Universidad de la República
Facultad de Ingeniería

ININ
Interfaces Inclusivas

Memoria de proyecto presentada a la Facultad de Ingeniería de la Universidad de la República por

Santiago Bacelar, Rocío Curiel, Santiago Vanoli

en cumplimiento parcial de los requerimientos para la obtención del título de Ingeniero Electricista.

Tutores

Pablo Pérez-Nicoli ..... Universidad de la República
Francisco Veirano ..... Universidad de la República

Tribunal

Federico Lecumberry ..... Universidad de la República
Pablo Pérez-Nicoli ..... Universidad de la República
Leonardo Steinfeld ..... Universidad de la República
Francisco Veirano ..... Universidad de la República

Montevideo
lunes 23 agosto, 2021

ININ
Interfaces Inclusivas, Santiago Bacelar, Rocío Curiel, Santiago Vanoli.

Esta tesis fue preparada en LaTeX usando la clase iietesis (v1.1).
Contiene un total de 169 páginas.
Compilada el lunes 23 agosto, 2021.
http://iie.fing.edu.uy/

"La decisión siempre pronta para bregar con energía y entusiasmo por el mejoramiento de las condiciones de vida".

Dr. Ricardo Caritat

Esta página ha sido intencionalmente dejada en blanco.

Agradecimientos

El presente proyecto no hubiera sido posible sin la ayuda de personas e instituciones que prestaron su conocimiento, tiempo y recursos para ayudarnos en el desarrollo.

En primer lugar, agradecemos a la Universidad de la República y en particular a la Facultad de Ingeniería por permitirnos el acceso a los espacios en los que se llevó a cabo gran parte del trabajo, así como a los equipamientos necesarios para realizar pruebas. También agradecemos la financiación recibida por la Unidad de Extensión de la Facultad de Ingeniería, Universidad de la República.

Buena parte de las decisiones que enriquecieron el diseño salieron de sugerencias o comentarios hechos tanto por nuestros tutores como por las funcionarias de la Escuela Nº 200 “Dr. Ricardo Caritat” y Casa de Gardel; a todos ellos muchas gracias. En particular queremos agradecer a Paula Ekaterina Mitrani Goncalvez (directora de la escuela), por permitirnos visitar las instalaciones y mostrarnos dispositivos que usan hoy en día; esto nos permitió entender mucho mejor las necesidades de los alumnos. Así como a María Agustina Pérez (Licenciada en Terapia Ocupacional) por dedicar su tiempo a probar las etapas intermedias de diseño aportando comentarios y sugerencias.

A todos los mencionados, muchas gracias por ser parte de este proyecto.

Esta página ha sido intencionalmente dejada en blanco.

Resumen

En los tiempos modernos las computadoras y los dispositivos móviles se han vuelto parte fundamental de nuestro día a día, estando presente en gran parte de nuestra rutina diaria. Es por esto que surge la necesidad de garantizar (o al menos simplificar) la accesibilidad de todas las personas a este tipo de herramientas. Con base en esta premisa surge el presente proyecto, en el que se busca desarrollar un dispositivo de interfaz humana (o *HID* por sus siglas en inglés) que emule las funciones básicas de un *mouse*, pero contemplando las distintas necesidades de los usuarios desde el punto de vista motriz. Para entender los aspectos a cubrir y las necesidades de los distintos usuarios se trabajó en conjunto con funcionarias del instituto Casa de Gardel y de la Escuela Nº 200 "Dr. Ricardo Caritat".

El dispositivo final puede configurarse en cuatro modos de uso distintos, que permiten el control y manejo del cursor mediante la realización de distintos gestos, así como a través de botones externos que faciliten tanto el movimiento como la realización de *clicks*. Se contempló también que la configuración del dispositivo pueda realizarse prácticamente sin ningún conocimiento técnico, simplemente estableciendo una conexión vía *Bluetooth* desde cualquier sistema operativo. Esto permite que pueda ser utilizado, por ejemplo en escuelas, sin necesidad de capacitación específica para ello.

Se hizo hincapié en la necesidad de que el dispositivo fuera inalámbrico, con una autonomía que permita su uso diario frente a un PC y que fuera además de bajo costo, pudiendo ser fácilmente replicable; para lograr esto se diseñaron los *PCB* y el firmware necesario. En esta documentación se detalla tanto el *hardware* cómo el *sofware* necesarios para replicar y/o mejorar el prototipo desarrollado.

Esta página ha sido intencionalmente dejada en blanco.

Tabla de contenidos

Capítulo 1

Introducción

Este capítulo tiene la finalidad de introducir al lector en la tesis Interfaces Inclusivas (ININ). En el mismo se explica el contexto y la motivación del proyecto, así como también se incluirán los requerimientos y una breve descripción del mismo.

1.1. Motivación

Las computadoras personales y dispositivos móviles están sumamente presentes en la vida diaria de las personas, siendo usadas para recreación, comunicación, estudio, trabajo, e incluso como fuente fundamental de información. Estas herramientas son manejadas a través de dispositivos de interfaz humana (*HID*), como pueden ser teclado, *mouse*, pantalla táctil, etc., cuya manipulación requiere de importantes habilidades motrices y comúnmente no son accesibles ni inclusivas. Esto genera que las personas con discapacidad motriz se enfrenten con nuevas barreras de accesibilidad que profundizan su exclusión.

La mayoría de las interfaces que se pueden encontrar en el mercado se focalizan en realizar modificaciones a ratones y teclados para que sean utilizables por los usuarios en cuestión. Por otro lado las soluciones comerciales son generalmente costosas y cuentan con poca adaptabilidad a las necesidades particulares de cada usuario (regulación de sensibilidad, de tamaño, resistencia y otros según la funcionalidad). En la tabla 1.1 se listan a modo de ejemplo soluciones que se encuentran en el mercado (una tabla más extensa se puede encontrar en la Sección A.1). Debido a que las opciones allí mostradas abarcan distintas modalidades de uso, sirven como referencia a la hora de diseñar una solución más integral. Como factor a destacar, ninguna de ellas cumple con todas las características destacadas en la tabla; por un lado, se tiene el *5 Switch* que funciona únicamente como una botonera. Es decir, permite realizar casi todas las acciones típicas de un *mouse* convencional (exceptuando *click* derecho), pero limita su uso al accionado de botones.

Por su parte, las soluciones *Quha Zono* y *enPathia* permiten realizar una amplia gama de acciones, y tienen la ventaja de poder ser utilizadas con distintas partes del cuerpo. Como clara desventaja de estas opciones está que ambas requieren de la utilización de un programa externo, ya sea para para su uso o para la configuración. Esto genera que los

dispositivos no sean completamente *Plug&Play*. Sumado a esto viene acoplado el hecho de que ninguno de ellos puede ser usado íntegramente en dispositivos móviles, debido a la falta de compatibilidad de los programas asociados.

Nombre	Precio aproximado	*Plug&Play*	*Click* izquierdo	*Click* derecho	Movimiento del cursor	*Click* y movimiento simultáneo	Usable solo con botones	Usable solo con movimientos
Quha Zono [34]	USD 1000	No[1]	Si[2]	Si[2]	Si	No[2]	No	Si
5 Switch [12]	USD 129	Si	Si	No	Si	Si	Si	No
enPathia [18]	227 €	No	Si	Si	Si	Si	No	Si

Tabla 1.1: Tabla comparativa de soluciones disponibles en el mercado

[1]Requiere de un programa externo para cambiar la configuración por defecto.
[2]Mediante accesorio.

1.2. Descripción del Proyecto

El objetivo general del proyecto fue desarrollar un dispositivo de interfaz humana capaz de sustituir un *mouse*, pudiendo ser utilizado por cualquier persona. En particular se buscó satisfacer principalmente las necesidades de aquellas personas que enfrentan una situación de discapacidad motriz la cual dificulta o impide el uso de un *mouse* o panel táctil convencional.

Este proyecto aborda la necesidad de desarrollar interfaces persona-computadora, que puedan ser fácilmente ajustadas/programadas para atender las necesidades de los usuarios y sus entornos. Se busca que el costo total del dispositivo no supere el de las soluciones disponibles en el mercado.

Para lograr esto, se propuso una solución compuesta por un microcontrolador capaz de comunicarse con el PC, así como sensores que permitan captar las acciones del usuario: acelerómetro, giroscopio y magnetómetro para captar los movimientos, y sensores capacitivos para funcionar como botones.

Dentro del alcance del proyecto se encuentra:

- Desarrollo del *firmware* y *hardware* necesario.
- Encapsulado a nivel de prototipo.
- Un dispositivo que pueda ser fácilmente usado en cualquier computadora o dispositivo móvil, sin necesidad de un *software* externo.

Queda por fuera del alcance del presente proyecto:

- Que sea una solución comercial.
- La evaluación sobre su desempeño con los usuarios finales.
- Encapsulado universalmente adaptable.

El equipo docente que apoyó el proyecto está formado por docentes de la Facultad de Ingeniería y de la Escuela Universitaria de Tecnología Médica, los últimos nucleados

en una Unidad Docente Asistencial (UDA), permitiendo tener diferentes enfoques para abordar el problema y así enriquecer los resultados del proyecto.

Algunas de las decisiones de desarrollo se basaron en interacciones con la Escuela Nº 200 "Dr. Ricardo Caritat", donde también se realizaron pruebas de concepto a modo de validar las ideas y/o prototipos. Es por esto que, a modo de simplificar las futuras referencias, se considerará a la escuela como el cliente del presente proyecto.

1.3. Requerimientos

En esta sección se presentan los requerimientos necesarios para el dispositivo diseñado. Los mismos surgieron a través de conversaciones con fisioterapeutas y profesionales que ayudaron en el transcurso de este proyecto. Estos criterios son relevantes, dado que el diseño fue realizado con la expresa intención de cumplir con estos requerimientos.

1.3.1. Requerimientos Mecánicos

- El encapsulado no debe superar el tamaño de 2 $cm \times 6\ cm \times 11\ cm$.
- El método de sujeción debe ser adaptable para facilitar el uso en diferentes partes del cuerpo del usuario.
- El método de sujeción debe ser apto para tener contacto con la piel del usuario.

1.3.2. Requerimientos Eléctricos

- El dispositivo poseerá una batería recargable mediante un conector *micro USB*, con una autonomía de al menos 10 horas.
- El dispositivo incluirá alguna interfaz que permita al usuario seleccionar el modo de uso.

1.3.3. Requerimientos de Desempeño

- El microcontrolador deberá tener la capacidad de procesar las señales recibidas para detectar los gestos establecidos.
- Los algoritmos de detección de gestos deberán ser capaces de filtrar movimientos involuntarios de los usuarios, así como también el ruido existente en las señales.

1.3.4. Requerimientos Funcionales

- El dispositivo deberá tener al menos un acelerómetro con un rango de operación de al menos $\pm 2g$ y 12 *bits* de resolución.
- El microcontrolador deberá disponer de al menos 16 kB de memoria no volátil libre para guardar configuraciones de cada usuario.

- El dispositivo deberá tener una interfaz que permita utilizar un protocolo de comunicación inalámbrica.

Capítulo 2

Descripción del Sistema

El objetivo de este capítulo es realizar una descripción general del proyecto, esto es: las especificaciones que debe cumplir el dispositivo, diagramas generales de *hardware* y características de la solución final.

La función principal del dispositivo es controlar el cursor de un *mouse* de computadora estando sujetado a una parte del cuerpo. En el desarrollo se trabajó en lograr un sustituto al *mouse* convencional para personas que tengan dificultades para controlar una PC. La solución obtenida es un dispositivo que puede ser sujetado a distintas partes del cuerpo, y que luego de una configuración inicial, se conecta automáticamente con la computadora o dispositivo móvil del usuario vía *Bluetooth*. El dispositivo permite un manejo fluido del cursor de forma completamente inalámbrica, dado que la alimentación la brinda una batería recargable que se ubica dentro del encapsulado. Dada la amplia gama de usuarios que se tiene en mente, se desarrollaron cuatro modos de uso incrementales en sus requerimientos de acción del usuario, a los que nos referiremos con los nombres de Barrido, Botonera, Cabeceo y Gestos. Los mismos se describen en la Sección 2.2.

2.1. Diagrama General

En términos generales, el *hardware* del dispositivo se puede separar en dos bloques bien definidos como se ve en la figura 2.1. Por un lado, se puede ver un bloque principal encargado de las comunicaciones *Bluetooth* que, por si mismo, no tiene la opción de soportar botones. Para esto se conecta un segundo bloque (denominado Botonera (mediante un conector *RJ45*), al que se anexan los botones utilizando conectores *Jack* de 3,5 *mm*.

En la figura 2.2 podemos ver un diagrama de la estructura general del bloque principal. Este será el encargado de conectarse mediante *Bluetooth* al dispositivo que se desea controlar, procesar la información recibida a través de los sensores, y con base en eso, determinar el accionar del cursor. También cuenta con la presencia de una memoria externa al microcontrolador, permitiendo persistencia y respaldo de datos para los casos que sea necesario. A su vez posee autonomía de funcionamiento al ser alimentado mediante una batería, la cual puede ser cargada mediante un puerto *micro USB*.

Por otro lado mediante un conector *RJ45* podrá conectarse con el otro bloque de

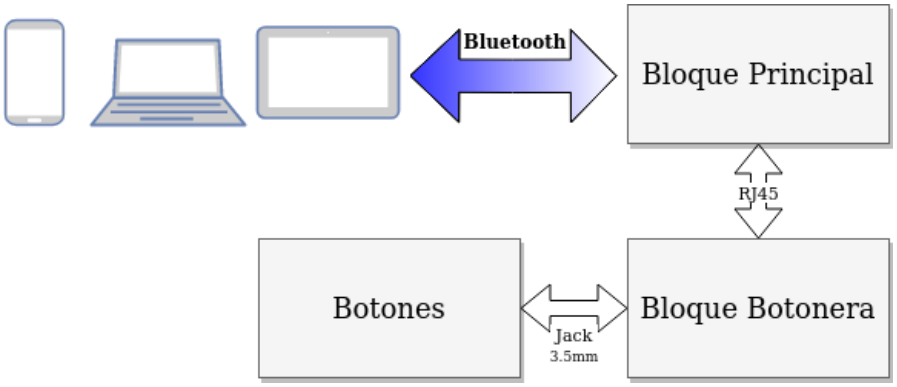

Figura 2.1: Diagrama general del dispositivo

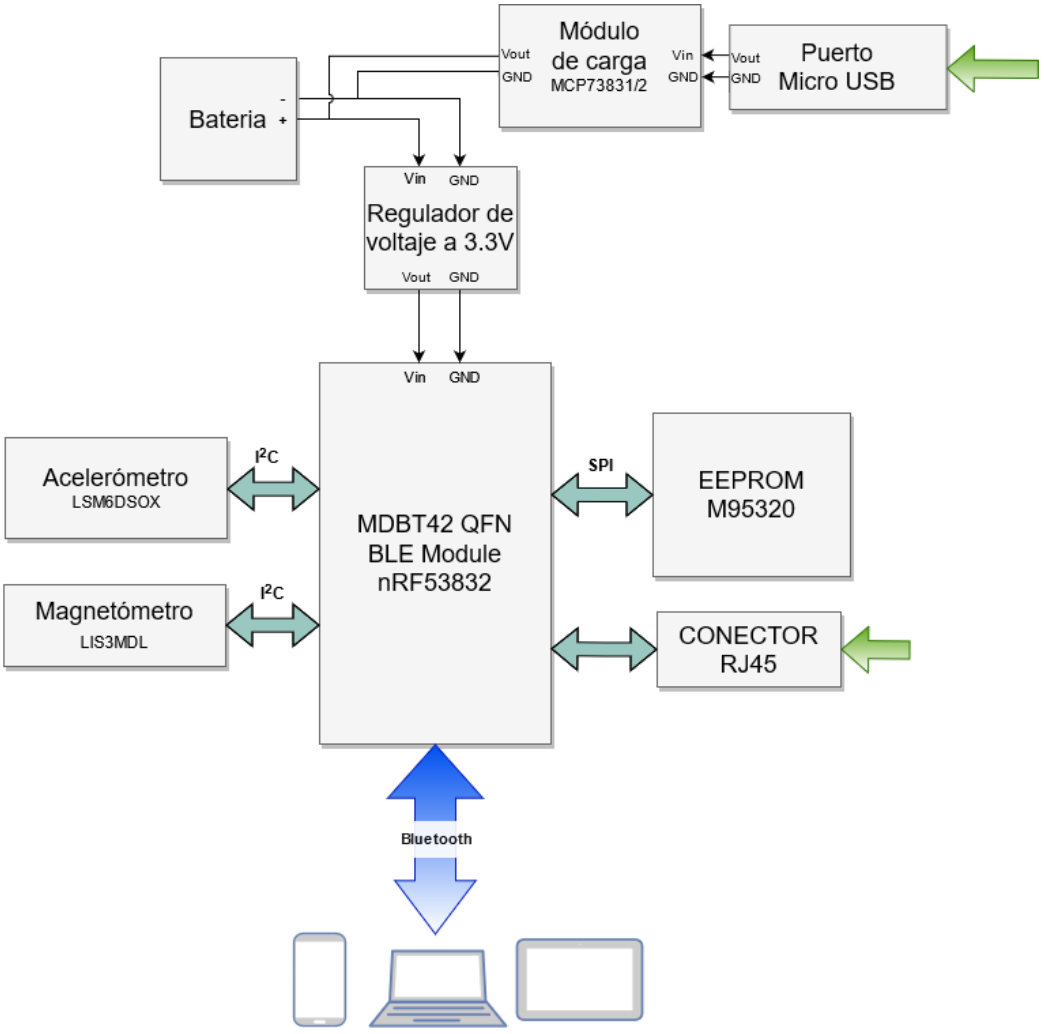

Figura 2.2: Diagrama general del bloque principal

hardware, el cual se aprecia en la figura 2.3. Este será conectado al principal cuando se desee utilizar en modo Botonera o Barrido, moviendo el cursor mediante la pulsación de distintos botones. Este bloque tiene la particularidad de aceptar dos modalidades en sus pulsadores; por un lado se puede utilizar del tipo capacitivos posicionando el selector en ese modo y conectando *pads* externos con la característica que el usuario desee. Además, se permite conectar botones externos con salida de conector de audio *Jack* de 3,5 mm siempre que sean normal abierto y estén conectados entre $IN2$ y V_{in}.

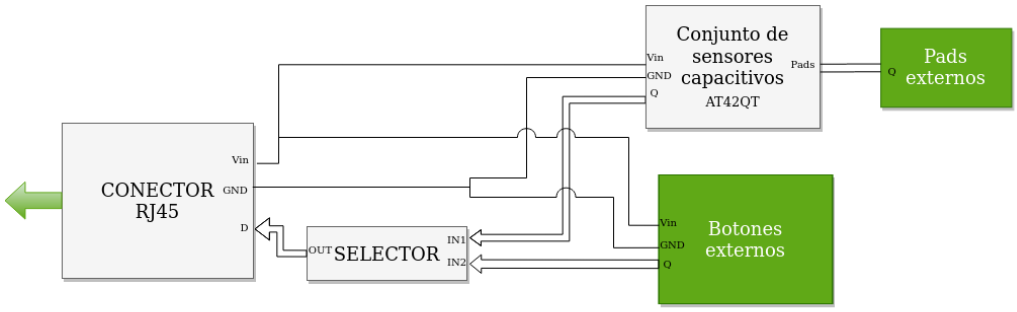

Figura 2.3: Diagrama botonera

2.2. Modos de Uso

Según lo mencionado, para cubrir diversas necesidades se implementaron cuatro modos de uso que se detallarán a continuación.

2.2.1. Barrido

Este modo de uso es el más sencillo de usar, ya que el movimiento del cursor es semi-autónomo, necesitando una interacción mínima por parte del usuario. Está compuesto por el dispositivo y cuatro botones que se conectan según lo mencionado en la Sección 2.1; uno controla el movimiento en el eje vertical, otro en el horizontal y los dos restantes permiten hacer *click* izquierdo y derecho.

En esta configuración cuando se presiona el botón correspondiente a un eje, el cursor comienza un "barrido": esto es, recorrer en bucle de borde a borde la pantalla en el eje seleccionado. Se puede alternar entre este movimiento y dejar el cursor quieto presionando el botón correspondiente al eje, al retomar el movimiento este se reanuda en la dirección que traía. De esta forma, se puede acceder fácilmente a cualquier ubicación en la pantalla a través de una composición de movimientos verticales y horizontales. Los botones de *click* se comportan exactamente igual que en un *mouse* convencional.

La idea de realizar este modo, surge en una de la interacciones con la Escuela Nº 200 en donde los profesionales que trabajan allí mencionaron una herramienta de barrido similar a este modo. Los detalles sobre estas interacciones se encuentran en la Sección A.2.

2.2.2. Botonera

Este modo fue creado gracias a una sugerencia de una de las colaboradoras de la Escuela Nº 200. El modo Botonera presenta más complejidad que el Barrido en términos de usabilidad. El mismo consiste en seis botones: cuatro de dirección, *click* izquierdo y *click* derecho. El funcionamiento en este modo es bastante sencillo: mientras el botón está presionado, el cursor se mueve en la dirección correspondiente. Se soporta que se presione más de un botón en simultáneo, para permitir movimientos en diagonal o gestos complejos como selección múltiple o *click&drag*.

2.2.3. Cabeceo

Originalmente concebido pensando en controlar el cursor del *mouse* con la cabeza, aunque puede ser utilizado también sujetando el dispositivo a un *joystick* o sistema análogo. En este modo de uso el cursor se controla modificando la orientación espacial del dispositivo. Dado que la orientación de un cuerpo en el espacio es un problema de interés en la aeronáutica, se utilizan los nombres típicos de estas variables: *yaw* (variando entre $[-180°, 180°]$), *pitch* (variando entre $[-90°, 90°]$) y *roll* (variando entre $[-180°, 180°]$). Las mismas son utilizadas para describir la posición de un cuerpo en el espacio como se aprecia en la figura 2.4. Teniendo esto en cuenta, se define que el *pitch* y *roll* controlan el movimiento del *mouse* en el eje vertical y horizontal respectivamente.

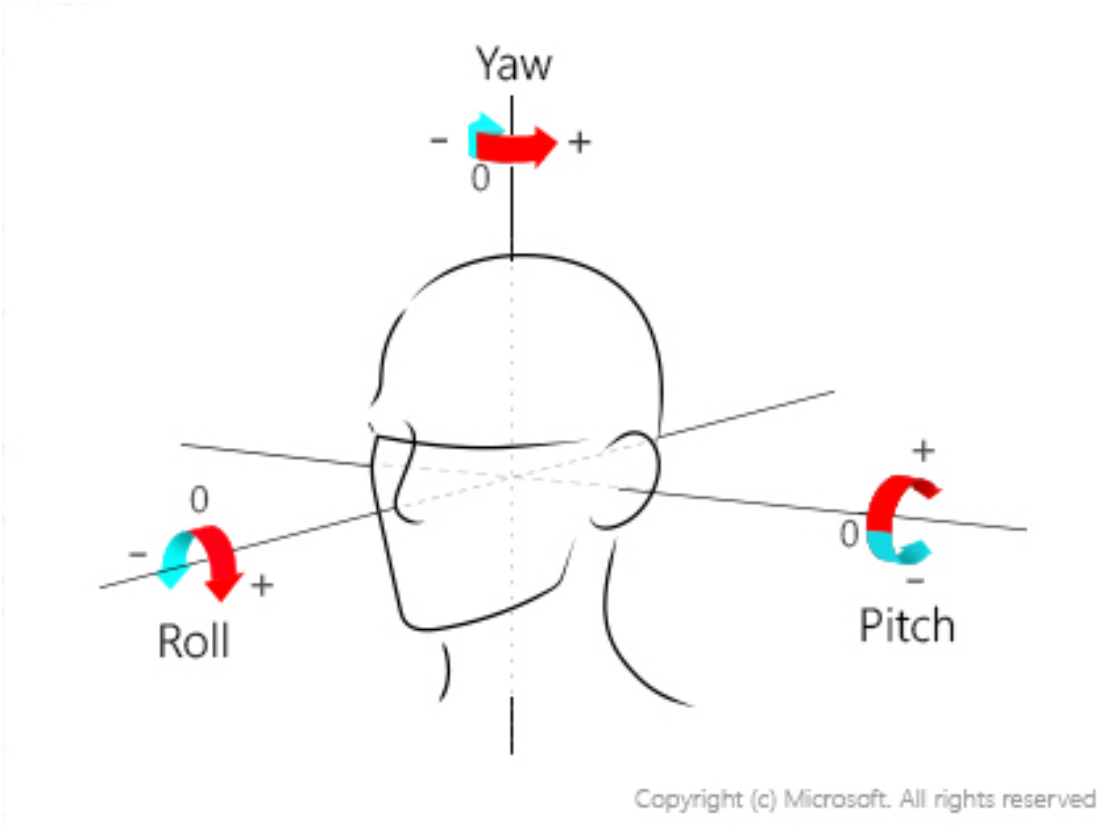

Figura 2.4: Ejes principales de rotación de un cuerpo en el espacio

2.2.4. Gestos

Este es el modo que más se asemeja en uso a un *mouse* convencional. El mismo está diseñado para ser utilizado con el bloque principal (sin botones) fijado a la muñeca (u otra parte del cuerpo), y permite traducir gestos a movimientos del cursor en una analogía directa al *mouse*, con la salvedad de que el movimiento se mantiene incluso cuando el gesto terminó. Para detener el movimiento del cursor es necesario realizar un gesto en particular. Esto permite controlar el cursor con movimientos menos precisos que con la solución convencional.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 3

Diseño de *Software*: Manejo de Sensores

Según lo descrito en la Sección 2.2, tanto el modo Cabeceo como Gestos basan su funcionamiento en detectar de forma correcta los movimientos del usuario, para lograr esto se decide usar un sensor capaz detectar la aceleración del dispositivo (acelerómetro), y de esta forma obtener información de su movimiento lineal. Por otro lado un giroscopio también es necesario, ya que el mismo brinda información sobre el movimiento angular del dispositivo. Podría pensarse que usando estos dos sensores es posible caracterizar completamente los movimientos que se realizan con el dispositivo, sin embargo resulta necesario también un magnetómetro para determinar la orientación del mismo.

