0.50203	0.941008
323	-1.6839	-1.57758
324	-1.96866	0.593171
325	-1.20461	0.191836
326	0.201212	-1.72256
327	1.499834	-1.79248
328	1.99941	1.987845
329	1.433786	0.903228
330	0.104304	-0.60439
331	-1.28075	1.804385
332	-1.98347	-1.04078
333	-1.62949	1.547689
334	-0.40738	1.951573
335	1.031756	1.087563
336	1.921222	-0.24391
337	1.787144	1.914109
338	0.700953	1.256055
339	-0.75868	0.982241

340	-1.81411	0.530796
341	-1.90307	-1.58837
342	-0.97816	0.966658
343	0.46788	1.989186
344	1.664649	1.808542
345	1.974565	0.074018
346	1.232519	1.995424
347	-0.16616	0.190758
348	-1.47632	-0.46807
349	-1.99996	-1.99375
350	-1.4581	-1.91586
351	-0.13944	-1.98173
352	1.253514	0.843768
353	1.978648	0.535324
354	1.649645	0.533456
355	0.441782	0.357145
356	-1.00144	-1.47565
357	-1.91114	-0.87935
358	-1.80267	-0.75514
359	-0.73381	1.866662

Anexo B. 1: Tabla de prueba de calibración para el simulador de la brújula.