A lo largo de este capítulo se busca introducir al lector en todo lo que respecta a los sensores utilizados en este proyecto. En el mismo se explican las configuraciones usadas en cada uno de los sensores, así como también las calibraciones realizadas a los mismos. También se detallan las señales obtenidas de cada uno de estos sensores, cuya relevancia quedará de manifiesto en el Capítulo 4.

3.1. Configuraciones de los Sensores

Los sensores seleccionados para este proyecto son: $LSM6DSOX$ (acelerómetro y giroscopio) y $LIS3MDL$ (magnetómetro); el proceso de selección y características de los mismos se detallará en la sección 5.1.2. Estos sensores admiten distintas configuraciones; a continuación se detalla cuales se eligieron, de forma que se pueda detectar los movimientos realizados por los usuarios, así como la orientación del dispositivo.

3.1.1. Acelerómetro

En esta sección se explican las configuraciones con las que funciona el acelerómetro. El objetivo del mismo es detectar los movimientos que realiza el usuario, por lo cual es necesario conocer las características de los gestos de interés. Los movimientos humanos están limitados en términos de aceleración y frecuencia; aunque haya personas más rápidas que otras se puede estimar un promedio de aceleración máxima y una frecuencia máxima (o duración mínima) de los movimientos. Este dato es sumamente relevante para configurar correctamente el acelerómetro, de forma tal que el mismo ignore el ruido

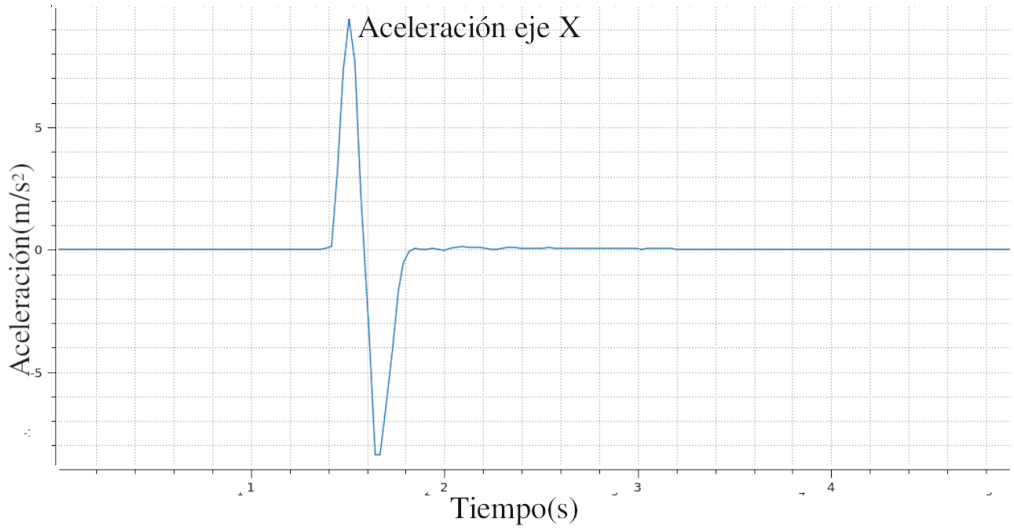

Figura 3.1: Aceleración en el eje X en un movimiento sobre una superficie plana

que provenga de frecuencias más altas.

Con la finalidad de estimar esta frecuencia típica se analiza la señal proveniente del acelerómetro al realizar un movimiento con el mismo. Para esto se obtiene una gráfica de la aceleración en función del tiempo al realizar un movimiento considerado rápido para las condiciones de uso previstas. Este movimiento se realiza partiendo del reposo, en la dirección positiva del eje X, deslizando el sensor sobre una superficie plana y volviendo al reposo. En la figura 3.1 se presenta la información obtenida para el movimiento descrito. Se observa que la duración del mismo es de 400 ms, derivando en una frecuencia de 2,5 Hz. Considerando que esta frecuencia no necesariamente es la máxima posible, se configura la frecuencia de corte del filtro pasa bajos implementado en el acelerómetro en 5 Hz.

Otra característica del acelerómetro es la posibilidad de configurar el fondo de escala del mismo. Considerando una vez más el movimiento típico mencionado, se puede notar un pico de aceleración de alrededor de 9,8 m/s^2 ($\approx g$), por lo que con argumento análogo al utilizado al fijar la frecuencia de corte, se fija el fondo de escala en $\pm 2g$.

A continuación, se recapitula la configuración del acelerómetro junto con la frecuencia de muestreo del mismo, dado que resulta de interés pese a no ser un parámetro configurado.

- Rango: $\pm 2g$, siendo g el valor de la aceleración gravitatoria.
- Filtro pasa bajos con frecuencia de corte en 5 Hz.
- Frecuencia de muestreo: 104 Hz.

3.1.2. Giroscopio

Los giroscopios dan una medida de la velocidad angular a la que está sometido el dispositivo; dado que en nuestro caso de uso se necesita la posición angular ($roll$, $pitch$, y

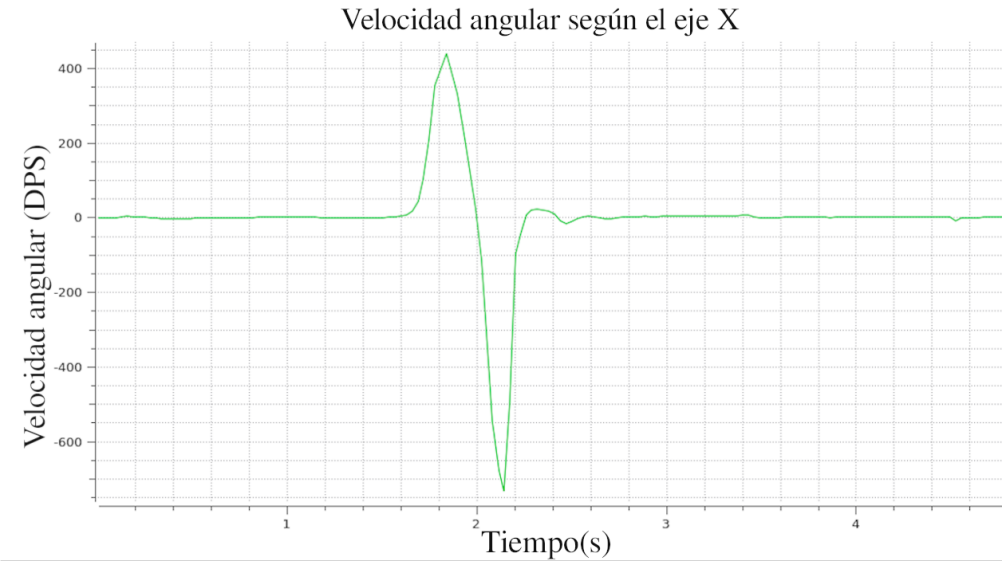

Figura 3.2: Velocidad angular según eje x para una rotación rápida.

yaw), a todos los efectos prácticos se trabaja con la integral de lo que mide el sensor. Esto presenta un problema: la medida de velocidad obtenida tiene una componente de *offset* constante y otra que varía en el tiempo que no puede ser compensada de manera trivial. Es por esto que al obtener la posición angular el error se va acumulando lentamente. Debido a que el *offset* variable es un problema que presentan los giroscopios en general, y considerando que el principal uso de estos sensores es para conocer la posición angular más que su velocidad, este es un problema ampliamente tratado y analizado [6]. Una mitigación a este problema consiste en aplicar un filtro pasa alto a las medidas obtenidas del sensor que, si bien no soluciona completamente, ayuda a disminuir su efecto. En el giroscopio con el que se trabaja esto es posible hacerlo por *hardware*, pudiéndose incluso configurar la frecuencia del filtro pasa alto (de forma análoga al acelerómetro). A efectos de minimizar el error acumulado este filtro se configuró para la frecuencia de corte más baja disponible: 16 mHz.

De forma similar al acelerómetro, en el giroscopio es posible configurar el fondo de escala en el que se desea trabajar en grados por segundo (DPS). A efectos de tener un criterio razonable para este valor se procede a obtener medidas del sensor para el caso de un giro brusco, obteniéndose los datos de la figura 3.2. En base a lo observado (picos de aproximadamente 700 DPS) y los fondos de escala disponibles, se determina usar como fondo de escala ± 1000 DPS

De esta forma las características definidas para el giroscopio (junto con la frecuencia de muestreo) son:

- Rango: ± 1000 DPS.
- Filtro pasa altos con frecuencia de corte en 16 mHz.
- Frecuencia de muestreo: 104 Hz.

3.1.3. Magnetómetro

Este sensor se utiliza para determinar la orientación del dispositivo con mayor precisión [6], como se detallará en la Subsección 3.3.3. La incorporación de este sensor en el cálculo de la posición angular brinda una solución al problema generado por el *offset* variable en el tiempo del giroscopio, mencionado en la Subsección 3.1.2.

Teniendo en cuenta que lo que se busca medir con este sensor son variaciones del campo magnético de la tierra (entre 25 y 65 μT) se fija el rango máximo en el menor posible, el cual es $\pm 400\ \mu T$. Para los otros parámetros a configurar no se poseen restricciones impuestas por el uso humano, y a priori no se tiene ninguna hipótesis razonable sobre especificaciones que deba cumplir. Es por esto que para estas configuraciones se utilizarán los valores por defecto, las cuales se listan a continuación:

- Rango: $\pm 400\ \mu T$.
- Filtro pasa bajos de frecuencia de corte: $5\ Hz$.
- Frecuencia de muestreo: $1000\ Hz$.

3.2. Calibraciones

Cuando se trabaja con sensores y se pretende cierta precisión, en general es necesario realizar una calibración [5]. En algunos casos, esta calibración se reduce simplemente a restar valores de *offset*, y en otros casos es necesario recurrir a funciones un poco más complejas. En esta sección se explican los procedimientos necesarios para calibrar cada uno de los sensores, y de esta forma lograr su correcto funcionamiento.

3.2.1. Acelerómetro

Previo a adentrarnos en la calibración es necesario realizar un breve análisis de los valores brindados por el sensor, para así justificar la necesidad o no de realizar dicho ajuste. A la hora de estudiar los datos vertidos por un acelerómetro, uno de los principales resulta ser el módulo de la aceleración. Este se calcula como

$$|\vec{A}| = \sqrt{a_x^2 + a_y^2 + a_z^2}. \tag{3.1}$$

Cuando el sensor se encuentra en reposo la única aceleración presente será la gravitatoria, por lo que es de esperar que el módulo de $\vec{A}$ se mantenga constante en un valor cercano a ésta (aproximadamente 9,8 m/s^2).

Para analizar más en detalle esto se realizaron algunas gráficas donde se muestra justamente el modulo del vector $\vec{A}$ para algunas posiciones significativas. Por ejemplo, en la figura 3.3 se puede ver el caso en el que el sensor se encuentra en posición horizontal, de forma tal que el eje Z queda orientado hacia arriba (es decir, la aceleración g se descompone casi enteramente en el eje Z con un valor positivo).

Como se puede apreciar, los valores obtenidos en la figura 3.3 difieren bastante de lo esperado ya que la aceleración en reposo están en el entorno de los 10,14 m/s^2. En contraposición, en la figura 3.4 podemos ver el resultado cuando el sensor se gira 180°

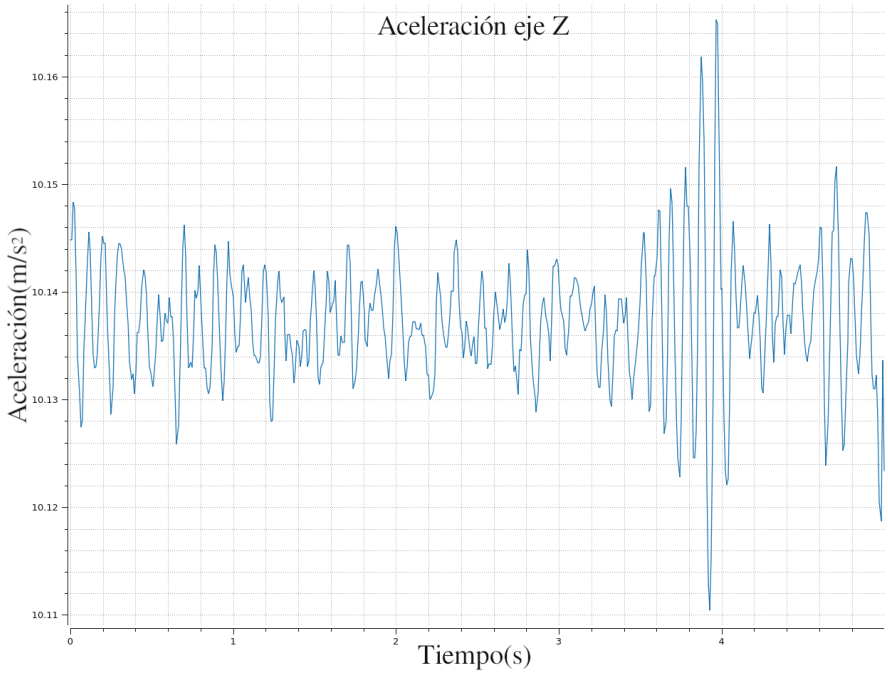

Figura 3.3: Aceleración gravitatoria en el eje Z positivo pre calibración

entorno al eje X (o Y, es indiferente) dejándolo en posición horizontal manteniendo el eje Z apuntando hacia abajo. En este caso vemos que el valor es también bastante malo, aunque la descompensación se aprecia hacia el lado opuesto que en el ejemplo anterior, ya que la aceleración se encuentra en el entorno de los 9,52 m/s^2.

Como último ejemplo, podemos apreciar la figura 3.5 en la que se observan los datos recogidos para el sensor posicionado de manera tal que el eje X queda apuntando hacia arriba, generando que la aceleración gravitatoria esté presente casi enteramente en dicho eje y con valor positivo. En este caso, la aceleración está en el entorno de lo esperado.

Considerando el rango de aceleraciones que se quieren medir ($\pm 2g$) y el uso que se le dará a estas señales (lo cuál se explicará en detalle en las Secciones 3.4 y 4.5), las gráficas obtenidas (principalmente las 3.3 y 3.4) resultan bastante elocuentes y son prueba suficiente de la necesidad de realizar una calibración.

Para obtener valores mas coherentes y fieles a la realidad, se procede a realizar una calibración de seis puntos [11]. Esta consiste en medir la aceleración en seis posiciones determinadas, y con base en ellas obtener otros seis valores que serán usados para ajustar los datos obtenidos directamente del acelerómetro.

Inicialmente, los valores a medir serán dos por cada eje y corresponden a las posiciones “boca-arriba” y “boca-abajo” en cada uno de ellos. Es decir, tomando como ejemplo el eje Z, colocamos el sensor de forma que ese eje quede paralelo a la gravedad con los valores de aceleración creciendo hacia arriba (como se ve en la imagen 3.6) y registramos esa medida como $a_{z_{+g}}$. Luego, se realiza un giro de 180° con respecto al eje X o Y y al valor obtenido lo denominaremos $a_{z_{-g}}$. Notar que estos dos valores deben tener un módulo similar, pero su signo debe ser opuesto. Extrapolando esta misma idea a los

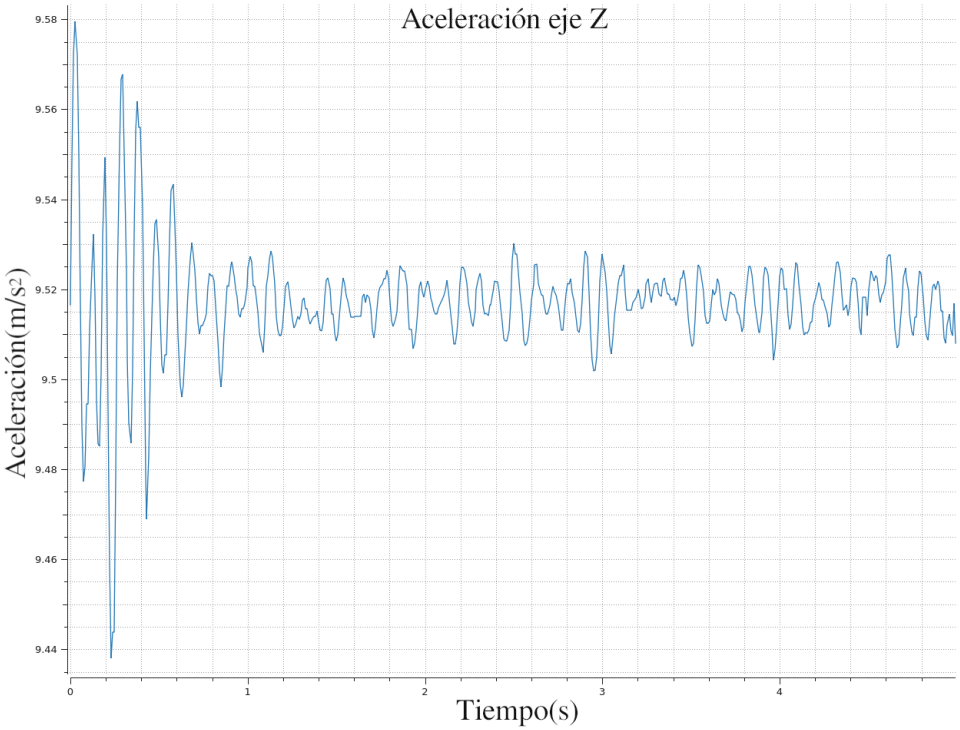

Figura 3.4: Aceleración gravitatoria en el eje Z negativo pre calibración

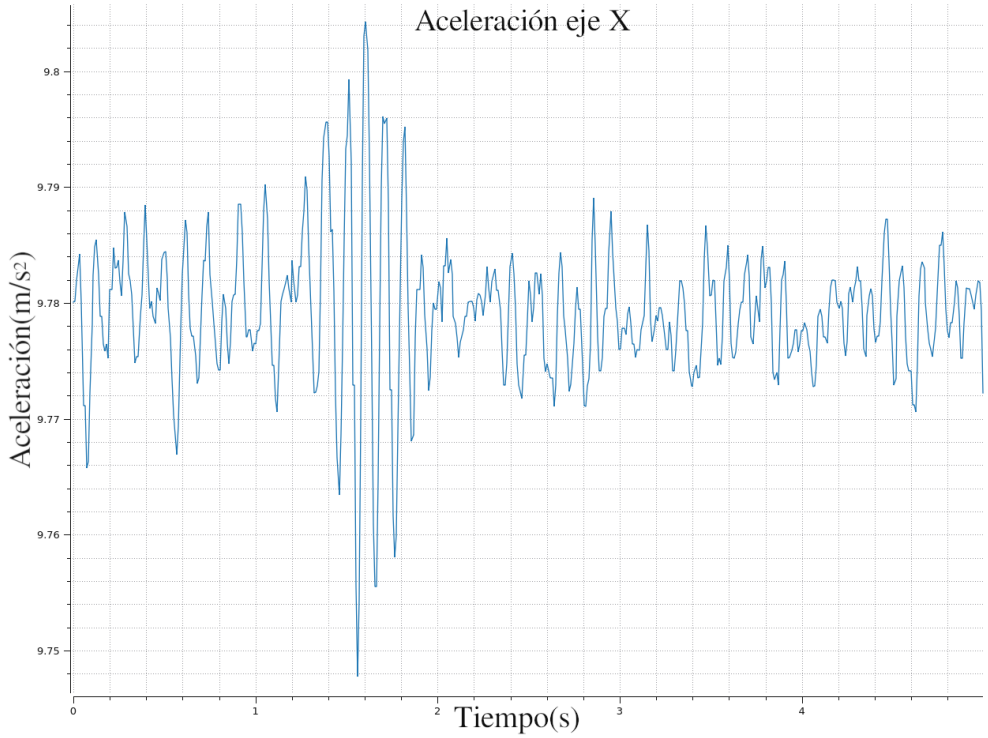

Figura 3.5: Aceleración gravitatoria en el eje X negativo pre calibración

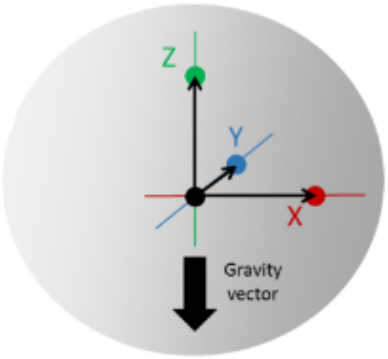

Figura 3.6: Posicionamiento del sensor

demás ejes, finalmente obtendremos dos vectores:

$$A_{+g} = \begin{bmatrix} a_{x_{+g}} & a_{y_{+g}} & a_{z_{+g}} \end{bmatrix}$$

$$A_{-g} = \begin{bmatrix} a_{x_{-g}} & a_{y_{-g}} & a_{z_{-g}} \end{bmatrix}.$$

Una vez que se tienen esos datos se debe calcular el promedio entre las medidas correspondientes a cada eje (denominado como m_i, con $i = \{x, y, z\}$) y la sensibilidad δ_i, dadas por las ecuaciones 3.2 y 3.3 respectivamente.

$$m_i = \frac{a_{i_{+g}} + a_{i_{-g}}}{2} \tag{3.2}$$

$$\delta_i = \frac{a_{i_{+g}} - a_{i_{-g}}}{2}. \tag{3.3}$$

Vale notar que para el caso ideal, $|a_{i_{+g}}| = |a_{i_{-g}}| = g$ (siendo $g \approx 9{,}8\ m/s^2$ la aceleración gravitatoria), lo que resulta en $m_i = 0\ m/s^2$ y $\delta_i = 9{,}8\ m/s^2$.

Realizando esto para cada eje, se obtienen dos nuevos vectores M y Δ

$$M = \begin{bmatrix} m_x & m_y & m_z \end{bmatrix}$$

$$\Delta = \begin{bmatrix} \delta_x & \delta_y & \delta_z \end{bmatrix}.$$

Teniendo esos seis valores, solo resta calcular las aceleraciones calibradas de la forma

$$a_{i_{calibrated}} = \frac{(a_{i_{raw}} - m_i)}{\delta_i} \times g \tag{3.4}$$

siendo $a_{i_{raw}}$ el valor crudo de aceleración y $a_{i_{calibrated}}$ el valor de aceleración luego de la calibración.

Luego de realizar una calibración utilizando este método se obtienen resultados más que satisfactorios. A modo de ejemplo, en la figura 3.7 podemos ver uno de los casos antes expuestos (3.4). Los valores utilizados para este ejemplo son:

$$A_{+g} = \begin{bmatrix} 9{,}83\ m/s^2 & 9{,}86\ m/s^2 & 10{,}07\ m/s^2 \end{bmatrix}$$

$$A_{-g} = \begin{bmatrix} -9{,}73\ m/s^2 & -9{,}92\ m/s^2 & -9{,}59\ m/s^2 \end{bmatrix}$$

lo que se traduce en dos vectores

$$M = \begin{bmatrix} 0{,}050\ m/s^2 & -0{,}030\ m/s^2 & 0{,}240\ m/s^2 \end{bmatrix}$$

$$\Delta = \begin{bmatrix} 9{,}780\ m/s^2 & 9{,}890\ m/s^2 & 9{,}830\ m/s^2 \end{bmatrix}.$$

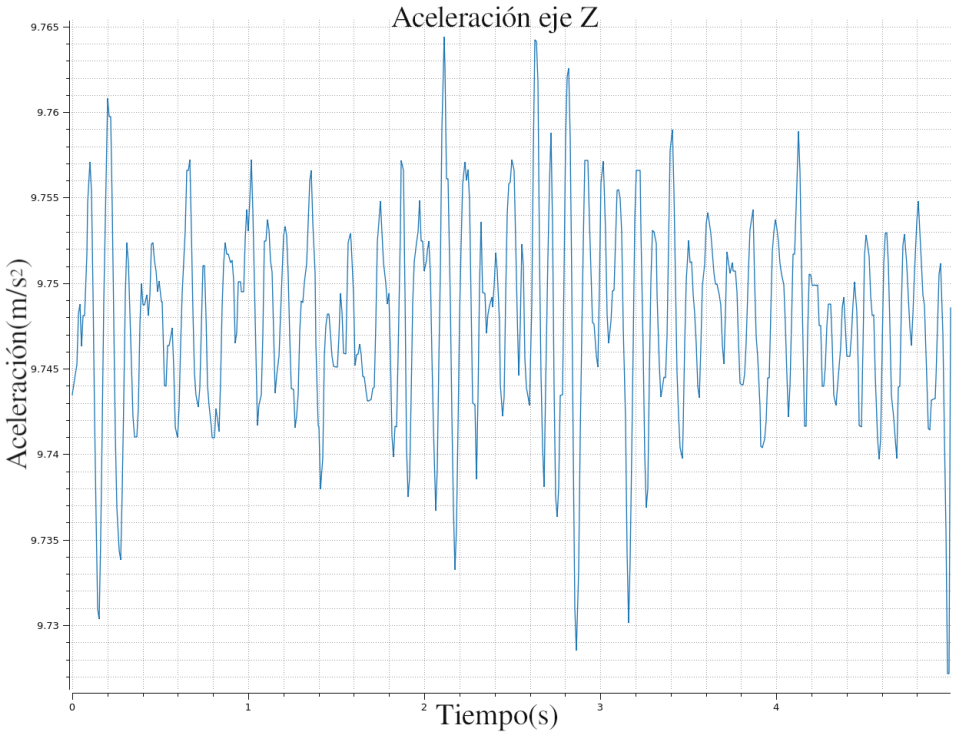

Figura 3.7: Aceleración gravitatoria en Z negativo post calibración

Como se puede apreciar, los resultados obtenidos están mucho más cerca del valor esperado, posicionándose la señal en un entorno de 9,8 m/s^2.

Vale realizar la aclaración que los valores mostrados son para un ejemplo y sensor particular; en la mayoría de los casos es necesario realizar este proceso para cada sensor que se utilice. Teniendo en cuenta que durante el proceso de desarrollo del presente proyecto no fue necesario recalibrar el acelerómetro, se considera que basta con realizar la calibración una única vez.

3.2.2. Giroscopio

Para el caso del giroscopio, como fue mencionado en la Subsección 3.1.2 las medidas tienen presente un *offset* variable sumado a una componente constante. Este último

genera que, cuando el sensor se encuentra en reposo se está midiendo un valor (levemente) distinto de cero, como se ve en la figura 3.8. Para atacar este problema se resta el valor medio medido en reposo, para obtener medidas en el entorno de cero.

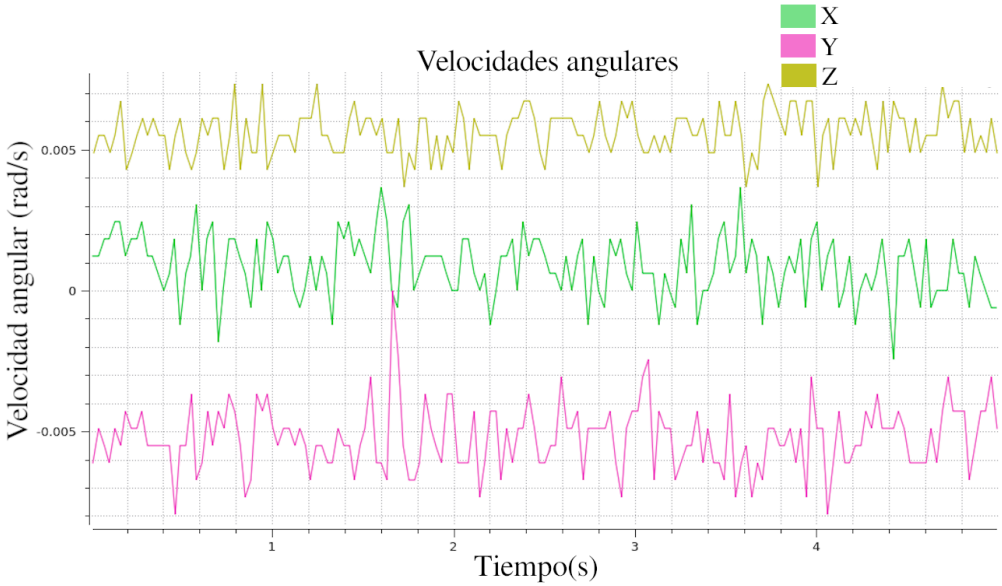

Figura 3.8: Datos del giroscopio sin calibración

El resultado de esto se puede observar en la figura 3.9, en donde los valores medidos están mucho más próximos al cero.

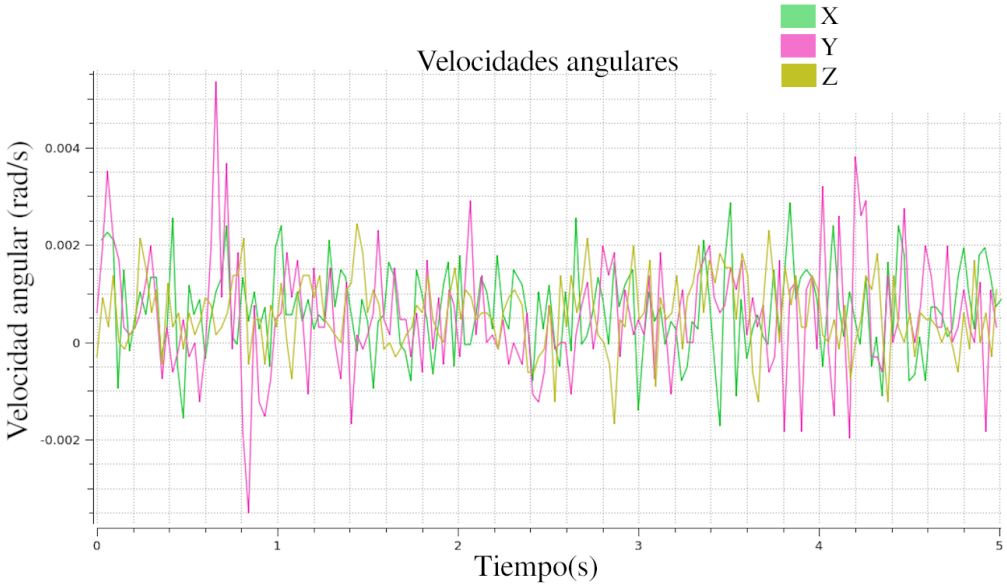

Figura 3.9: Datos del giroscopio calibrados

3.2.3. Magnetómetro

Para el caso del magnetómetro, la calibración no es tan trivial [4] como en los casos anteriores. Por eso, para esto se utiliza un *software* brindado por *Adafruit* llamado **Motion Sensor Calibration Tool** (figura 3.10), también denominado *MotionCal*. Dicho programa se complementa con un *sketch* disponible en la librería de *Arduino IDE* que debe correr en el microcontrolador encargado de manejar el sensor.

Figura 3.10: Motion Sensor Calibration Tool

Una vez que se ha cargado el código mencionado y se ha seleccionado el puerto serial correspondiente, el programa comenzará la toma de datos y desplegará los datos necesarios para la calibración.

Principio de funcionamiento

Como se mencionó, la calibración del magnetómetro no resulta tan sencilla como en el caso del acelerómetro o el giroscopio. Esto se debe a que, debido a su naturaleza, es necesario recolectar datos en todas (o al menos una gran cantidad) de las orientaciones en las que el sensor puede posicionarse. De esta manera, una vez que el *MotionCal* está operativo, es necesario mover el sensor en todas las direcciones hasta que la nube de puntos forme una esfera. Una vez que se posean suficientes datos, el programa automáticamente centrará esa esfera en el origen de coordenadas, y desplegará en el panel de la derecha los valores necesarios para realizar la calibración que se explica en esta sección.

En la figura 3.11 podemos ver un ejemplo de la nube de puntos generada por el programa previo a la corrección. El caso que se aprecia es un tanto "artificial", ya que se colocó un imán permanente cerca del sensor con el fin de poder mostrar en esta sección las distintas posibilidades que existen. La realidad es que, si no hay presencia de metales o campos magnéticos considerables, la esfera formada por la nube de puntos

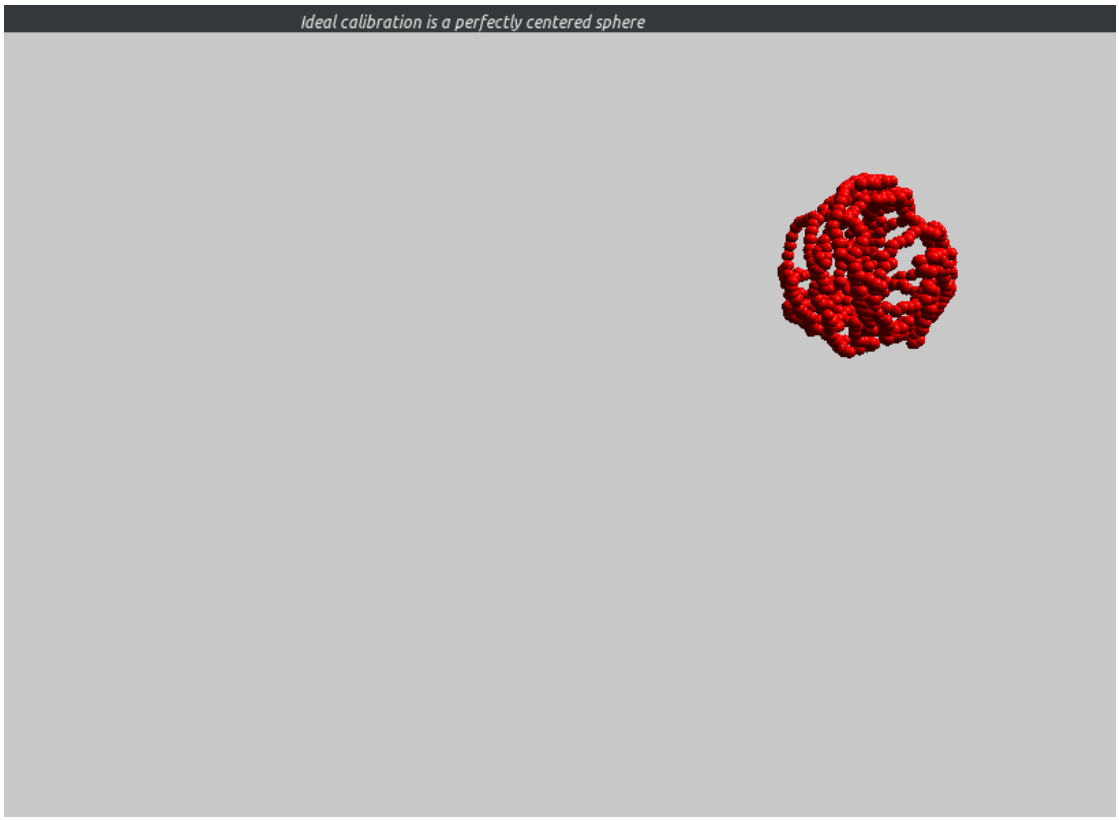

Figura 3.11: Magnetómetro sin calibrar - *MotionCal*

estará naturalmente centrada en el origen (figura 3.12), haciendo que no sea necesaria una calibración.

Para una correcta calibración del magnetómetro se requieren doce valores:

- Un vector de tres valores correspondiente al *offset* (también denominado calibración *hard iron*) y que responde a las distorsiones creadas por objetos que producen un campo magnético.
- Una matriz cuadrada de 3×3 de mapeo (calibración *soft iron*) denominada S. La misma se debe a las variaciones o distorsiones del campo magnético existente.

El vector de *offset* brinda una calibración inicial, que tiene en cuenta los grandes campos magnéticos que afectan al sensor de manera más o menos homogénea. Por ende, podemos obtener una primera aproximación a la solución realizando la operación simple que se ve en la ecuación 3.5

$$Mg_{cal} = \begin{bmatrix} mg_x \\ mg_y \\ mg_z \end{bmatrix} = \begin{bmatrix} mg_{x_{\mathbf{raw}}} \\ mg_{y_{\mathbf{raw}}} \\ mg_{z_{\mathbf{raw}}} \end{bmatrix} - \begin{bmatrix} hi_{\mathbf{x}} \\ hi_{\mathbf{y}} \\ hi_{\mathbf{z}} \end{bmatrix} \tag{3.5}$$

con $mg_{j_{\mathbf{raw}}}$ los datos adquiridos crudos del magnetómetro y $hi_{\mathbf{j}}$ los valores de *offset*.

Por su parte, también existen variaciones del campo magnético presente, que puede variar dependiendo de la dirección. Es por esto que la matriz de mapeo S busca arreglar

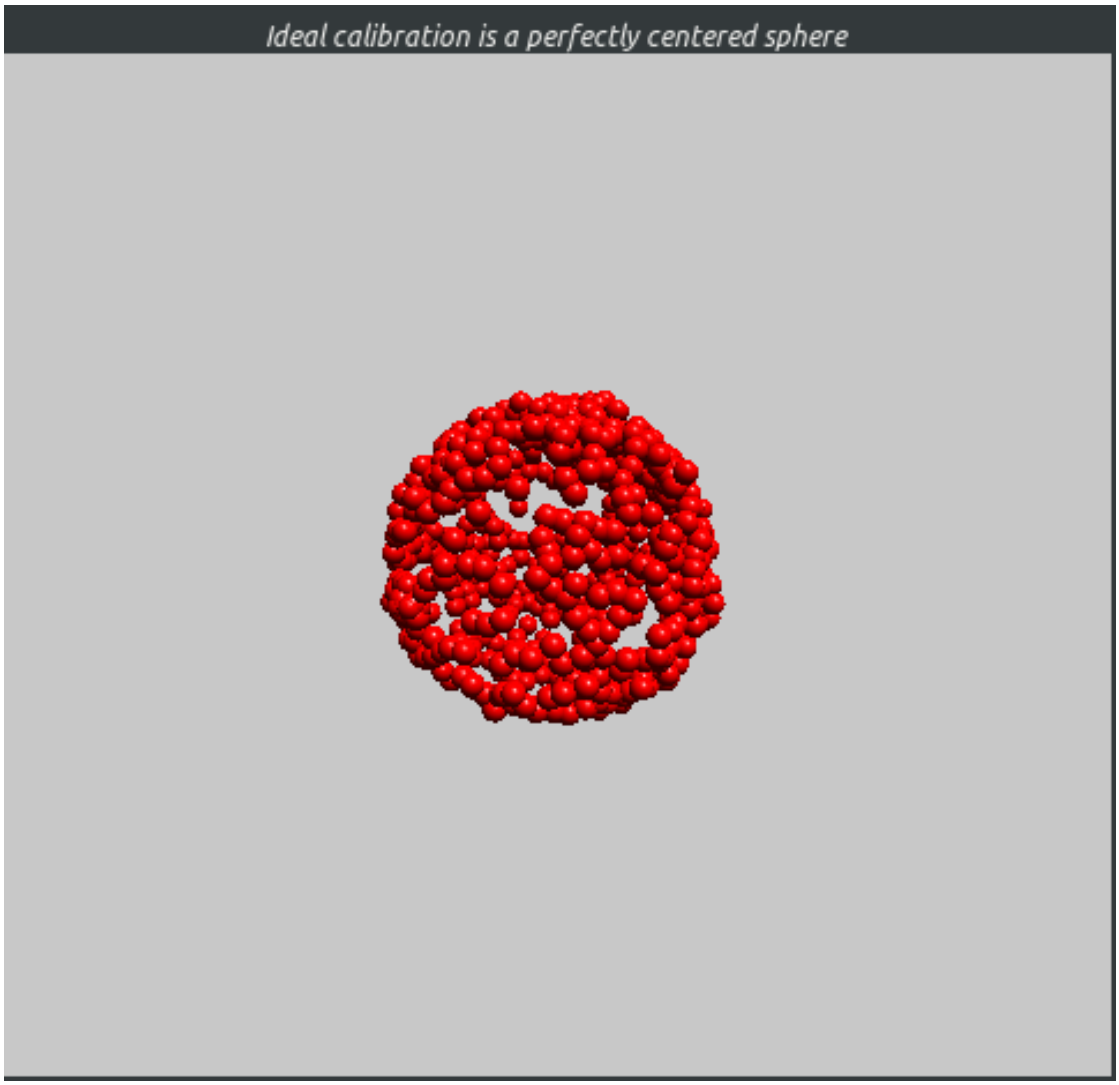

Figura 3.12: Magnetómetro calibrado - MotionCal

las medidas de un eje, apoyándose en las medidas adquiridas en los otros dos. Por ende, cada entrada de esta matriz es un factor de correlación entre ejes.

$$S = \begin{bmatrix} si_{xx} & si_{xy} & si_{xz} \\ si_{yx} & si_{yy} & si_{yz} \\ si_{zx} & si_{zy} & si_{zz} \end{bmatrix}. \tag{3.6}$$

Una vez que se tiene esta matriz, solo resta realizar una multiplicación matricial para obtener los valores calibrados del magnetómetro

$$M_{calibrated} = \begin{bmatrix} si_{xx} & si_{xy} & si_{xz} \\ si_{yx} & si_{yy} & si_{yz} \\ si_{zx} & si_{zy} & si_{zz} \end{bmatrix} \times \begin{bmatrix} mg_x \\ mg_y \\ mg_z \end{bmatrix}. \tag{3.7}$$

Vale notar que en un caso ideal esta matriz sería la identidad, ya que el campo magnético en cada eje depende pura y exclusivamente de la medida realizada, por lo que no existe una correlación con los otros ejes.

Por último, uniendo las expresiones 3.5 y 3.7, podemos obtener una fórmula final para la calibración de la forma:

$$M_{calibrated} = \begin{bmatrix} si_{xx} & si_{xy} & si_{xz} \\ si_{yx} & si_{yy} & si_{yz} \\ si_{zx} & si_{zy} & si_{zz} \end{bmatrix} \times \begin{bmatrix} mg_{x_{\mathbf{raw}}} - hi_{\mathbf{x}} \\ mg_{y_{\mathbf{raw}}} - hi_{\mathbf{y}} \\ mg_{z_{\mathbf{raw}}} - hi_{\mathbf{z}} \end{bmatrix}. \tag{3.8}$$

En el caso particular del magnetómetro, teóricamente debería realizarse una calibración para cada espacio en el que se vaya a utilizar, ya que existe una variabilidad propia del ambiente que lo rodea que puede generar alteraciones en las medidas. El dispositivo fue calibrado al comenzar el desarrollo de *software*, y a lo largo del proyecto se utilizó el dispositivo en diversos ambientes y no se notó que fuera necesario recalibrar este sensor. Sin embargo es importante notar que dependiendo de las condiciones del lugar de uso esto puede llegar a ser necesario.

3.3. Orientación

Como se comentó anteriormente en este capítulo, conocer la orientación del dispositivo es necesario para determinar por completo el movimiento del mismo. Se entiende por "orientación" la posición angular de un objeto; en este caso y como se expresó en la Subsección 2.2.3 utilizamos coordenadas en ángulos de navegación. Para calcularla se pueden utilizar diferentes métodos aprovechando los distintos sensores del *MARG* (sigla proveniente de *Magnetic, Angular Rate, and Gravity*), o más aún se puede unir la información que brinda cada uno de ellos para tener una orientación más precisa. En esta sección se explican tres métodos para la obtención de la orientación del dispositivo.

3.3.1. Únicamente Acelerómetro

El acelerómetro es un sensor que mide la aceleración (lineal+gravitatoria) en los tres ejes cartesianos. Teniendo la aceleración en cada uno de estos ejes y bajo la hipótesis de que la aceleración lineal es cero, se puede usar este sensor para medir los ángulos de alejamiento con respecto a la vertical (*roll* y *pitch*). Sin embargo este sensor no permite calcular el ángulo *yaw*, ya que la aceleración gravitatoria es colineal con el eje de rotación del mismo. Otro problema que presenta este método para obtener la orientación es que el acelerómetro debe estar en ausencia de aceleración lineal para que funcione. Esto implica que cualquier aceleración lineal en el dispositivo genere un error en el cálculo de la orientación, ya que este sensor no puede separar $\vec{g}$ de una aceleración lineal impuesta. Por esta razón no es el mejor método para usar cuando el acelerómetro va a estar en movimiento.

Para mostrar esta problemática se realiza la siguiente prueba: se coloca el sensor sobre una mesa con el eje Z colineal con la aceleración gravitatoria, y se procede a realizar una rotación de cero a noventa grados en el ángulo *roll*, para luego volver a la posición inicial. Seguido a esto se mueve el sensor sobre la mesa en la dirección del eje X.

En la figura 3.13 se muestra una gráfica del ángulo *roll* obtenido en función del tiempo al realizar la prueba antes descrita. La primera cresta de la gráfica corresponde a la rotación de noventa grados realizada al comienzo de la prueba; este resultado se corresponde con el experimento realizado. Sin embargo los siguientes picos que se observan en la gráfica muestran dos rotaciones ficticias, que se generan al realizar un movimiento lineal con el sensor. Esto muestra que no es viable usar únicamente un acelerómetro para obtener la orientación de un objeto si el mismo está sometido a aceleraciones lineales.

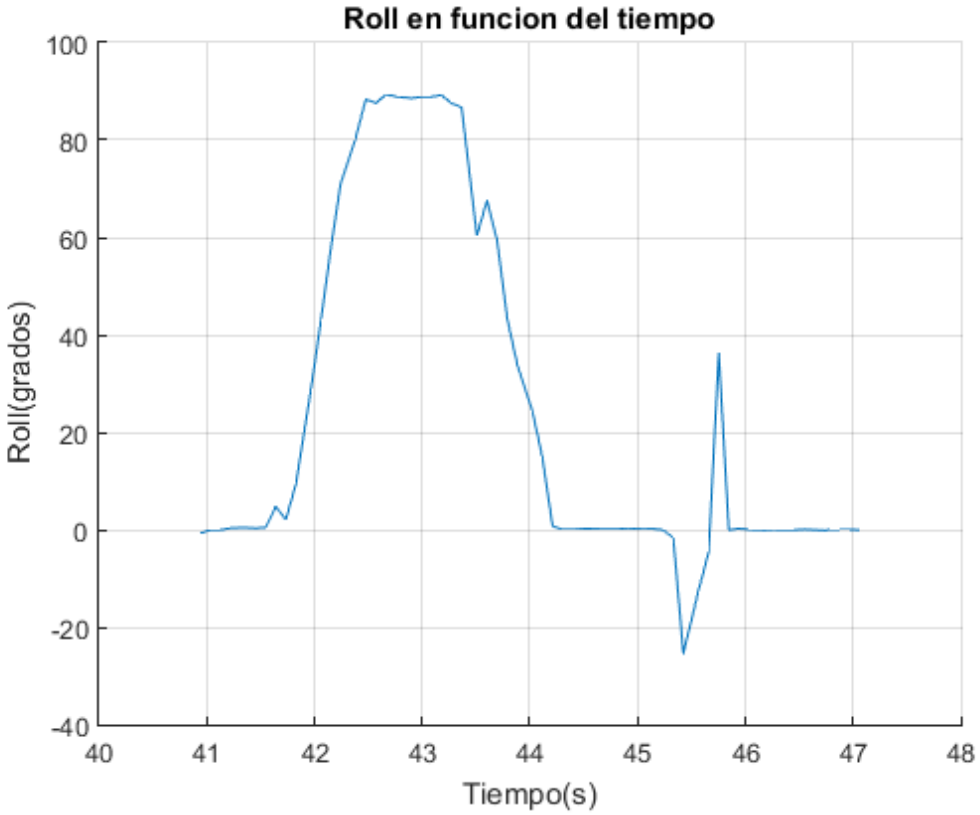

Figura 3.13: Rotación en *roll* en función del tiempo, obtenido mediante el acelerómetro

3.3.2. Únicamente Giroscopio

El sensor giroscopio brinda la velocidad angular del dispositivo. Al integrar la velocidad angular en el tiempo se puede obtener la posición angular. Sin embargo este método tiene un problema que se origina por el *drift* propio que tiene el giroscopio, como fue mencionado en la Subsección 3.2.2. Esto genera que la posición angular acumule error y no se logre repetibilidad. En la gráfica 3.14 se muestra una rotación de cero a noventa grados en el ángulo *roll* con varios ciclos. Como se puede observar en esta gráfica aunque se realice la misma rotación varias veces, el error acumulado impide que se llegue a los noventa grados después del primer ciclo.

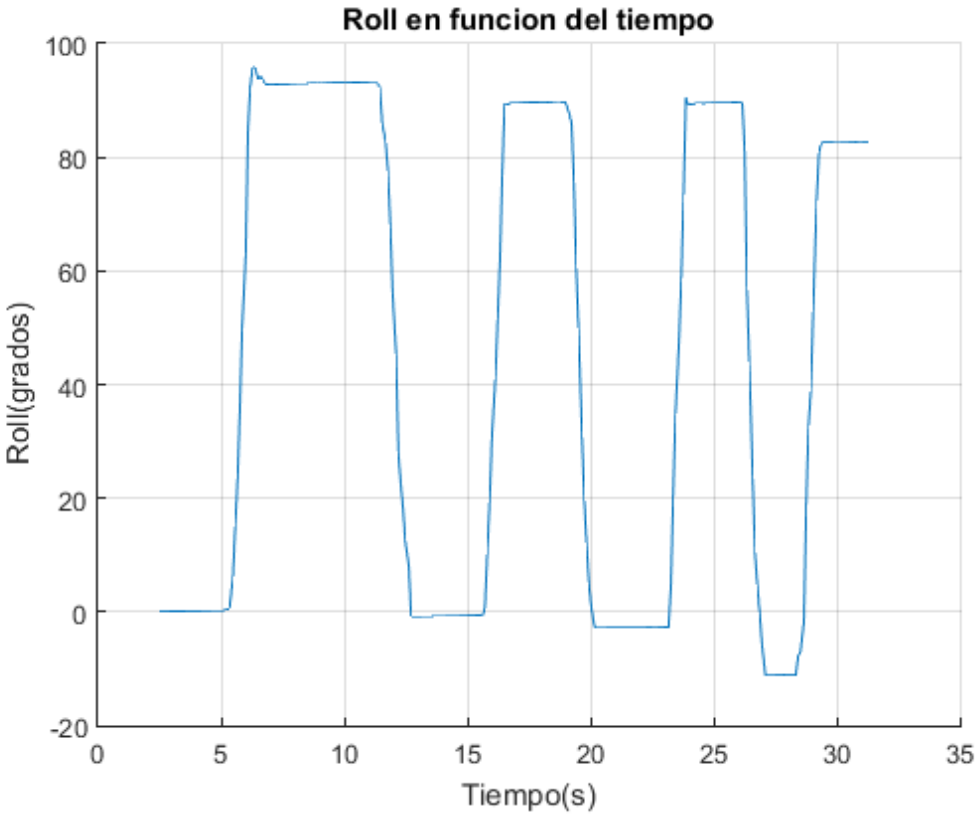

Figura 3.14: Rotación en *roll* en función del tiempo, obtenido mediante el giroscopio

3.3.3. Acelerómetro, Giroscopio, y Magnetómetro

Como fue mencionado a lo largo de esta sección, utilizando el giroscopio y el acelerómetro es posible obtener la posición angular del dispositivo, que resulta necesaria para determinar completamente el movimiento del mismo. No obstante, los valores obtenidos no poseen la precisión necesaria para la aplicación. Es por esto que surge la necesidad de integrar el magnetómetro; éste permite corregir algunos de los problemas introducidos por los otros sensores (tales como el *drift* introducido por el giroscopio) [6], permitiendo así obtener datos de orientación más precisos.

La implementación de la fusión de los sensores no es para nada una tarea trivial, y excede los objetivos particulares de este proyecto. Por esto se usa una implementación desarrollada por *NXP Semiconductors* [31]. Este algoritmo se basa en el filtro de *Kalman* [50], muy usado para predecir variables de estado no observables de un sistema dinámico. Su principio se sustenta en el uso de variables de estado observables tomando conocido el tipo de errores que presentan para predecir las no observables. En este caso en particular las variables observables son los datos obtenidos del acelerómetro, giroscopio, y magnetómetro, que brindan la orientación del dispositivo pero con muchos errores. En la figura 3.15 se observa el diagrama de bloques del algoritmo en cuestión.

En la figura 3.16 se presenta la orientación obtenida mediante este algoritmo al rotar el dispositivo 90°. Notar que en este caso el *roll* vuelve al valor original luego de variar

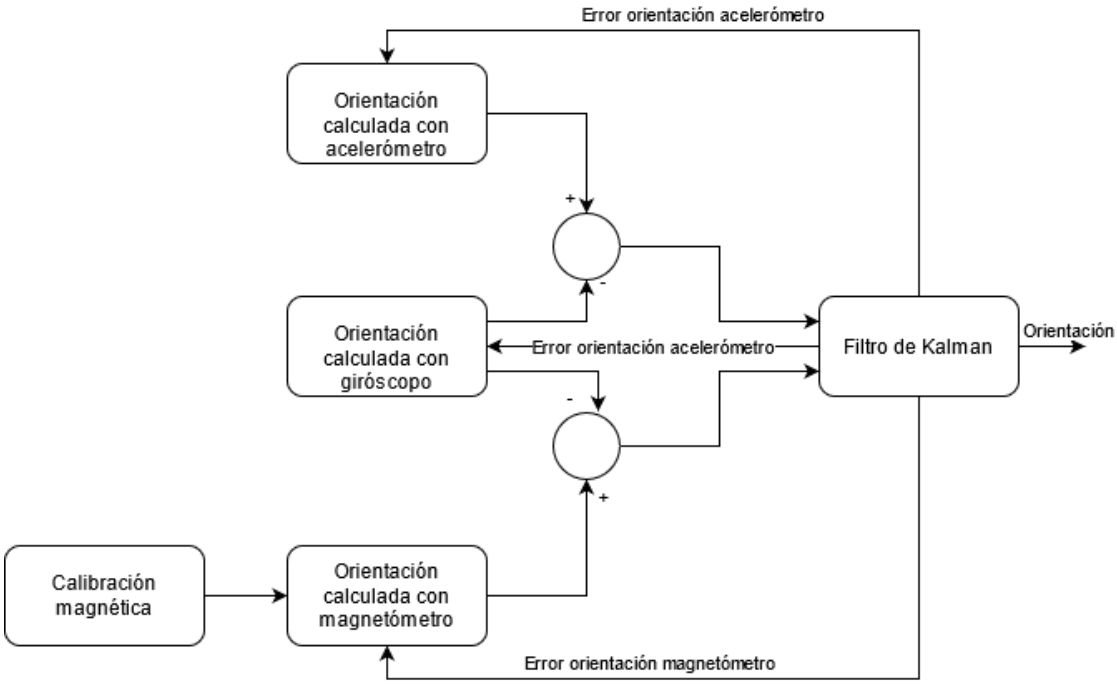

Figura 3.15: Diagrama del algoritmo desarrollado por N*XP Semiconductors*

la orientación, a diferencia de lo observado en la gráfica 3.14.

Para el caso de la obtención de la posición angular *yaw* el resultado no es el esperado. A diferencia de los casos de *roll* y *pitch*, la señal obtenida no se condice con la realidad; en la figura 3.17 podemos ver graficado el *yaw* para el caso en el que el dispositivo se mantiene en reposo. Como se puede apreciar, la señal no se mantiene estable como debería. No obstante, esta problemática no será un impedimento a la hora de decidir utilizar este algoritmo. Esto es debido a que se puede evitar el uso del *yaw*, y además el *roll* y el *pitch* demostraron poseer una fidelidad adecuada. Ya que esto resulta suficiente para resolver los modos propuestos (*i.e.* Cabeceo y Gestos), se decide utilizar este algoritmo sin considerar la información brindada sobre el ángulo *yaw*.

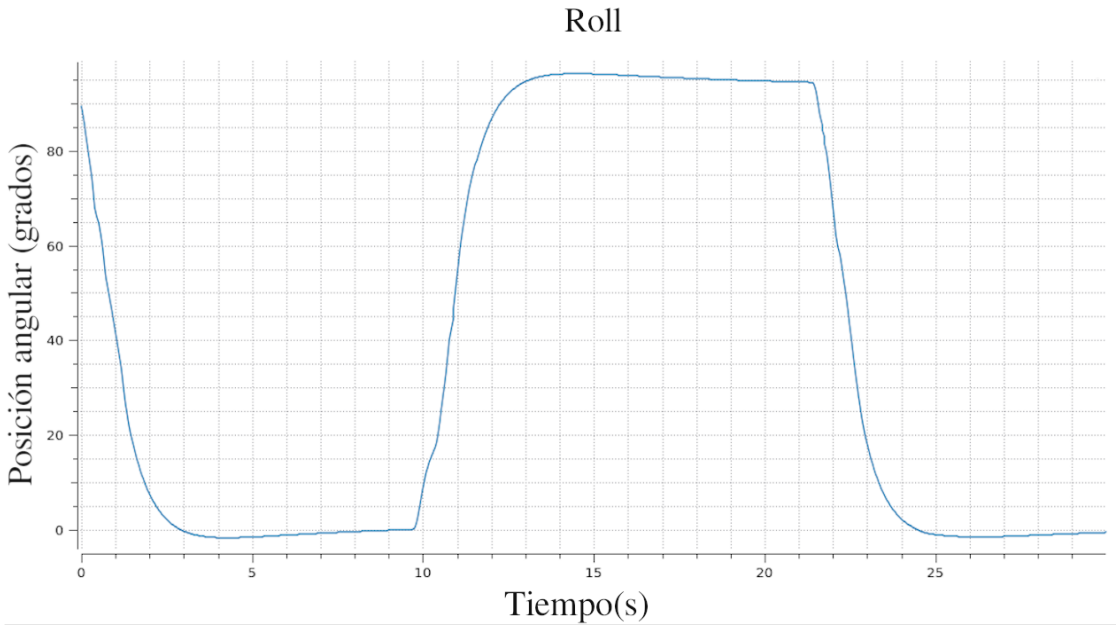

Figura 3.16: Rotación en $roll$ en función del tiempo

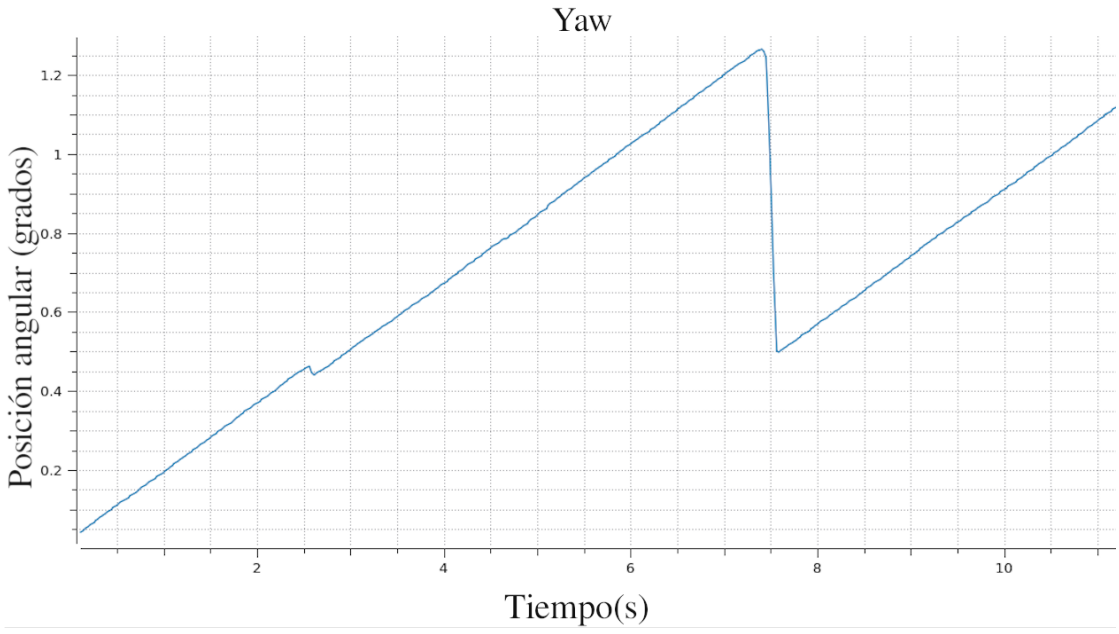

Figura 3.17: Yaw en función del tiempo

3.4. Señales de Interés

Una vez que se tienen configurados y calibrados los sensores, es hora de hablar de las señales que estos proveerán al sistema (lo cual se detallará en el Capítulo 4). Algunas de las señales serán tomadas tal cual provienen de los sensores, realizando únicamente el proceso de calibración ya expuesto. En otros casos será necesario recurrir a tratamientos previos, integrando información de todos los sensores para la obtención de distintas señales que serán de utilidad.

Velocidades angulares

Una de las señales que serán utilizadas a lo largo del tratamiento que se detallará más adelante son las velocidades angulares. Estas señales son provenientes directamente del giroscopio, pasando únicamente por el proceso de calibración visto en la Subsección 3.2.2.

A modo de ejemplo, en la figura 3.18 podemos ver el caso de un movimiento angular en el eje Z. En dicha imagen se representa la velocidad angular para el caso en que se rota el sensor variando el ángulo *yaw* 90° y se retorna a la posición inicial.

En términos generales, los datos brindados por este sensor son suficientemente buenos para la aplicación particular, por lo que no será necesario realizarle ningún tratamiento adicional.

Posiciones angulares

Otra señal que resultará de interés es la que se deriva de la Sección 3.3; para las implementaciones y procesamientos venideros será necesario conocer la posición angular del dispositivo en el espacio. Como se explicó en la Subsección 3.3.3, para esto se utilizarán las medidas angulares de *roll* y *pitch* obtenidas del algoritmo de orientación desarrollado por *NXP Semiconductors*. En la figura 3.19 vemos el caso de una rotación de 90° entorno al eje X, variando así el *roll*.

Este algoritmo obtiene una aproximación más que confiable a la orientación del dispositivo en el espacio. Uno de los problemas que surge a la hora de analizar las señales referidas a la posición angular, es que las mismas poseen un retardo en cuanto a su reacción y convergencia. Es decir, al realizar una rotación la señal comenzará a moverse acompañando el movimiento; una vez que el dispositivo retome su reposo, la curva de posición angular demorará cierto tiempo en converger al valor final alcanzado. Una primera pista sobre esto se observa en la figura 3.19, donde se ve que la señal sube hasta cierto punto y posteriormente comienza un leve descenso hasta el nivel deseado ($\approx 90°$). Este problema se hace mucho más evidente en la figura 3.20, donde vemos el caso en el que el dispositivo se rota 180° entorno al eje Y. Como primer observación, vale notar que nunca se alcanzan los 90°, pese a que este valor fue el máximo en el movimiento descrito. Por otro lado, la línea roja allí marcada representa el instante real en el que el dispositivo retomó su estado de reposo. Se aprecia claramente que, una vez que se dejó quieto el dispositivo, la señal de posición angular demora un tiempo más que considerable (en el entorno de los 5 *seg*) en estabilizarse.

No obstante, los beneficios presentados por la utilización de esta señal superan con creces a las desventajas expuestas, por lo cual se deberán tener en cuenta y lidiar con

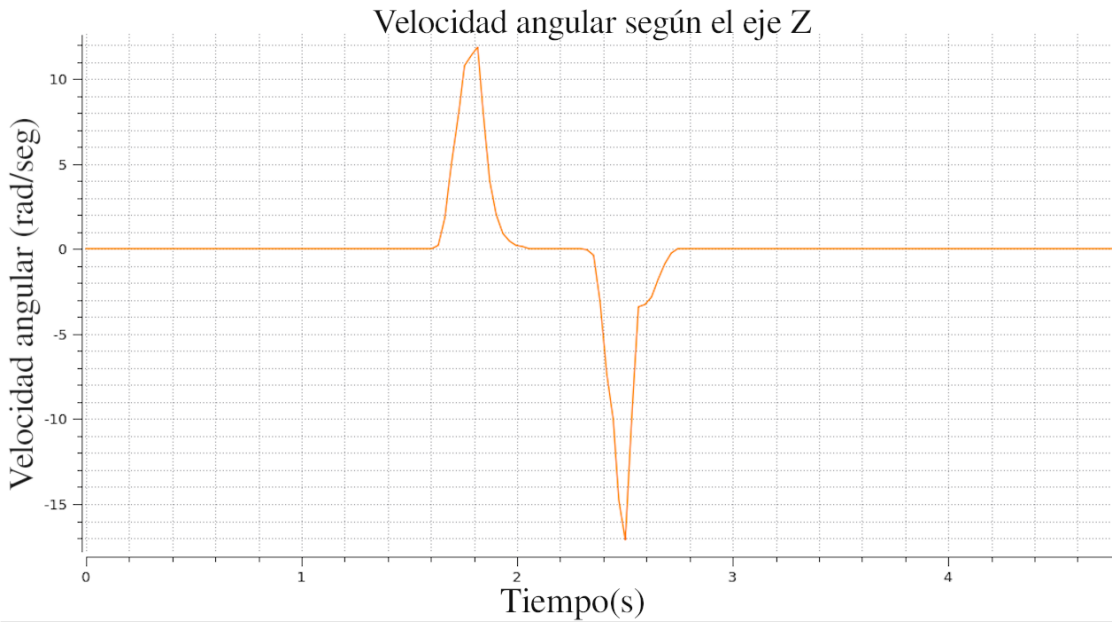

Figura 3.18: Velocidad angular - variación entorno al eje Z

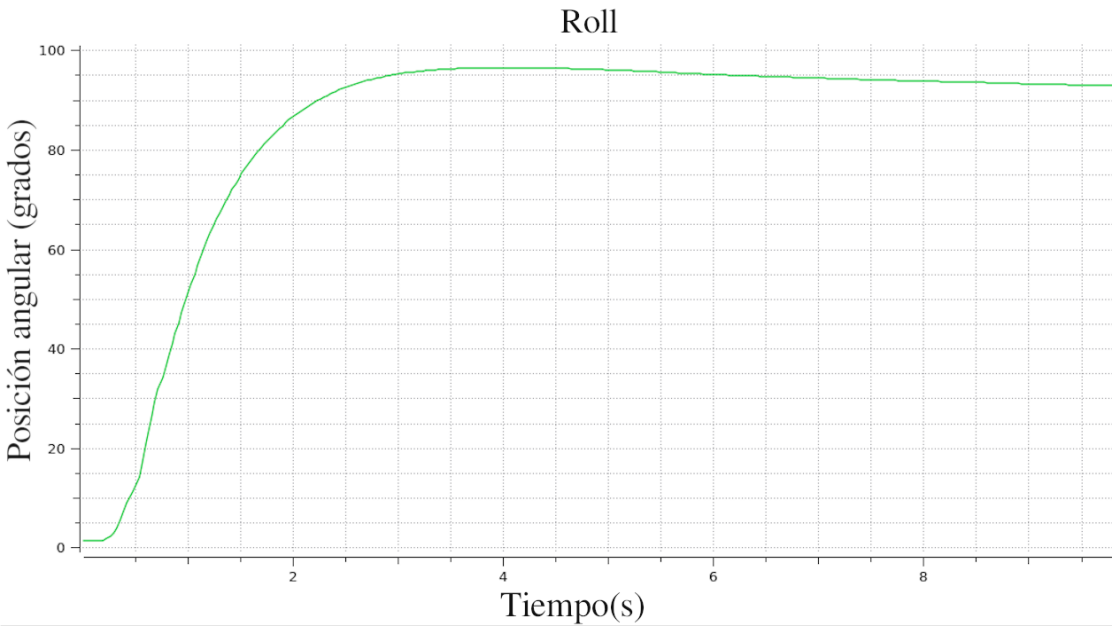

Figura 3.19: Posición angular - rotación de $90°$ entorno al eje X

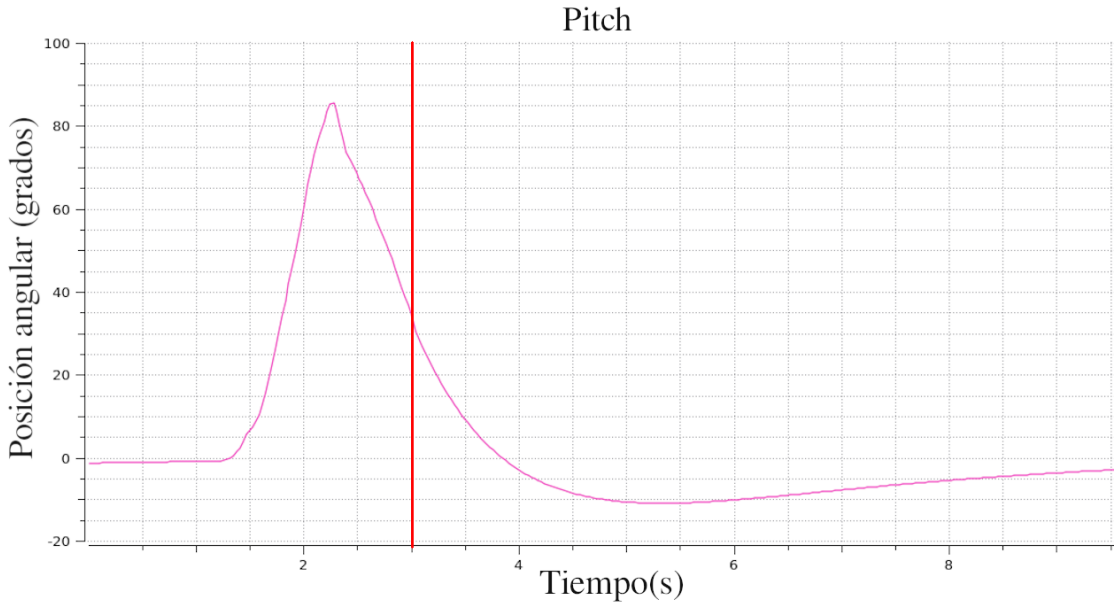

Figura 3.20: Posición angular - rotación de $180°$ entorno al eje Y

ellas cuando sea pertinente. En el Capítulo 4 se detallará cómo se utiliza la orientación del dispositivo para el funcionamiento del modo Gestos y el modo Cabeceo.

Aceleración

Otra de las señales esenciales para el desarrollo de los distintos modos de uso es la aceleración. Inicialmente se toman los valores brindados por el acelerómetro y se les realiza el pre-procesamiento correspondiente a la calibración (como se explicó en la Subsección 3.2.1). Una vez realizado esto, se tienen las aceleraciones respectivas a cada eje prontas para ser analizadas y procesadas. Esta señal será utilizada fuertemente para la detección de movimientos lineales; en la figura 3.21 podemos ver cual es su comportamiento para el caso en el que justamente se realiza un movimiento de traslación en la dirección positiva del eje X, con el dispositivo apoyado sobre una superficie plana.

Estas medidas provenientes directamente del acelerómetro tienen embebida la componente de la aceleración gravitatoria ($g = |\vec{g}| \approx 9{,}8\ m/s^2$) correspondiente a cada eje. Como se verá mas adelante (Sección 4.5) la detección de gestos se realizará utilizando la velocidad; para obtener esta magnitud con cierta precisión estas componentes de g resultan ser una complicación. A priori, las componentes de la aceleración gravitatoria (por más pequeñas que sean) serán indistinguibles de las generadas por un gesto de un usuario, por lo cual es necesario eliminarla de nuestras medidas. Para esto se utiliza la fusión de sensores; teniendo la orientación en el espacio del dispositivo en todo momento, se puede obtener la descomposición de g en cada eje y restarlo a cada una de las medidas brindadas por el acelerómetro.

Para calcular las componentes de g correspondientes a cada eje se utilizarán las

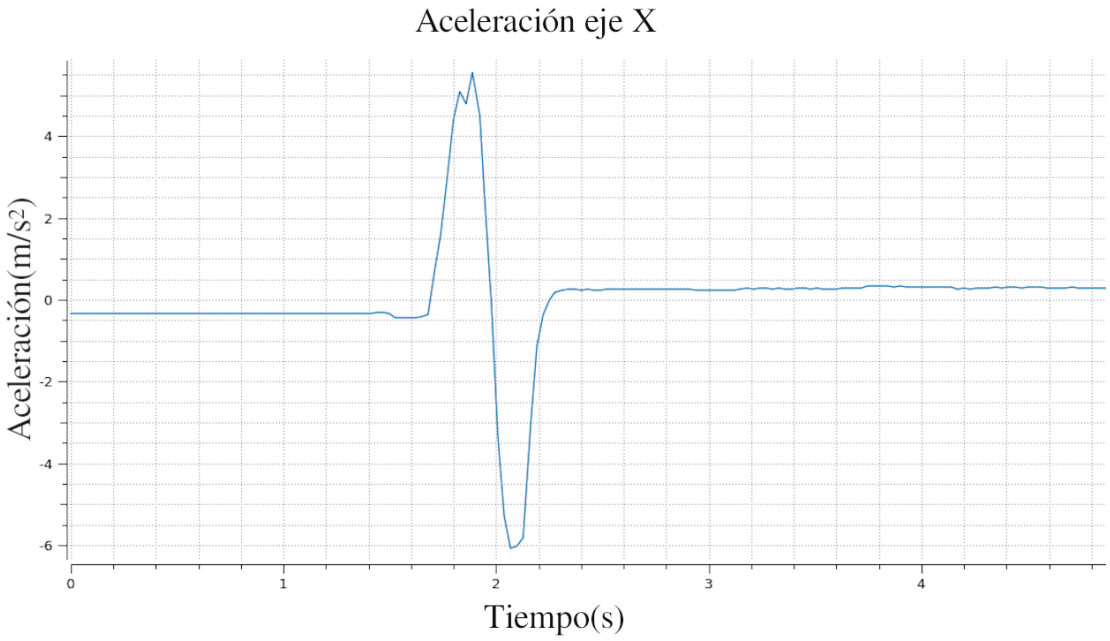

Figura 3.21: Aceleración lineal eje X - movimiento lineal

ecuaciones:

$$\begin{aligned} g_x &= -|g| \times sin(pitch) \\ g_y &= |g| \times cos(pitch).sin(roll) \\ g_z &= |g| \times cos(pitch).cos(roll) \end{aligned} \tag{3.9}$$

de forma tal que los valores *roll* y *pitch* corresponden a las posiciones angulares antes mencionadas.

Una vez que se tienen calculadas las componentes de la aceleración gravitatoria para cada uno de los ejes, solo falta restárselas a las aceleraciones provenientes del sensor:

$$\begin{aligned} accel_x &= accel_{x_{raw}} - g_x \\ accel_y &= accel_{y_{raw}} - g_y \\ accel_z &= accel_{z_{raw}} - g_z \end{aligned}$$

siendo $accel_i$ y $accel_{i_{raw}}$ (con $i \in \{x, y, z\}$) las aceleraciones procesadas y crudas respectivamente.

De este modo se consigue que, cuando el dispositivo está en reposo, las aceleraciones consideradas sean nulas.

En la figura 3.22 podemos ver el caso de la aceleración en el eje Z cuando el dispositivo está en reposo apoyado sobre una superficie plana. Como es de esperar, si observamos la aceleración cruda veremos que oscila entorno a un valor cercano a 10 m/s^2 (más precisamente, 9,8 m/s^2), mientras que la aceleración procesada se encuentra en valores cercanos a cero.

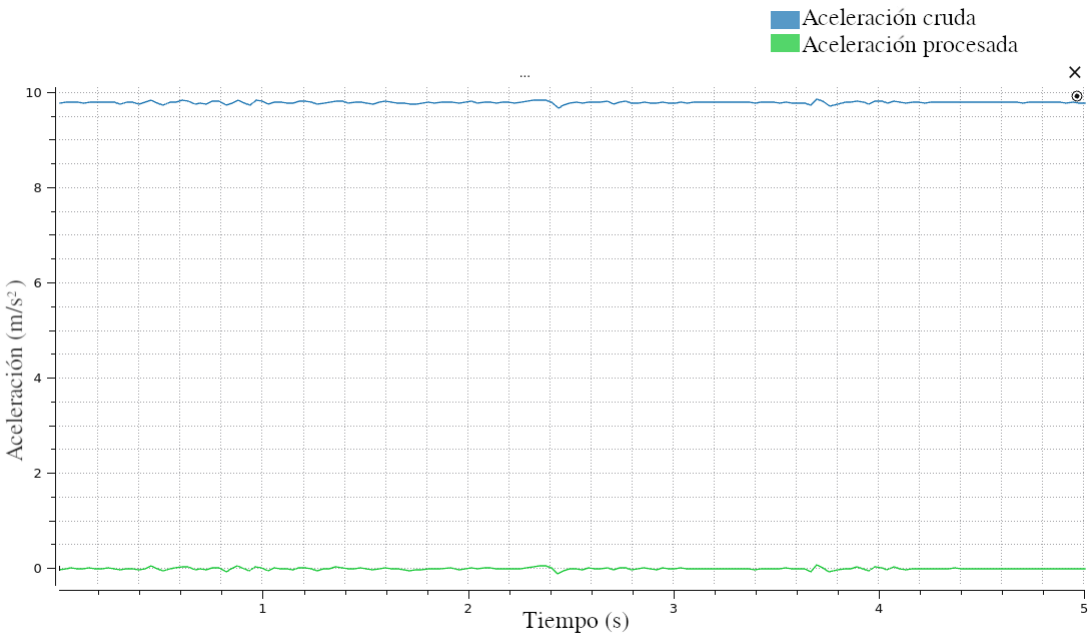

Figura 3.22: Aceleración cruda VS Aceleración procesada: eje Z

En la figura 3.23 podemos ver nuevamente el ejemplo de la aceleración en el eje X cuando se ejecuta un movimiento de traslación en dicho eje (en una superficie plana paralela al mismo), pero en este caso cotejada con la aceleración procesada; resulta claro que, una vez que la aceleración cruda se mantiene constante la procesada demora un determinado tiempo en converger a cero. Esto se debe a la problemática del algoritmo de orientación ya expuesta en esta sección, que genera que el restado de las componentes de $\vec{g}$ (ver ecuación 3.9) se retrase debido a la lenta convergencia de la posición angular (*roll* y *pitch*). Por esta razón las curvas mostradas en la figura 3.23 no son idénticas en su forma, y en el caso procesado se ve que la aceleración se mantiene por debajo del cero aún después de finalizado el movimiento. Esto deberá ser tenido en cuenta en los análisis venideros para así evitar caer en errores.

A partir de este punto, cuando se haga referencia a "aceleraciones" se estará hablando de las aceleraciones entorno al cero, es decir, las que se obtuvieron tras el restado de la componente de $\vec{g}$. Para los casos que se quiera referenciar a las aceleraciones provenientes directamente del acelerómetro, se aclarará de manera explícita denominándolas "aceleraciones crudas".

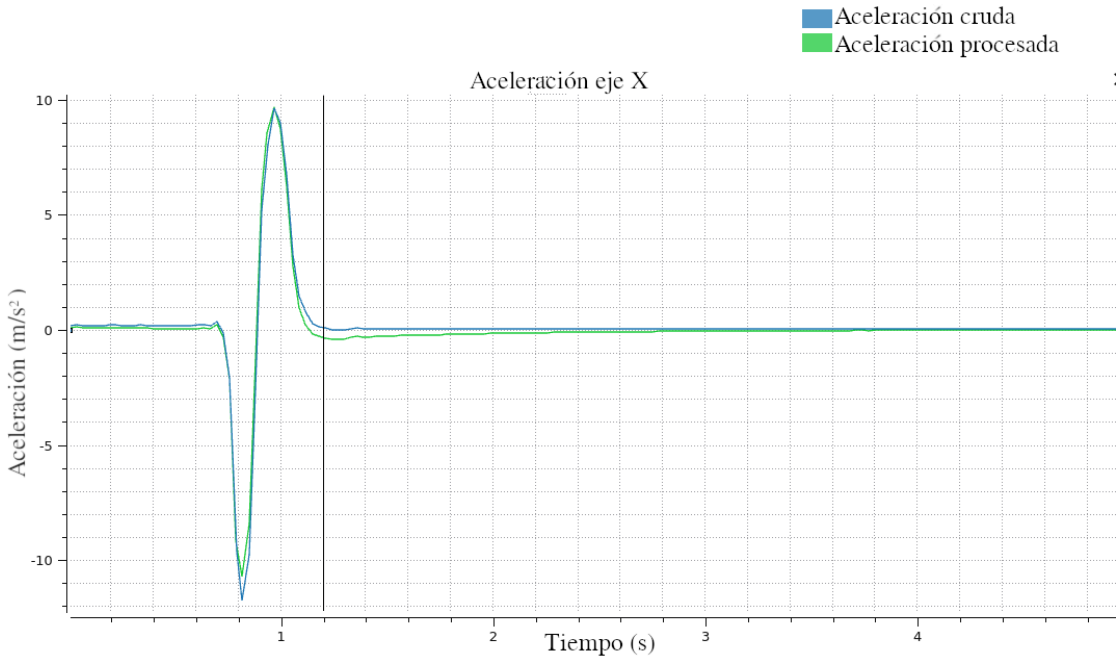

Figura 3.23: Aceleración cruda VS Aceleración procesada: eje X

Aplicación a los Modos

A lo largo de este capítulo se introdujeron los sensores que se utilizan en la implementación de los modos del dispositivo. Al comienzo se explicó con qué configuración se usa cada uno, para luego pasar a justificar la necesidad de calibrarlos, explicando brevemente de qué forma se realiza. A continuación se mencionó cómo se obtiene la orientación del dispositivo a través de los datos de tres sensores presentados. Por último se mencionaron cuales son las señales de interés que luego se utilizarán en la implementación de los modos del dispositivo.

Según lo que se presentó en la Sección 2.2 los modos de uso del dispositivo son: Barrido, Botonera, Cabeceo y Gestos. A continuación explicitan las señales que usa cada uno de ellos.

El modo Gestos se basa en detectar movimientos predefinidos del dispositivo; por esta razón necesita la mayor cantidad de señales. En particular utiliza: las aceleraciones crudas ($accel_{i_{raw}}$), las aceleraciones procesadas ($accel_i$), las velocidades angulares en los tres ejes, y por último la posición angular u orientación del dispositivo (*roll* y *pitch*).

Por otro lado el modo Cabeceo necesita únicamente dos magnitudes angulares relacionadas con la orientación del dispositivo, los ángulos *roll* y *pitch*, ya que el funcionamiento del mismo se basa en la inclinación del sensor. Dado que el ángulo *yaw* no resulta confiable se utiliza la velocidad angular respecto al eje Z para tomar decisiones respecto a las rotaciones en ese eje.

Por último los modos Botonera y Barrido no usan las señales de ninguno de estos sensores ya que su funcionamiento está basado en sensores capacitivos que son configurados por *hardware* y se presentarán en el Subsección 5.2.2.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 4

Diseño de *Software*: Arquitectura Principal

Habiendo introducido ya los distintos modos de uso del dispositivo en la Sección 2.2 (Barrido, Botonera, Cabeceo y Gestos), es momento de referirnos a la implementación particular de cada uno de ellos. En el desarrollo de este capítulo se desglosa la arquitectura de *software* principal, y se representa el funcionamiento de cada uno de los modos como bloques auto contenidos, es decir con inicialización y bucle principal.

4.1. Programa Principal

Dado que el dispositivo fue concebido para satisfacer necesidades de diversos usuarios, el desarrollo de los cuatro modos pierde valor si no se logra cierta flexibilidad para alternar entre ellos. Es por esto que el programa principal del proyecto aúna los modos previamente descritos, y provee un método para seleccionar el deseado. Este programa consiste en una máquina de estados con interrupciones, en la que cada estado se corresponde con un modo, y cuenta con un estado extra para seleccionar entre Barrido, Botonera, Cabeceo y Gestos. Para lograr esto, se utilizó el *encoder* incluido en la placa principal, girándolo para alternar entre los modos y utilizando el botón para habilitar o deshabilitar esta alternancia.

En la figura 4.1 se presenta el diagrama de flujo correspondiente al programa principal. En la configuración inicial del dispositivo, se realiza una lectura a la memoria, en la que se leen los datos correspondientes a la calibración del *MARG* y al estado inicial (último modo utilizado).

En el diagrama se puede observar que el programa tiene una estructura de máquina de estados, en la que cada estado representa un modo. Los diagramas correspondientes a estos modos serán presentados en las siguientes secciones con una salvedad: al ser presentados como programas auto contenidos, su estructura no coincide exactamente con la de un procedimiento tal como es invocado en este diagrama. Pese a esto, se toma este camino a efectos de facilitar la compresión de los modos por separado.

Como se puede notar en el diagrama, no existe ninguna condición explícita que genere el cambio de un modo a otro; esto se debe a que estos cambios se ejecutan a raíz de rutinas de atención a interrupciones (*ISR*). Estas rutinas están asociadas a los pines de giro y botón del *encoder* mencionados en la Sección 5.2.1.

El comportamiento de la *ISR* asociada al botón del *encoder* depende del estado del sistema. Al presionar el botón por primera vez se dispara una *ISR* que inicia un *timer*, el cual se detiene al soltar el botón. La acción que se realiza está dada por el tiempo que se mantenga presionado y por el modo en el que se encuentre el dispositivo. Si el modo es distinto de *SELECCION* y este tiempo es menor a 3 segundos se habilita la configuración de parámetros ($config\_parametros = 1$). Cada modo cuenta con un parámetro configurable; en el caso de Barrido, Botonera y Cabeceo será posible ajustar la velocidad del movimiento del cursor, mientras que en el modo Gestos se podrá configurar el umbral de detección. Por otro lado, si el tiempo que estuvo presionado el botón es mayor a 3 segundos, se entra en el modo *SELECCION*. En cualquiera de estos dos casos, una segunda presión del botón (independientemente del tiempo que esté presionado) aplica los cambios que se hayan generado y vuelve al estado inicial.

Las *ISR* asociadas a los giros del *encoder* también dependen del estado del sistema: si el botón no fue presionado previamente, los giros se ignoran. En caso contrario, se cuentan los pasos dados entre lecturas; estos pasos pueden ser positivos o negativos dependiendo de si el giro es en sentido horario o antihorario respectivamente.

De esta forma, estando en modo *SELECCION* los giros del *encoder* determinan que se modifique el modo pre-seleccionado, el cual se muestra mediante un movimiento del cursor característico para cada modo. La correspondencia entre el modo pre-seleccionado y el patrón del cursor puede consultarse en la Subsección D.4.1. Análogamente, si está habilitado el cambio de parámetros del modo, los giros modifican el valor definido. Luego de esto, la llamada al procedimiento correspondiente al modo ya contará con el valor actualizado, lo que permite evaluar en tiempo real si el valor del parámetro cumple con lo deseado o no.

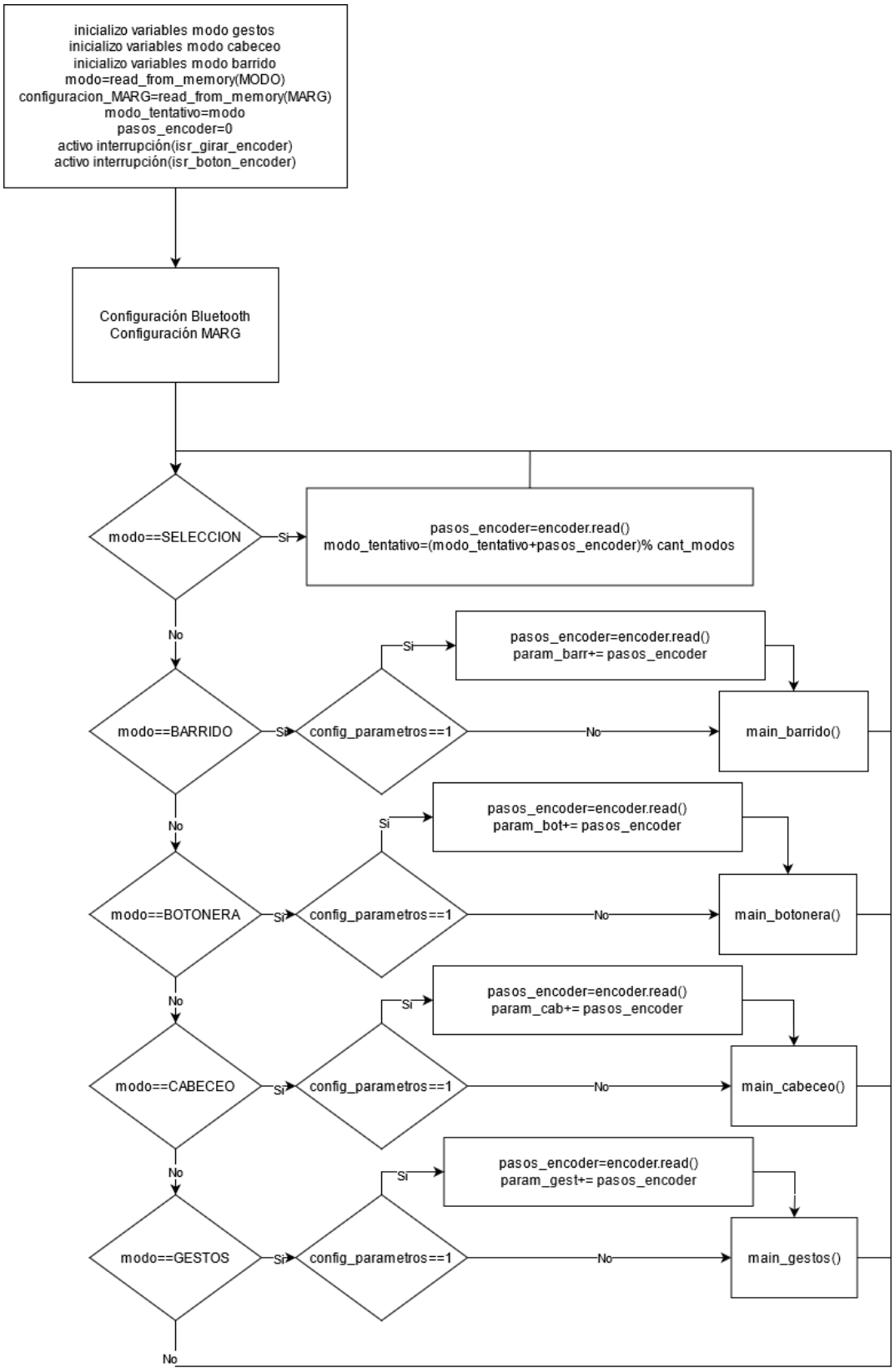

Figura 4.1: Diagrama de flujo del programa principal

4.2. Barrido

Dentro de las soluciones propuestas, el Barrido es la requiere menos interacción del usuario para controlar el cursor del *mouse*. Esto se debe a que en este modo el movimiento se da de forma semi autónoma recorriendo la pantalla de extremo a extremo (en el eje vertical o en el horizontal) hasta que el usuario realice una acción. El sistema cuenta con cuatro botones: uno corresponde al *click* principal del *mouse*, otro al *click* secundario, mientras que los dos restantes cambian el modo de desplazamiento de forma secuencial. El flujo de *software* para este modo se puede apreciar en la figura 4.2.

Como se puede notar en el diagrama al presionar el botón correspondiente al eje X (alternativamente Y), se desactiva el barrido en ese eje si estuviera activado y se activa en caso contrario. Con el barrido activado, el cursor del *mouse* se mueve entre el rango máximo y mínimo de la pantalla avanzando en cada iteración la cantidad $STEP\_X$ (alternativamente $STEP\_Y$). El valor $STEP$ merece un comentario en sí mismo ya que su relación con el desplazamiento en la pantalla no es trivial. Al mover el cursor del *mouse* con el microcontrolador, este movimiento se puede hacer de dos formas: enviándole al sistema operativo el desplazamiento del cursor, o la posición a la que se lo debe llevar. En el primer caso, la posición final del cursor dependerá de su ubicación previa y por lo tanto se define como movimiento relativo, mientras que en el segundo caso se define como movimiento absoluto. Esto profundiza en la Subsección C.1.5.

En el caso del modo Barrido el movimiento es absoluto, y la posición en determinado eje se puede expresar con un número entre 0 y 127. Por esto al incrementar *pos_cursor_y* en uno (es el valor de $STEP$ en este caso), el cursor se desplazará $\frac{Altura\ pantalla}{127}$ píxeles, donde *Altura pantalla* hace referencia a la cantidad de píxeles totales en el eje. Teniendo en cuenta que solo hacen falta 127 pasos para recorrer la pantalla de extremo a extremo en un eje, se implementa un contador de iteraciones para aumentar el tiempo entre envío y envío; cuando el contador llega a un valor predeterminado, se envía la nueva posición del cursor. Es por esto que el modo Barrido no realiza transferencias *Bluetooth* continuamente dentro del bucle, sino que lo hace sólo cuando el contador determina que se debe mover el cursor.

Para el manejo de los *click* (tanto principal como secundario) se detectan los flancos en los botones correspondientes; cuando la función *UP_Slope_Detected()* detecta que la señal correspondiente al botón en cuestión cambia de nivel bajo a alto, se envía la orden de realizar un *click*. Análogamente, la función *DOWN_Slope_Detected()* será la encargada de detectar flancos de bajada, y en caso de hacerlo libera el *click* en cuestión. Esta lógica permite que el *click* se mantenga presionado mientras el botón está accionado.

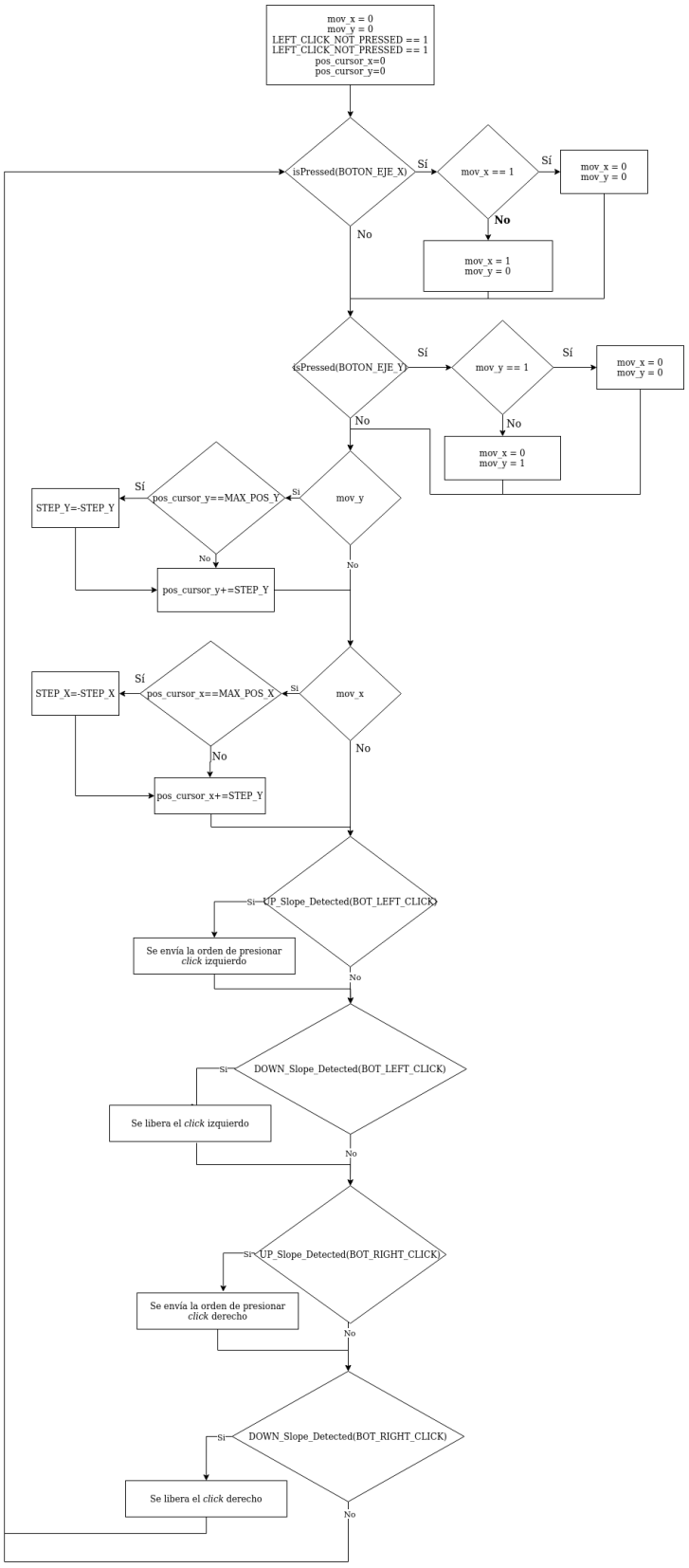

Figura 4.2: Diagrama de flujo Barrido

4.3. Botonera

Esta sección está dedicada a explicar el funcionamiento a nivel de *software* del modo Botonera, que fue presentado en la Subsección 2.2.2. En este modo se usan seis botones: cuatro botones que disparan movimientos en las cuatro direcciones, y otros dos botones que realizan la acción de *click* principal y secundario. Cabe destacar que en este modo, a diferencia del modo Barrido, el movimiento se lleva a cabo siempre y cuando el botón esté presionado. Cuando el usuario deja de presionar, el botón el cursor se detiene.

El flujo de este modo es sencillo; consiste en una arquitectura *round-robin*, en donde se chequea si alguno de los botones está presionado, como se puede observar en la imagen 4.3. El mismo comienza verificando si alguno de los cuatro botones de dirección está presionado. Si alguno de ellos lo está se mueve el cursor en la dirección correspondiente a ese botón. Luego se ejecuta la lógica correspondiente a los *click*; al igual que fue explicado para el modo Barrido en la Sección 4.2, mediante las funciones *UP_Slope_Detected()* y *DOWN_Slope_Detected()* se detectan los flancos de subida y bajada respectivamente, con el fin de mantener presionado el *click* mientras el botón correspondiente se encuentre accionado.

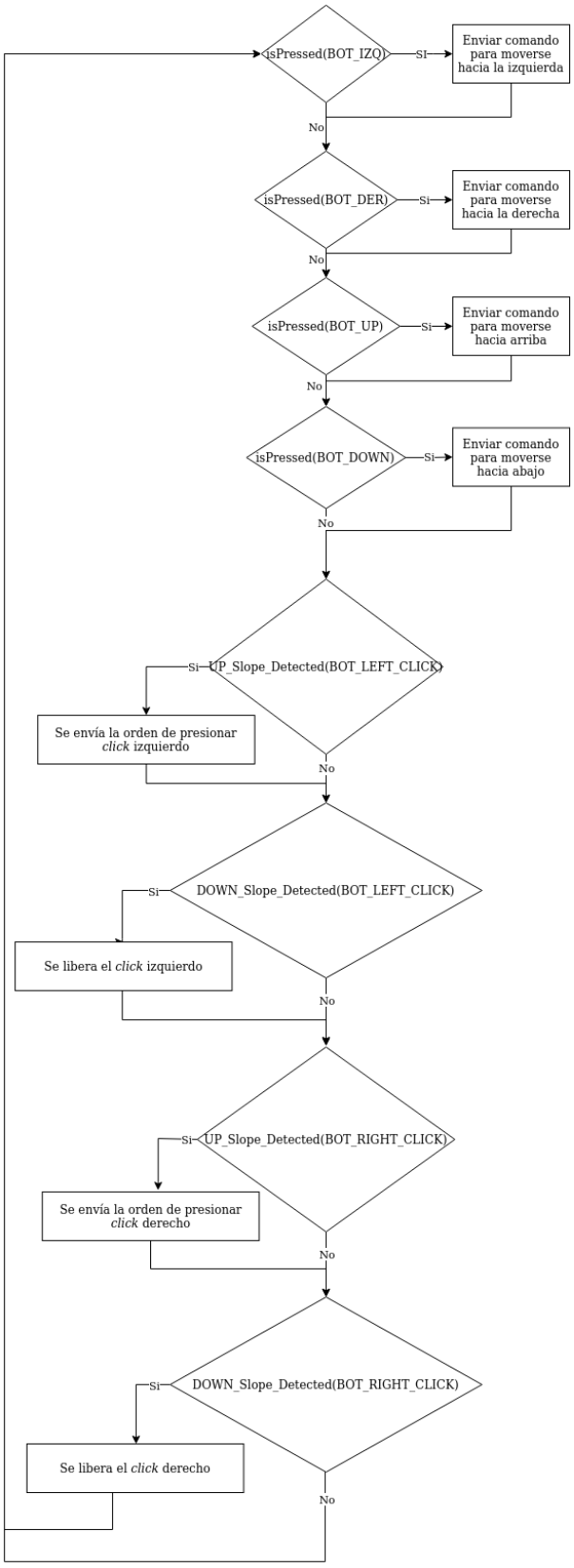

Figura 4.3: Diagrama de flujo Botonera

4.4. Cabeceo

Este modo de uso tiene una lógica muy similar a modo Botonera, en el sentido que mientras se verifique una condición se mantiene el movimiento en ese eje. La diferencia radica en la condición *per se*; en el caso del modo Botonera la condición es que un botón esté presionado, mientras que en este caso se evalúan los ángulos respecto a la vertical. Para realizar esto es necesario tener la posición angular del dispositivo de forma confiable; estos datos se obtienen del algoritmo de orientación, como se menciona en la Sección 3.4. Para facilitar la comprensión de cómo funciona este modo, es útil recordar cómo se definen los ángulos *pitch* y *roll*, cuya definición fue introducida en la Subsección 2.2.3.

En la figura 4.4 se presenta el diagrama de flujo de este modo. Como se puede ver, hay una instancia en la que se obtiene la orientación inicial del dispositivo; esto se realiza promediando las medidas de *pitch* y *roll* obtenidas durante 500 ms. Estas medidas serán consideradas como el origen de coordenadas, y los movimientos serán en referencia a esta posición angular.

Una vez definido el origen, el modo consiste en un bucle que considera únicamente la posición angular del dispositivo con respecto a dos ejes (*pitch* y *roll*) y la velocidad angular según el eje restante (eje Z). Como se puede ver, el movimiento del cursor no sólo está sujeto a la orientación del dispositivo, sino al estado en el que el *software* se encuentra; de esta forma, una vez que se determinó un movimiento, por ejemplo hacia abajo, el cursor se moverá en esta dirección hasta que el *pitch* vuelva a valores cercanos a la referencia obtenida inicialmente, sin importar el valor del *roll*. Esto determina que en este modo no se soportan movimientos en direcciones intermedias entre los semi ejes; una vez superado el umbral correspondiente a un semi eje se ignorará el otro hasta volver al estado *Freno*. Si bien esto obliga al usuario a volver siempre a la posición inicial, minimiza las posibilidades de movimientos involuntarios del cursor.

Además de la posición angular del dispositivo, vale notar que se considera la velocidad angular según el eje Z para hacer *click*; en este caso no se considera el sentido del giro ya que según fue descrito en la Sección 3.3 la información referente a la orientación para este ángulo no es confiable.

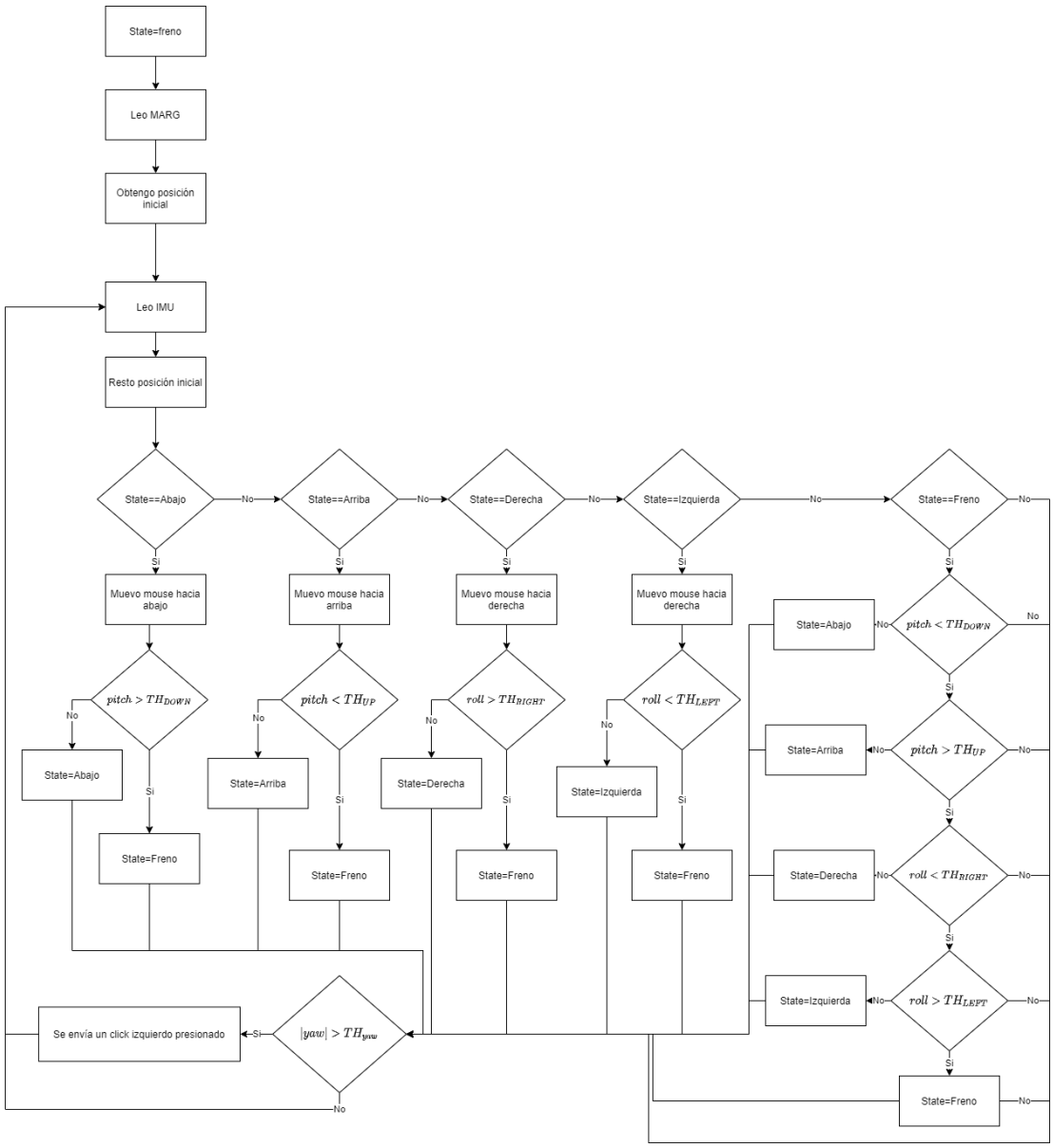

Figura 4.4: Diagrama de flujo de modo Cabeceo

4.5. Gestos

Este modo es el que posee la mayor complejidad de los hasta aquí detallados; el dispositivo detecta distintos movimientos predefinidos (denominados "gestos") y, en base a un mapeo interno, los traduce en acciones del cursor. En la presente sección se hablará en detalle de cómo se lleva a cabo la detección y clasificación de dichos gestos.

Previo a profundizar en cómo se realiza esto, resulta pertinente explicitar cuáles serán las características de estos gestos a detectar, y que acciones desencadenarán; en esencia serán movimientos lineales en cada uno de los ejes (X, Y y Z), pudiendo ser en dirección positiva o negativa. Los movimientos del dispositivo en el eje X desencadenan desplazamientos del cursor en el eje horizontal y, análogamente, los movimientos del dispositivo en el eje Y se mapean en movimientos verticales del cursor. Por último los gestos en el eje Z (como por ejemplo levantar el dispositivo) servirán para frenar el movimiento del cursor. La magnitud mínima que los gestos deberán tener para ser clasificados será configurable mediante el manejo de un umbral, existiendo algunas excepciones (las cuáles se profundizarán más adelante). A su vez los movimientos angulares se englobarán en un gesto particular denominado *shake*, que se mapeará en un *click* izquierdo acompañado del frenado del cursor.

La complejidad mencionada para este modo se debe también a que es el que utiliza un mayor volumen de datos en comparación con los otros. Para la detección de gestos será necesario apoyarse en la información brindada por los tres sensores del *MARG* (acelerómetro, giroscopio y magnetómetro), utilizando así la mayoría de las señales descritas en la Sección 3.4. Tanto las velocidades angulares como las aceleraciones (crudas y procesadas) jugarán un rol fundamental a lo largo de esta sección.

4.5.1. Descripción del Algoritmo de Detección

Como fue brevemente expuesto con anterioridad, este modo tiene como función principal detectar gestos que el usuario realice con el dispositivo. Estos podrán ser angulares o lineales, siendo estos últimos los que revisten una mayor complejidad. Por esto, antes de adentrarnos en los pormenores de la implementación, se hará una breve introducción del funcionamiento del algoritmo de detección de gestos.

En la figura 4.5 se puede ver en rasgos generales la lógica utilizada para realizar la detección. Lo que allí se representa es realizado para cada uno de los ejes X, Y y Z, que a su vez tienen asociados tres magnitudes:

- Aceleración: tras el procesamiento de la información proveniente del *MARG* se tiene la aceleración instantánea.
- Velocidad: teniendo los valores de aceleración se realiza una integral y se obtiene la velocidad del dispositivo en el eje correspondiente, la cual es almacenada en un *buffer* de tamaño predeterminado.
- Suma de velocidades: se realiza una suma de los elementos almacenados en el *buffer* de velocidades y se usa este valor para determinar el inicio de un gesto, así como su magnitud (máximo de esta suma) y sentido.

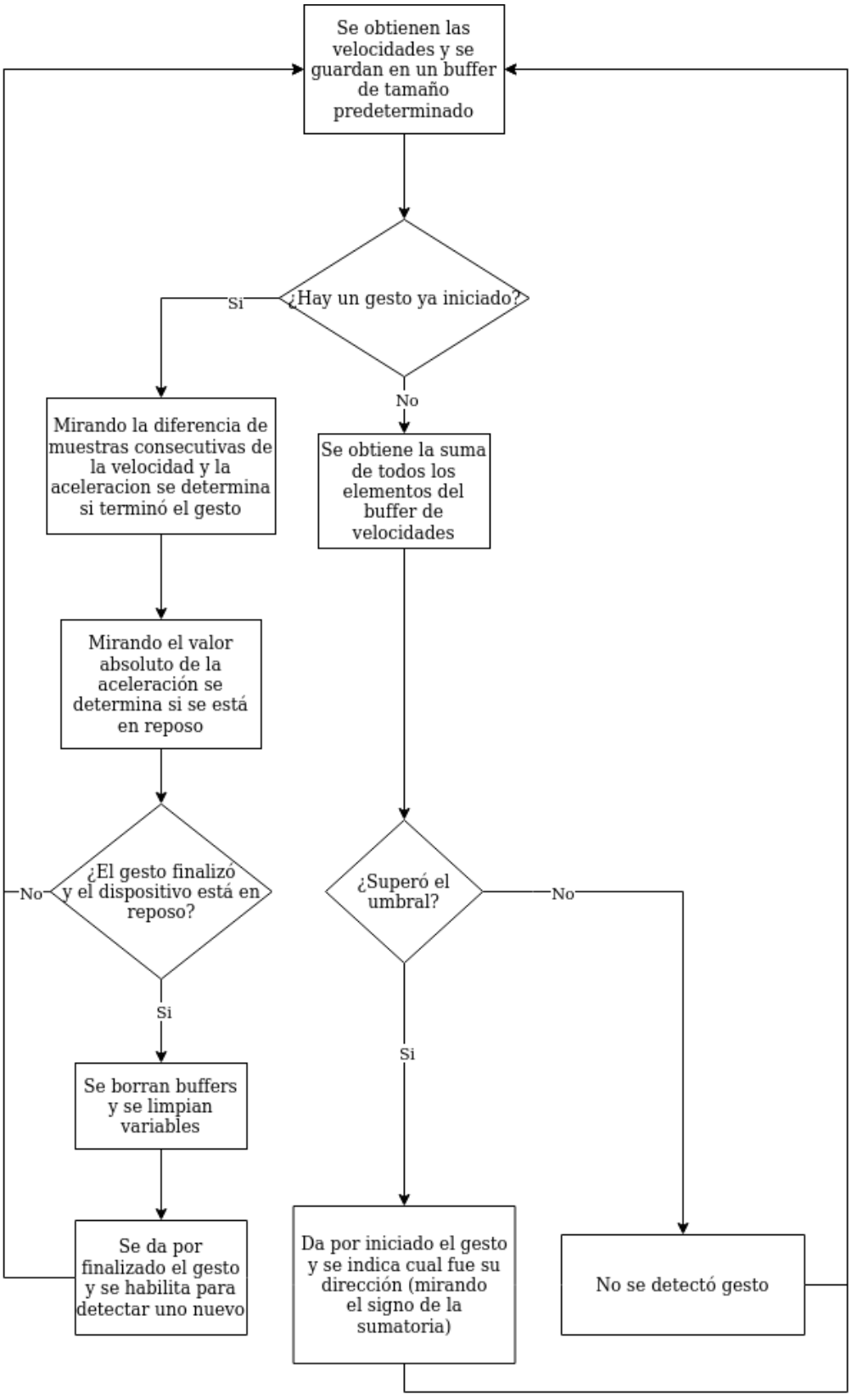

Figura 4.5: Algoritmo de detección

4.5.2. Procesamiento de las Señales

Además de las señales detalladas en la Sección 3.4, el algoritmo de detección de gestos se apoya también en dos señales nuevas: *velocidad* y *suma de velocidades*. A continuación se detallará como se obtienen las mismas, así como sus características.

Velocidades

A partir de las aceleraciones correspondientes a cada eje (variando entorno a cero ya que se les restó la componente de g), es momento de obtener la velocidad del dispositivo. Para esto se utiliza la lógica descrita en el diagrama de la figura 4.6, la cual es aplicada para cada uno de los ejes por separado, obteniendo así las velocidades en las tres direcciones principales (X, Y y Z).

Para este cálculo se tienen en cuenta dos factores: primero, considerando el problema mencionado en la Sección 3.4, que genera que se pueda tener aceleraciones distintas a cero aún cuando se está en reposo, se compara el módulo del vector de aceleraciones crudas (*accel_module*) con g; si esta diferencia es cercana a cero, se puede determinar que sólo se encuentra presente la aceleración gravitatoria, y por ende el dispositivo está quieto, lo que deriva en que se fije velocidad nula. Si bien existen casos en los que el módulo de la aceleración cruda es de valor g aún cuando el dispositivo está moviéndose (*e.g* si se mueve con aceleración $2g$ hacia arriba), se consideran como casos excepcionales y no se contemplan en la lógica. En segunda instancia se debe tener en cuenta que la aceleración, como es de esperar, tiene un piso de ruido superpuesto. Esto ocasiona que nunca sea estrictamente cero, y por esto es que se verifica que los valores estén por encima de cierto umbral; si es en términos relativos cercana a cero, se considera que la aceleración instantánea (*accel_inst*) es nula.

La ecuación convencional para la obtención de la velocidad es

$$\vec{v}(t) = \vec{v_0} + \vec{a}(t) \times \Delta t \tag{4.1}$$

con $\vec{v_0}$ la velocidad inicial. En este caso particular la ecuación 4.1 se aplica para cada uno de los ejes y, al estar trabajando en el contexto discreto, la velocidad inicial corresponde a la velocidad obtenida en la iteración anterior. Es por esto que la fórmula se reduce a lo que vemos en el diagrama 4.6 de la forma

$$vel_i[n] = vel_i[n-1] + accel\_inst_i[n] \times \Delta t$$

donde n es el tiempo discreto (iteraciones del algoritmo) e $i \in \{X, Y, Z\}$. Esta notación será utilizada de ahora en más para referirnos a las señales y valores utilizados. Cada uno de estos valores obtenidos es almacenado en un *buffer* de tamaño fijo correspondiente al eje.

En la imagen 4.7 vemos un ejemplo de las velocidades obtenidas. El caso que allí se grafica corresponde a dos movimientos en el eje X (uno en cada dirección), que en la gráfica se ven como dos curvas pronunciadas. Una característica interesante a observar es que, en la segunda curva, se observa la presencia de un pico de menor magnitud y en la dirección contraria al movimiento; ya que el movimiento observado fue realizado puramente en la dirección negativa del eje en cuestión, este pequeño pico de velocidad no

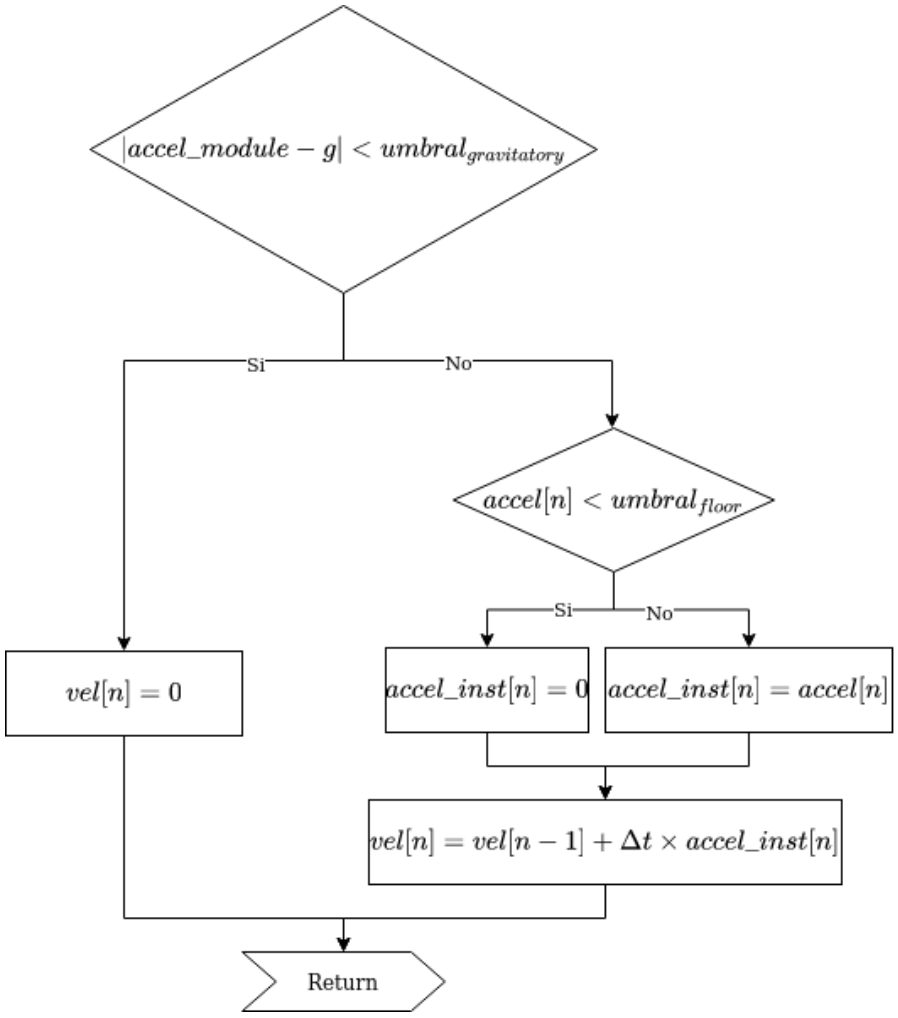

Figura 4.6: Diagrama de flujo: obtención de las velocidades

se condice con la realidad. Esto se debe principalmente a los errores propios de cualquier acelerómetro, así como al tratamiento que se le realiza a las aceleraciones (Sección 3.4), generando una pequeña deformación en la forma de onda que deriva en esta problemática. Al ser esto un efecto no deseado, deberá ser tenido en cuenta a la hora de realizar los posteriores análisis.

Sumatoria de velocidades

Como se comentó previamente en esta sección las velocidades según X, Y y Z calculadas son almacenadas en un *buffer* correspondiente a cada eje. Posteriormente (y para cada uno de los *buffers*) se calcula la suma de todos los elementos almacenados; esto surge como una estrategia para atacar el problema observado en la imagen 4.7, donde se evidencia un pequeño pico de velocidad inexistente en el movimiento real. En la gráfica de la figura 4.8 se ven contrastadas las curvas de velocidad y la suma de ellas; resulta claro que, aún cuando la velocidad tiene comportamientos no deseados, la sumatoria auspicia

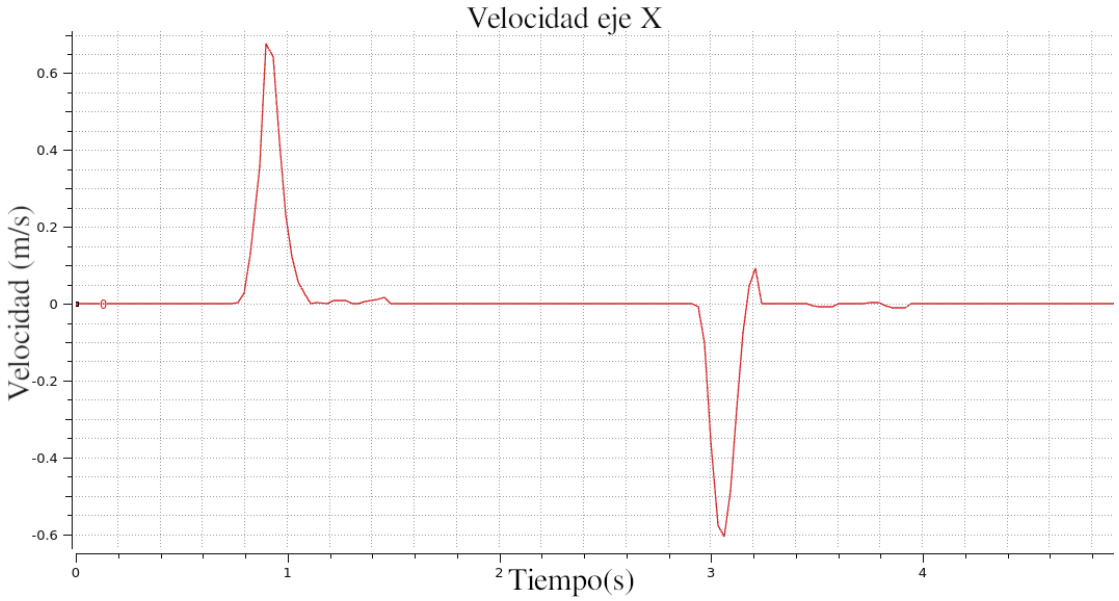

Figura 4.7: Gráfica de velocidades obtenidas

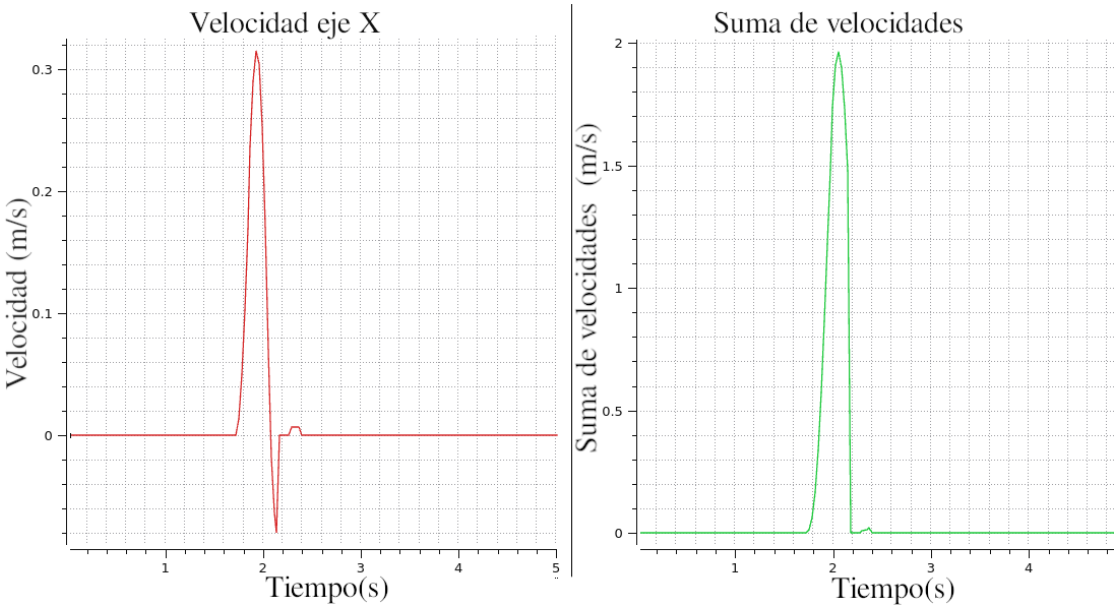

Figura 4.8: Gráfica de sumatoria de velocidades

como un filtro pasa bajos eliminando estas imperfecciones. Es por esto que en muchos de los análisis que se harán en las siguientes secciones, la sumatoria de velocidades cumple un rol fundamental.

4.5.3. Programa Principal Modo Gestos

A más alto nivel, el programa principal comienza configurando el sistema (tanto la parte de *Bluetooth* como el *MARG*) y se inicializan las variables que serán necesarias en su desarrollo. Luego se ingresa en un bucle donde constantemente se realizan las lecturas y procesamiento de las señales, para posteriormente pasar a verificar si fue detectado algún gesto. Finalmente esa información recolectada se traduce en acciones del *mouse*. En el diagrama 4.9 se puede ver una representación gráfica de este proceso.

En dicho diagrama aparecen algunas funciones que tienen como objetivo compartimentar las distintas tareas que el sistema debe realizar; la función *getGesture*() se encarga de determinar si el usuario realizó algún gesto con base en los datos procesados y/o provenientes directamente de los sensores. Esta información será guardada para que luego la función *get_mouse_order*() la recoja y la traduzca en las correspondientes acciones del cursor. Finalmente, *mouse_control*() será el encargado de enviar esa información al dispositivo controlado (PC, dispositivo móvil, etc.) vía *Bluetooth*.

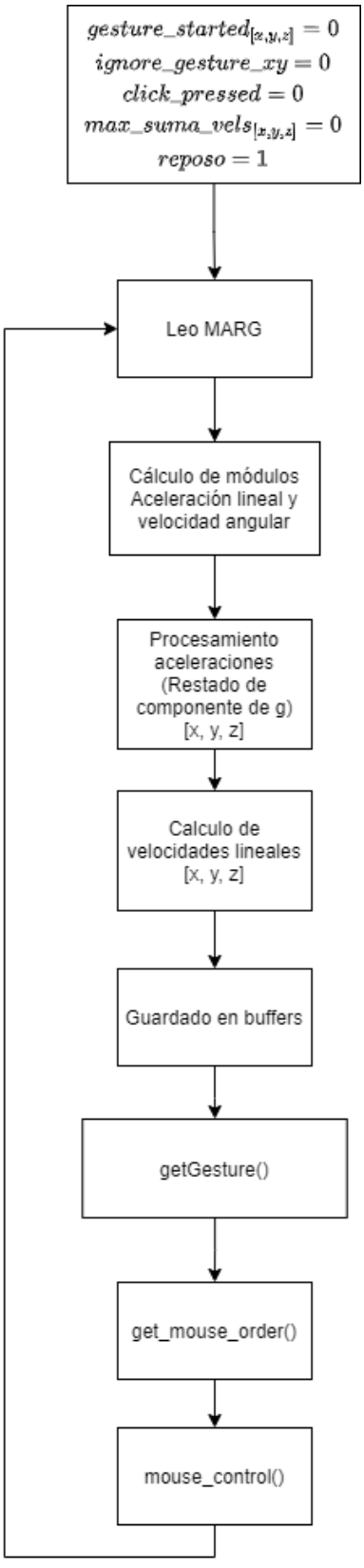

Figura 4.9: Diagrama de flujo del programa principal en modo Gestos

4.5.4. Detección de Gestos

Para el caso de la detección de gestos, en la función *getGesture()* podremos encontrar embebida toda la lógica necesaria. Justamente, en el diagrama de la figura 4.10 vemos el funcionamiento de esta función.

Como se puede apreciar, esta lógica se apoya a su vez en distintos bloques que se irán comentando en el correr de esta sección. A los efectos de entender este diagrama, se harán comentarios generales sobre algunas de las funciones expuestas. Inicialmente vemos que se realiza un llamado a la función *set_movement_flags()*, que básicamente será la que determine cuando el dispositivo está quieto. A continuación se invoca a la función *detect_classify_gestures()* que, en esencia, es la responsable de determinar si se detectó un gesto en alguno de los ejes y clasificarlo según su dirección y magnitud. Por otro lado, también aparece la función *is_shake()* que será la encargada de detectar el gesto de *shake*. En este diagrama aparecen magnitudes asociadas a los ejes X, Y y Z que son representadas explicitando el eje correspondiente entre paréntesis rectos (*[ ]*).

En las figuras 4.11 y 4.12 podemos ver como se comporta el flujo para el caso que se realiza un *shake* con el dispositivo, y posteriormente se ejecuta un gesto en el eje X (lo cual cubre todos los caminos posibles de este flujo); cuando el *shake* es detectado impacta en el flujo de *getGesture()* de forma tal que se comienza a tomar el camino de ignorar gestos en los ejes X e Y (*ignore_gestures_xy=1*) y, mientras esta condición se mantenga, los gestos realizados no se traducirán a movimientos del cursor. Esto se mantiene hasta detectar que el dispositivo ha recobrado el estado de quietud, lo cuál se representa con la variable *reposo=1*. Un hecho interesante a destacar es que la curva de aceleración no brinda información de utilidad en el transcurso de un *shake*, lo cuál es uno de los motivos para ignorar los gestos en X e Y mientras que este no se extinga completamente. Una vez que se detectó que el dispositivo retornó a la quietud, el sistema queda a la espera de un nuevo gesto; para el caso puntual del ejemplo mostrado, a continuación se realiza un gesto en el eje X, el cual desencadena un movimiento del cursor al recorrer el camino por defecto del flujo.

A su vez, vemos que se pregunta si la sumatoria de velocidades en el eje Z *max_suma_vels[z]* supera cierto umbral; cuando esto ocurre, también se activa la bandera *ignore_gestures_xy* al igual que en el caso que *is_shake()* retorna un valor verdadero. En el diagrama de la figura 4.17 se muestra de donde proviene este valor de *max_suma_vels[z]*, y se mostrará cual es el recorrido dentro del flujo de *getGesture()* cuando se supera el umbral establecido.

La primer función que aparece en el flujo del diagrama de *getGesture()* es *set_movement_flags()*; como se mencionó, esta función será la que determine si el dispositivo se encuentra en estado de quietud. El llamado a esta función se realiza al comienzo del flujo ya que, como veremos más adelante, la información devuelta será de utilidad en algunas de las demás funciones.

En resumidas cuentas, esta función determina si todos los gestos han finalizado y si el dispositivo está en reposo, retornando esta información mediante el manejo de dos variables *booleanas* (*gesture_finished* y *reposo* respectivamente). Si bien estos dos conceptos pueden resultar similares, no son del todo equivalentes. Como se ve en el diagrama de la figura 4.13, para determinar que no hay ningún gesto en curso se utiliza la diferencia de velocidades y aceleraciones; es decir, si la variación de estas magnitudes es considerable-

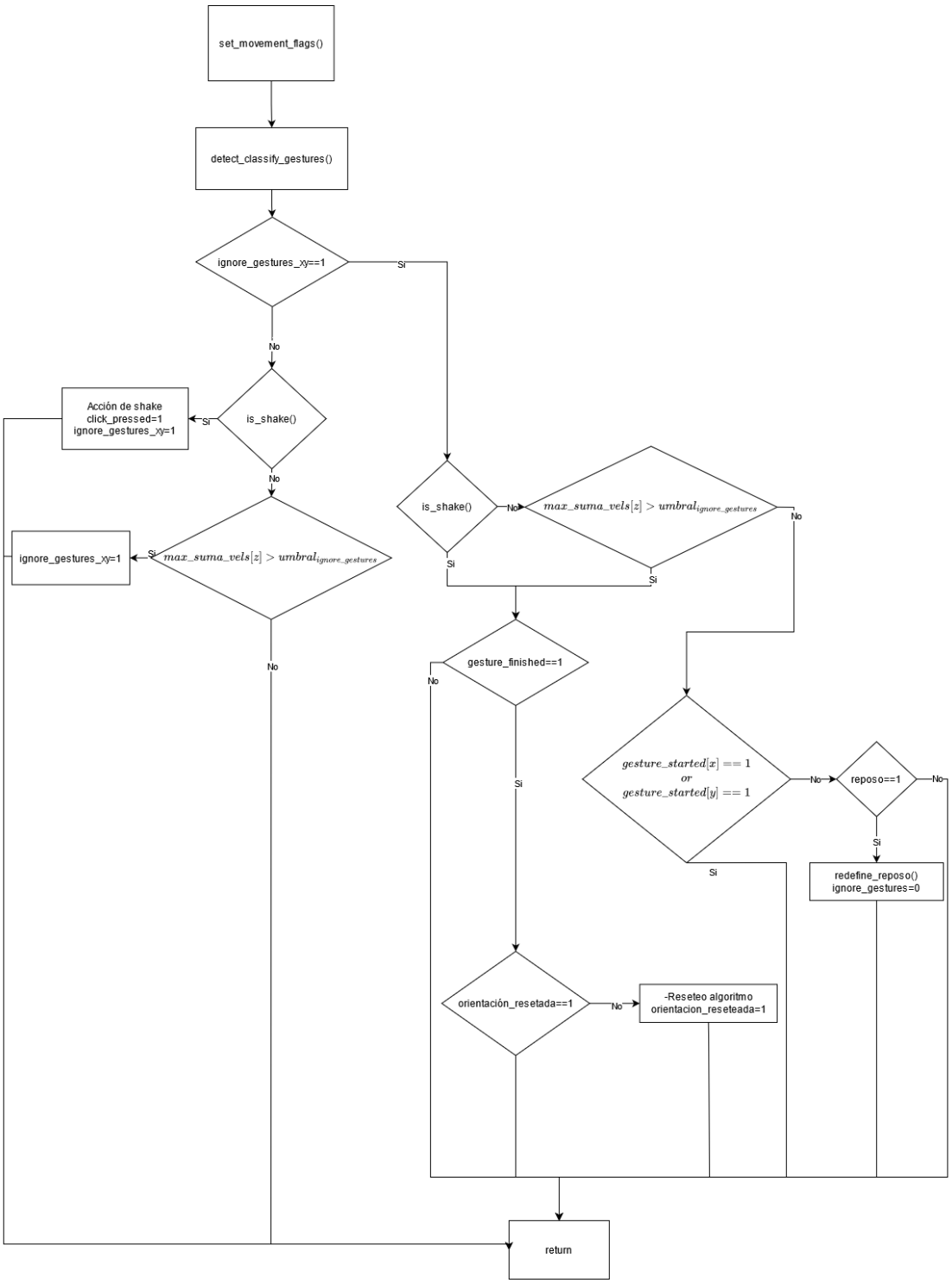

Figura 4.10: Diagrama de flujo de la función *getGesture()*

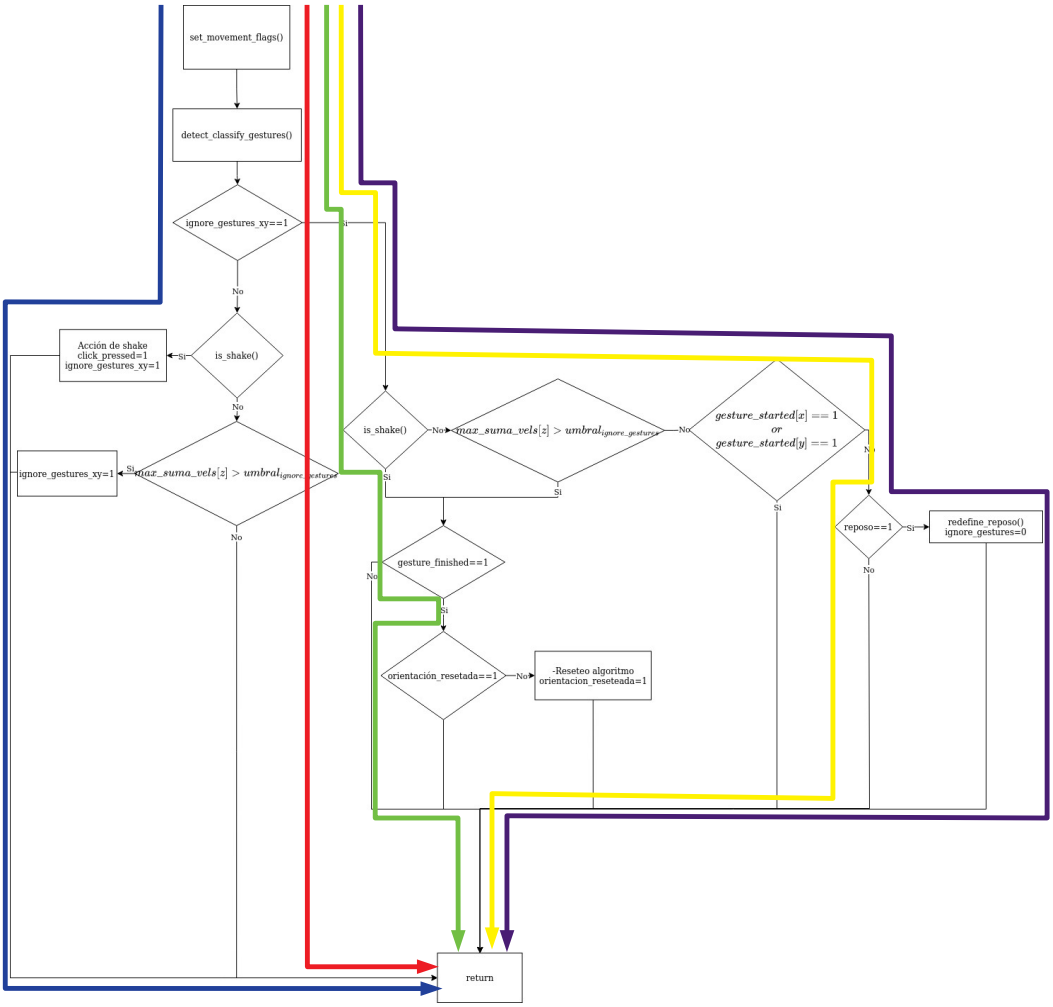

Figura 4.11: Diagrama getGesture() con distintos recorridos

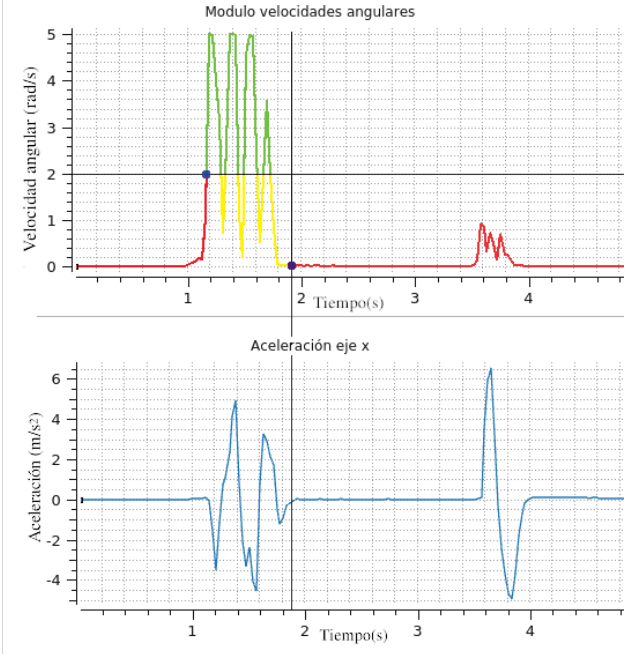

Figura 4.12: Módulo de velocidades angulares y aceleración en el eje X

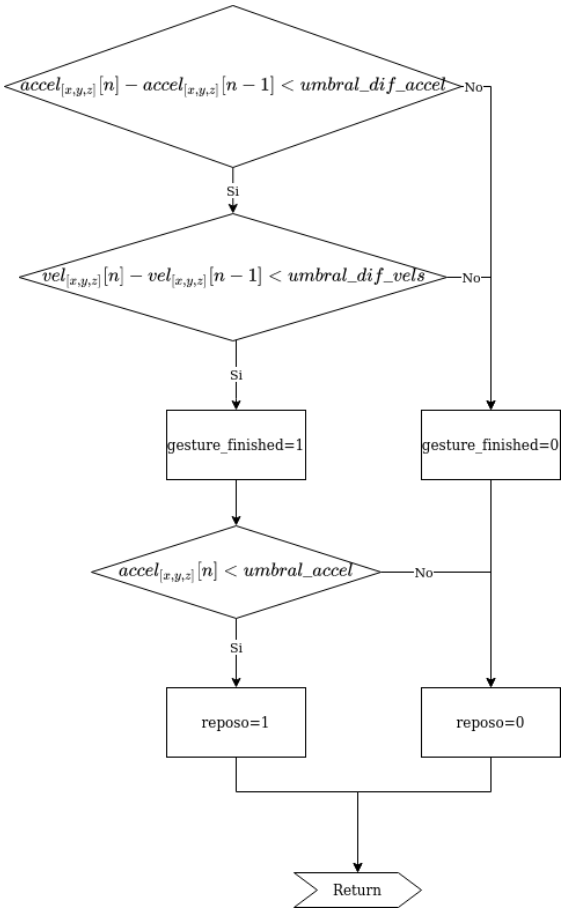

Figura 4.13: Diagrama de flujo de la función *set_movement_flags()*

mente pequeña, se puede determinar que no hay un gesto transcurriendo. Por otro lado, para la determinación de que el dispositivo está en reposo se exige que las aceleraciones tengan un módulo cercano a cero. Si tenemos en cuenta lo explicado en la Sección 3.4 y, particularmente, en la figura 3.23, este concepto de *reposo* considerado en el algoritmo puede resultar un tanto confuso. Debido a la problemática de la convergencia del algoritmo de orientación y, en consecuencia, de las aceleraciones, existen casos en los que el dispositivo estará quieto y el sistema demorará cierto tiempo (en general pequeño, pero existente) en detectarlo. Esta tardanza en la detección del reposo auspicia, en parte, como una protección frente a esa problemática mencionada y evita que se analicen datos erróneos (como aceleraciones distintas de cero cuando no existe movimiento). Para apoyar esto que se menciona, podemos ver la figura 4.14; allí vemos cual es el comportamiento de la aceleración procesada del eje X cuando el dispositivo se encuentra en reposo sobre una superficie plana paralela a dicho eje, se realiza un movimiento errático y luego se retorna a la posición inicial. Para este caso particular, la aceleración tarda un tiempo más que considerable (en el entorno de 600 ms) en estabilizarse.

Este comportamiento se debe a que el algoritmo de orientación está optimizado para movimientos breves (menores a 1 segundo). Esto quiere decir que para movimientos más

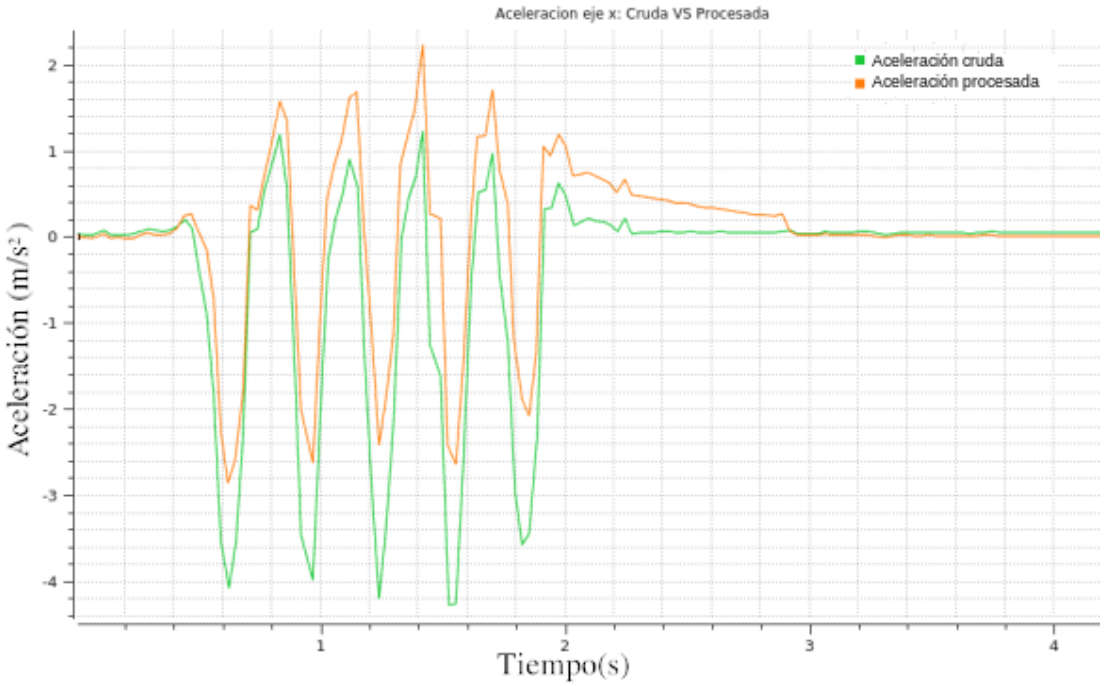

Figura 4.14: Aceleración cruda *vs* aceleración procesada

largos los valores obtenidos se separan de la realidad. Para entender qué tanto afecta la duración del movimiento a la correctitud de las medidas de aceleración, podemos referirnos a las figuras 4.15 y 4.16 extraídas de la documentación oficial de *NXP* [31]. En dichas gráficas, la curva marcada como *TRUE* representa la aceleración cruda, mientras que *FUSION* hace referencia a la aceleración procesada. Como resulta claro, cuanto mayor sea el tiempo del movimiento, menor será la precisión del algoritmo.

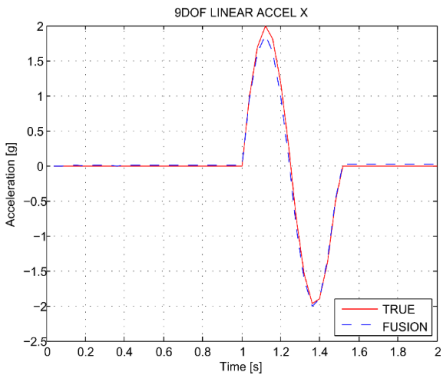

Figura 4.15: Aceleración real *VS* la obtenida de la fusión de sensores, movimiento de 500 ms

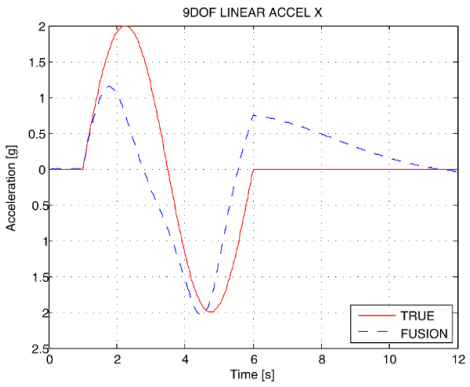

Figura 4.16: Aceleración real *VS* la obtenida de la fusión de sensores, movimiento de 5 s

Si bien hasta ahora no se habló con demasiada profundidad de la función *detect_classify_gestures()*, ésta juega un papel fundamental en la lógica del sistema. Como se comentó con anterioridad, es esta función la encargada de realizar la detección y clasificación de los gestos que se realicen en los ejes X, Y y Z, para que después se tomen las decisiones correspondientes.

En el diagrama de la figura 4.17 podemos ver el principio de funcionamiento de esta función, y se puede decir que, a groso modo, es quien implementa la mayor parte de la lógica de alto nivel observada y comentada en la Subsección 4.5.1.

Para simplificar el entendimiento del diagrama 4.17, tenemos las figuras 4.18 y 4.19; en ellas se puede observar el caso en el que se ejecuta un gesto en el eje X, y cuales son los distintos caminos que se van tomando dentro del flujo para su detección. La curva de la figura 4.19 representa la sumatoria de velocidades para el eje X (*suma_vels[x]*) que se comentó en la Sección 4.5.2, donde vemos que cada segmento de la curva está coloreado y corresponde a un camino particular dentro del diagrama 4.18.

Al ingresar se observan dos caminos claros, dependiendo de si hay un gesto iniciado o no. Vale notar que varias de las acciones que se ven representadas en el diagrama son realizadas para los tres ejes independientemente. Esto se ve plasmado utilizando el subíndice $[x, y, z]$. Para el caso en el que no hay un gesto iniciado, se ejecutan los análisis correspondientes para detectar el próximo gesto que sea realizado. Conforme avanzamos por esa rama del esquema nos topamos con la función *is_gesture()*, la cual determina si la forma de onda de la aceleración cruda coincide con la esperada en caso de la realización de un gesto.

Como se ve en la figura 4.20, esta función obtiene la varianza de las aceleraciones crudas para los valores almacenados en el *buffer* (*varianza_accels*), retornando verdadero si dicho valor supera cierto umbral superior. Una vez que se supera dicho umbral, se seguirá retornando 1 hasta que la varianza se sitúe por debajo de un umbral inferior. Esto se realiza con el fin de que *is_gesture()* retorne verdadero durante todo el proceso del gesto. De este modo se podrá determinar que la curva de aceleraciones crudas cumple con los requisitos para ser un gesto válido. Observando la figura 4.21 podemos apreciar cómo crece la varianza al realizar el gesto; se utiliza este método ya que cómo se vio en la Sección 3.4 (y particularmente en la figura 3.23) la aceleración puede demorar en volver a cero, y por lo tanto podría darse que la velocidad supere el umbral, configurando así un falso positivo.

Ante un resultado verdadero de la función *is_gesture()*, se debe determinar si el gesto tiene las características como para ser detectado. Aquí es donde entrará en juego el valor de *suma_vels* graficado en la figura 4.19; cuando supera cierto umbral preestablecido, se determinará que en ese eje está transcurriendo un gesto. Para determinar el sentido en el que se está realizando basta con observar el signo de la suma *suma_vels* y sabremos si transcurre en la dirección positiva o negativa del eje en cuestión. Por último, para determinar la magnitud del mismo se buscará el máximo del valor absoluto de la suma (el cual se almacena en *max_suma_vels*); una vez alcanzado se tendrá el gesto completamente clasificado y se da por comenzado a través del manejo de la variable *gesture_started*.

Una vez iniciado el gesto, el flujo de *detect_classify_gestures()* será tal que se esperará que finalice. Para esto, y como se comentó brevemente con anterioridad, se usará la información brindada por la función *set_movement_flags()* (figura 4.13). Una vez que se

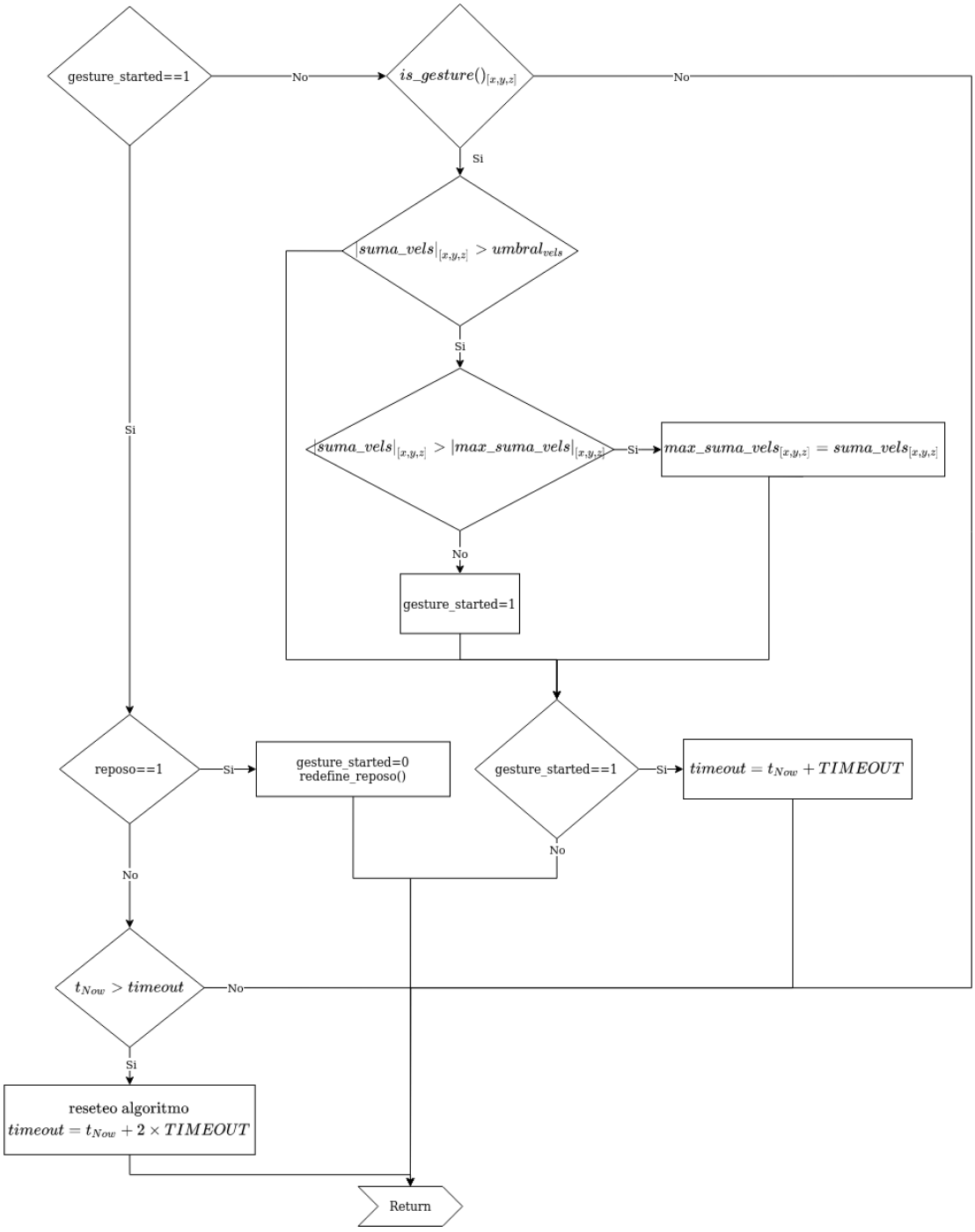

Figura 4.17: Diagrama de flujo de la función *detect_classify_gestures()*

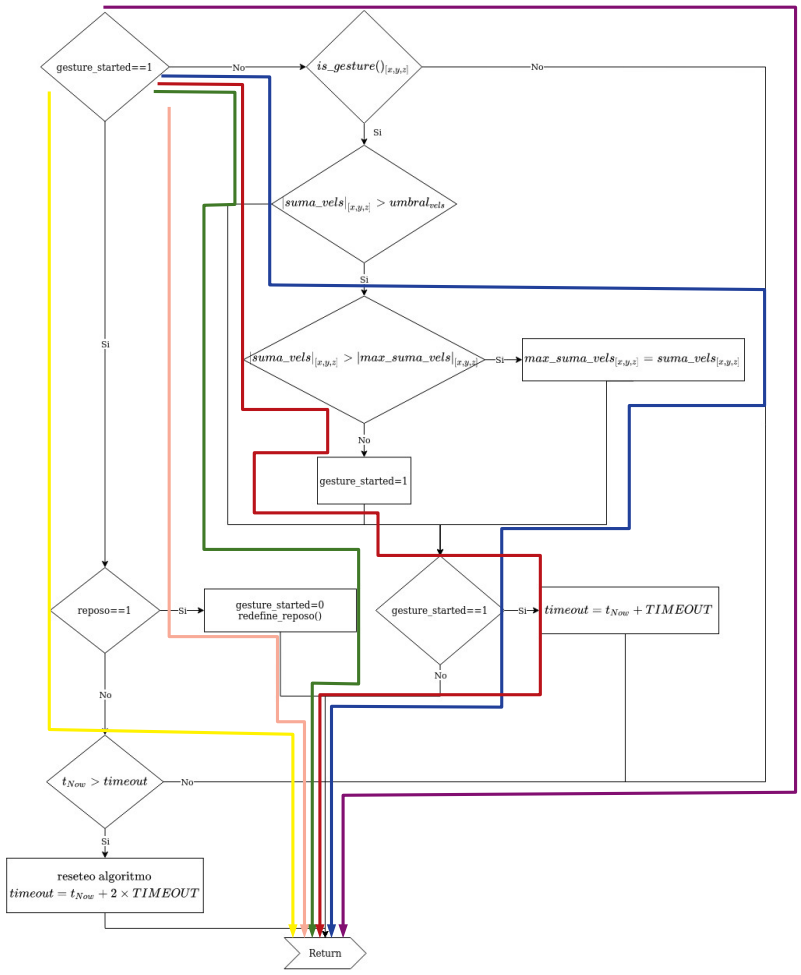

Figura 4.18: Diagrama con recorridos: detección de gesto

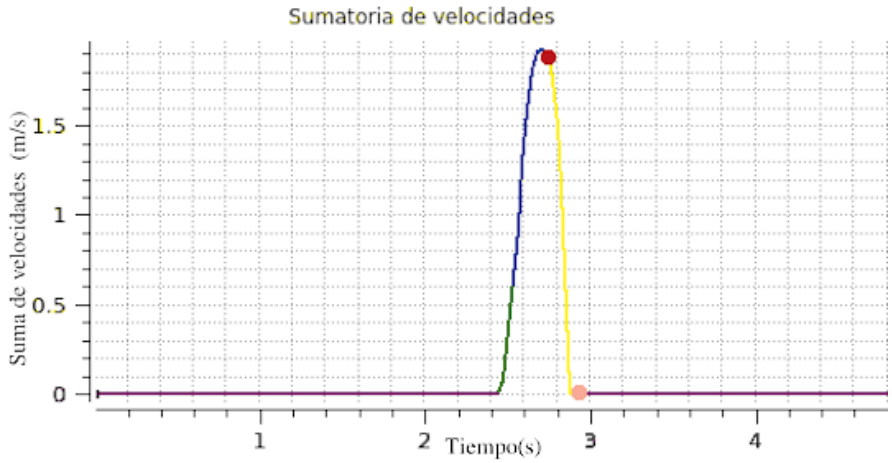

Figura 4.19: Sumatoria de velocidades cuando se ejecuta un gesto

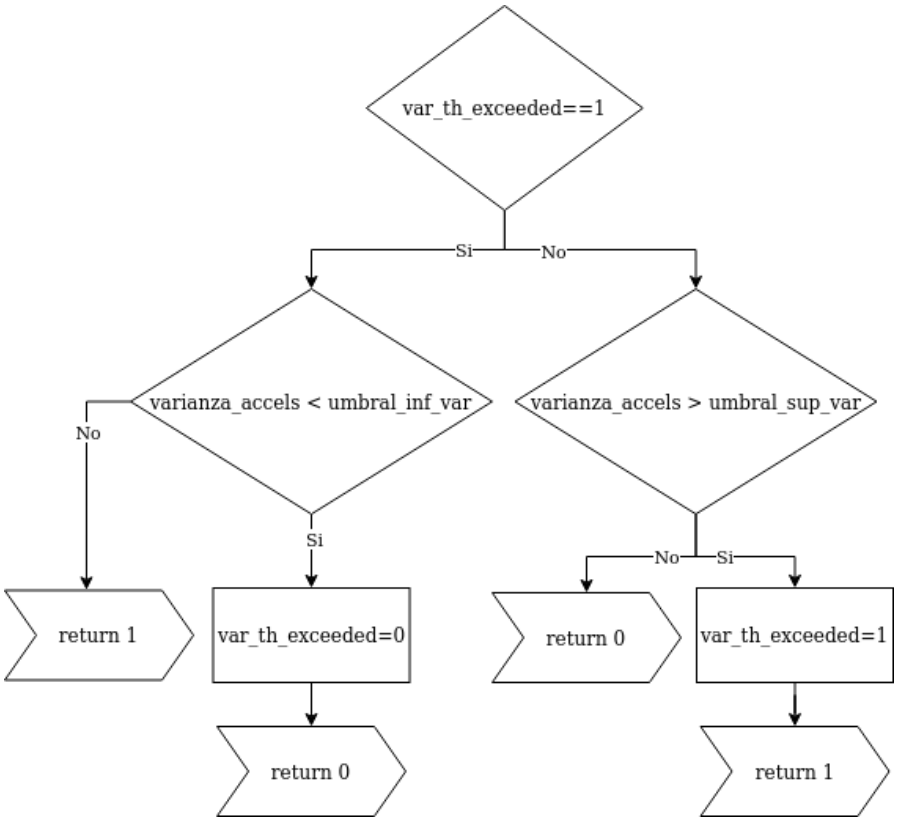

Figura 4.20: Diagrama de flujo de la función *is_gesture()*

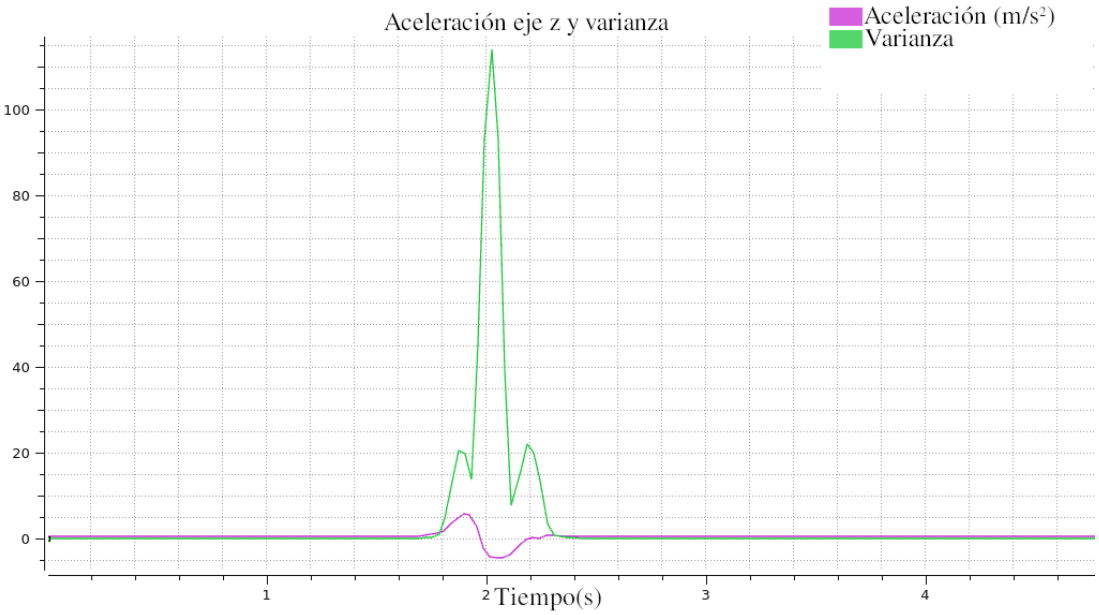

Figura 4.21: Aceleración cruda y su varianza en el transcurso de un gesto

detecta que el dispositivo ha quedado en reposo, se limpian todos los *buffers* y se reinician las variables usadas a su estado inicial. Estas acciones son realizadas por la función $redefine\_repose()$. Finalizado este proceso, el sistema estará listo para la ejecución de un nuevo gesto.

Sabiendo ya como se realiza la detección de los gestos lineales es hora de realizar lo propio sobre los movimientos angulares a detectar; en el diagrama de *getGesture()* (figura 4.10) aparece la función *is_shake()* que, como se mencionó, es la encargada de detectar los gestos de tipo *shake*. La operativa de este bloque es bastante sencilla como se puede ver en la figura 4.22. Básicamente lo que se hace es verificar si el módulo del vector velocidad angular (*omega_module*) supera cierto umbral preestablecido. En caso de verificarse la condición se puede determinar que está transcurriendo un *shake* y eso desencadenará ciertas acciones en el flujo de *getGesture()*. Como se apreció en el diagrama 4.11, la detección de este gesto angular genera que se ignoren los demás gestos realizados. Además, al momento de detectar por primera vez su presencia, se fijará una bandera que a la postre será utilizada para realizar un *click*.

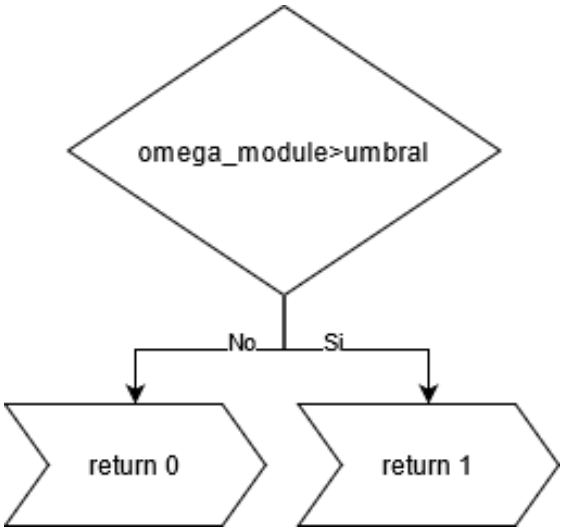

Figura 4.22: Diagrama de flujo de la función *is_shake()*

Ahora que fue explicado cual es el procedimiento para la detección de gestos, estamos en condiciones de realizar una pequeña retrospección y analizar el vínculo entre la detección y sus efectos. Recordando el diagrama de flujo de *getGesture()* (figura 4.10), para que un gesto detectado sea tenido en cuenta es necesario recorrer el camino por defecto; es decir, que la respuesta a todas las preguntas sea "no". También resulta interesante observar qué ocurre en el flujo de *getGesture()* cuando se detecta un gesto en el eje Z; como factor principal a destacar, tenemos que no cualquier gesto en dicho eje será tenido en cuenta. Debido a que en muchos casos (principalmente cuando no se mueve el dispositivo sobre una superficie plana) los gestos en X y/o Y vienen acompañados por gestos en Z, es pertinente tomar acciones que contemplen esta problemática. Es por esto que se impondrá una cota inferior en la magnitud de los gestos en Z para que estos desencadenen la acción de ignorar lo que ocurre en X e Y.

En resumidas cuentas, se exigirá que *max_suma_vels[z]* supere un umbral mayor al que aparece en 4.17. En las figuras 4.23 y 4.24 vemos en detalle cual es el comportamiento de *getGesture()* a medida que comienza a transcurrir un gesto en el eje Z. Vale destacar que el umbral impuesto en 4.24 (línea negra horizontal) es considerablemente mayor al impuesto en 4.19. Esto a su vez implica que los gestos de magnitud menor a la establecida serán detectados correctamente, pero no desencadenarán ninguna acción en el flujo principal del algoritmo.

Como se puede apreciar, uno de los caminos remarcados lleva a la decisión de reiniciar

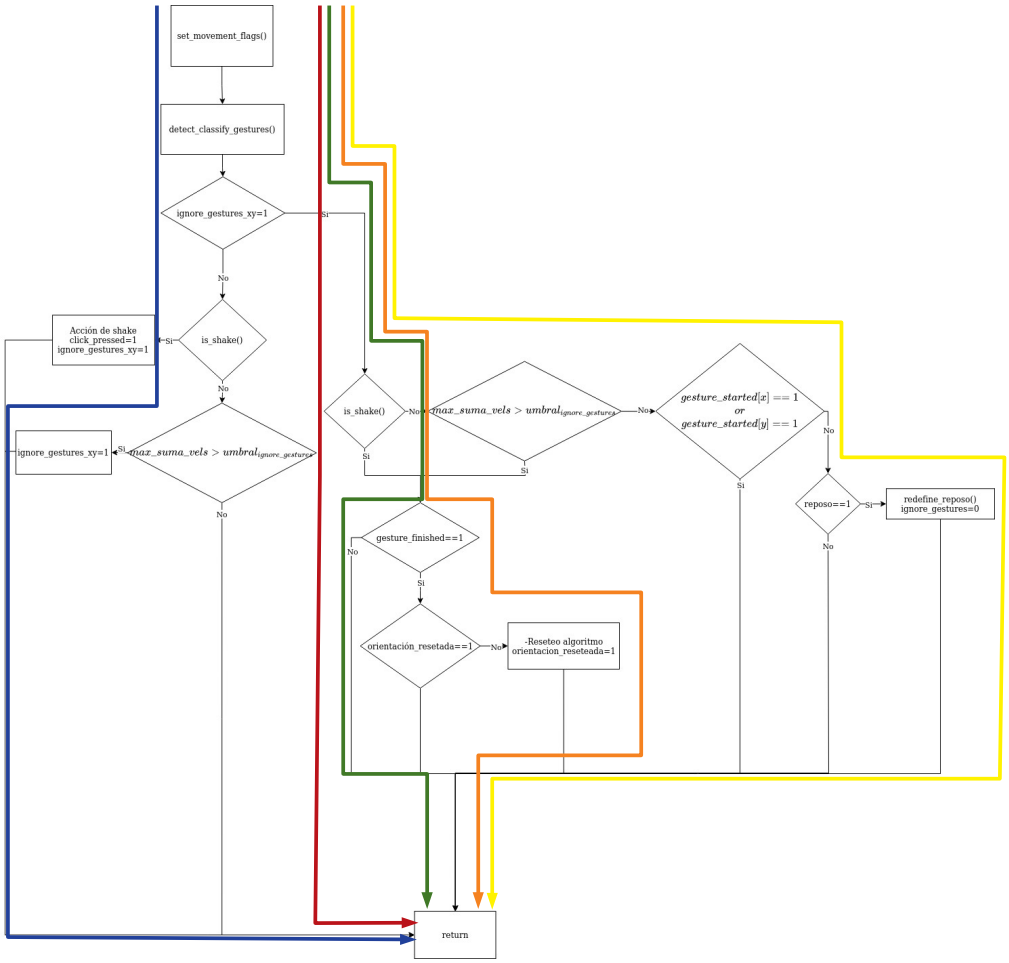

Figura 4.23: Diagrama getGesture(): gesto Z

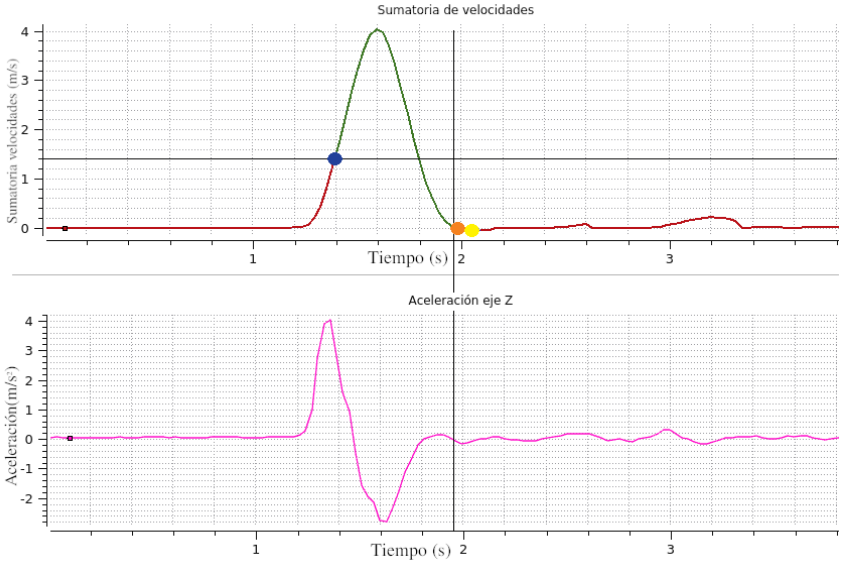

Figura 4.24: Módulo de velocidades angulares y aceleración en el eje X

el algoritmo de orientación, pudiéndose también acceder a este punto luego de un *shake*. Esto se debe a que empíricamente se comprobó que estas dos acciones generan que la información de orientación obtenida no sea confiable. Debido a que el algoritmo en cuestión se basa en el pasado para determinar la orientación en cada instante, reiniciarlo consigue que se elimine toda esa información remanente que generó el mal funcionamiento y así se logre una convergencia más rápida.

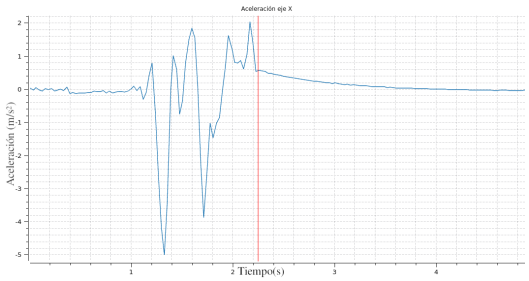

Figura 4.25: Aceleración eje X: sin reinicio de algoritmo

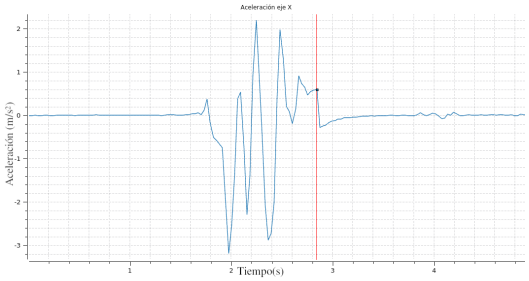

Figura 4.26: Aceleración eje X: con reinicio de algoritmo

Como soporte a esta idea se presentan las imágenes 4.25 y 4.26; en ambas observamos el comportamiento de la aceleración en el eje X luego de una acción que afecta negativamente al algoritmo de orientación (particularmente un *shake*). En la primera de ellas (figura 4.25) vemos el caso en que el algoritmo no es reiniciado luego de finalizado un comportamiento errático de la aceleración (linea vertical roja). Resulta claro notar que la señal tarda más de un segundo en converger al cero. Como contrapartida, en la figura 4.26 se aprecia el mismo caso pero con la diferencia que se reinicia el algoritmo cuando se detecta que el *shake* finalizó. Es evidente que el tiempo de convergencia se reduce considerablemente, dejando operativo el dispositivo más rápido que cuando no se realiza ninguna acción.

Por último, hay un concepto que no se ha tratado hasta el momento y es el de *timeout* o "tiempo limite"; en un uso normal del dispositivo, la finalización de gestos se detecta con precisión, unos pocos milisegundos después de que el dispositivo se dejó en reposo. No obstante, existen casos donde el uso anómalo (el cual se describe en el Apéndice D) genera que el gesto se mantenga activo un tiempo considerable después de haber efectivamente finalizado. Esto en esencia se debe a la convergencia del algoritmo de orientación, ampliamente tratada en este documento. Es en estos casos atípicos donde aparece en la ecuación el *timeout*, generando que se reinicie el algoritmo de orientación después de un tiempo suficientemente grande de comenzado el gesto. Esto genera que se agilice la convergencia y el gesto pueda ser debidamente finalizado.

4.5.5. Problemas Encontrados

El procesamiento y el algoritmo descritos previamente funcionan bien para una amplia gama de gestos lineales, permitiendo utilizar el dispositivo con cierta fluidez. No obstante, existen problemas en algunas señales y/o gestos que generan un notorio mal

funcionamiento del sistema.

Uno de los problemas que podemos encontrar surge al realizar movimientos en el eje paralelo a la gravedad; esto se ve representado en las figuras 4.27 y 4.28, donde podemos ver las aceleraciones crudas en Z y X respectivamente para los casos en que se realizan dos gestos seguidos con sentido opuesto. En los casos graficados el dispositivo se encuentra inicialmente apoyado en una superficie paralela al eje X. Como se puede observar, la forma de onda de la aceleración en X está mucho más definida que para el caso del eje Z. Este problema viene asociado al hecho de que, al realizar un movimiento en la dirección vertical, no se tiene una guía clara como en los casos de movimientos horizontales. Es decir, cuando el dispositivo esta apoyado sobre una superficie plana horizontal, ésta funciona como referencia para los movimientos en el plano que la contiene, cosa que no ocurre cuando se saca el dispositivo de ese plano. Esto genera un aumento considerable de las rotaciones del dispositivo, lo que se traduce en una gran variación de la componente de g en el eje en cuestión. Como es natural, esto conlleva problemas en las detecciones de los movimientos en el eje Z, derivando que en algunos casos se detecten movimientos que no son tales o se demore más tiempo del deseado en recobrar el estado de reposo.

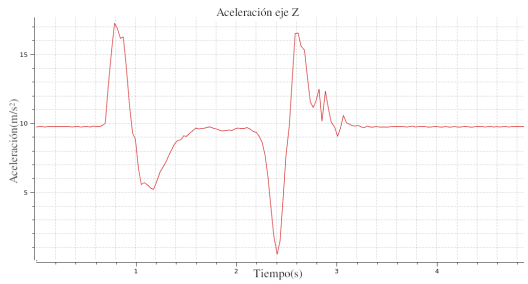

Figura 4.27: Aceleraciones en el eje Z: dos gestos opuestos

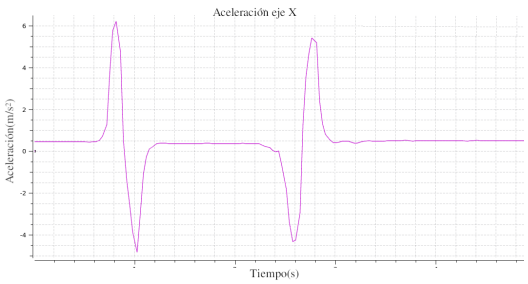

Figura 4.28: Aceleraciones en el eje X: dos gestos opuestos

Otro problema a la hora de detectar gestos surge cuando el mismo es demasiado "lento"; es decir, para los casos en que la aceleración y velocidad del dispositivo son considerablemente pequeños puede no ser posible clasificar el gesto con precisión, y en muchos casos esto se reflejará en una acción errónea (por ejemplo, accionando un movimiento del cursor en la dirección opuesta al gesto realizado). En la figura 4.29 se puede ver el comportamiento de la aceleración cuando se ejecuta un movimiento lento en la dirección positiva; el gesto transcurre hasta $t \approx 4{,}4\ s$. Para el tiempo restante se aprecia el ruido en las medidas cuando el dispositivo está quieto. Resulta evidente que tanto la aceleración devuelta por el sensor como la procesada carecen de una forma de onda definida, lo que hace imposible la correcta detección de un gesto a partir de allí. A su vez, muchos de los valores obtenidos están por encima del piso de ruido habitual y su variabilidad hace que la varianza calculada sea mayor al umbral impuesto; todo esto sumado genera que los valores de velocidad calculada disten de la realidad y por ende también su sumatoria, lo que se traduce en la detección errónea de gestos.

Otro de los problemas que decantan en un mal funcionamiento del dispositivo son los cambios bruscos de orientación (figura 4.30). Cuando esto ocurre, en la aceleración

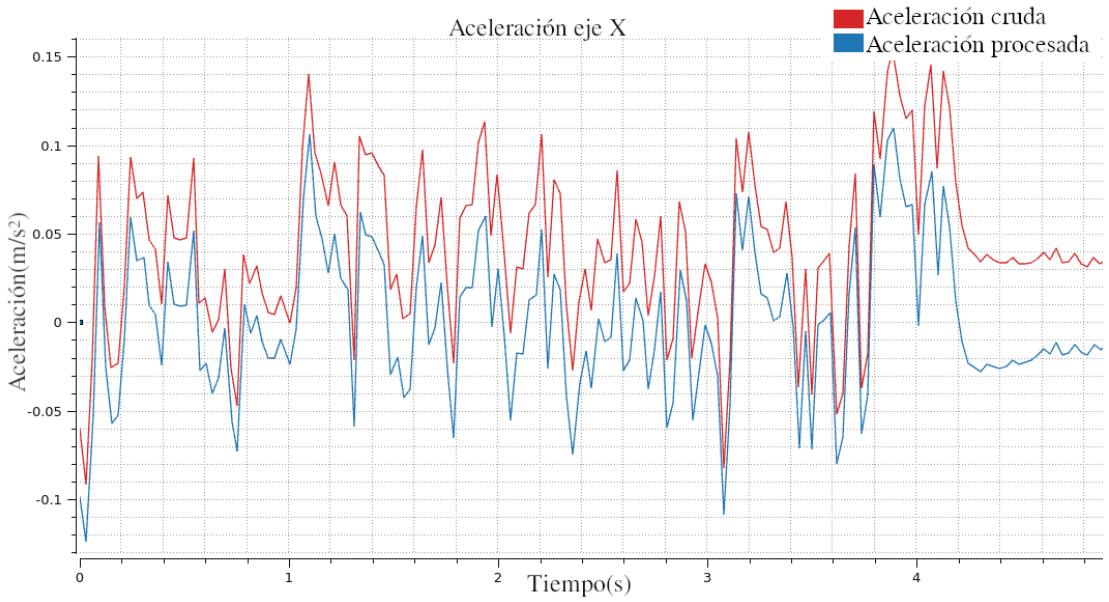

Figura 4.29: Aceleraciones eje X: gesto lento

cruda se observa un escalón (debido al cambio de la componente de g en ese eje) y la aceleración procesada tiene picos que, pese a que se extinguen rápidamente generan un aumento ficticio de la velocidad. A su vez estas variaciones generan que la varianza también aumente significativamente por un breve periodo de tiempo. La adición de todos estos elementos se traduce en que, en algunos casos, se detecten gestos erróneamente.

Es por todo lo expuesto hasta aquí que existen casos en los se detectan gestos inexistentes o se mal catalogan algunos otros, lo que genera que el dispositivo no funcione de manera óptima.

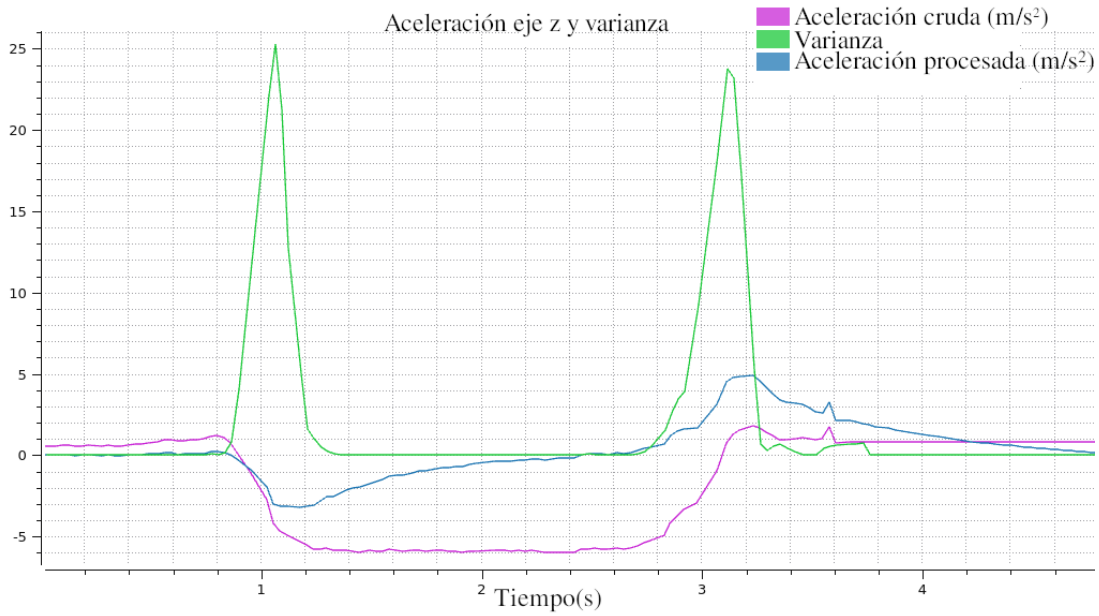

Figura 4.30: Varianza de aceleración en cambio de orientación

4.5.6. Traducción a Acción del Mouse

Hasta el momento hemos comentado con cierto detalle como se realiza la detección de los distintos gestos realizados con el dispositivo. Ahora es momento de ver como esos gestos definidos se mapean en acciones del *mouse*. Aquí es donde entra en juego la función *get_mouse_order()*; la lógica que vemos diagramada en la figura 4.31 indica como, una vez que se detectó un gesto en los ejes X y/o Y, se define hacia donde y cuán veloz se moverá el cursor mediante el manejo de los "pasos" (*mouse_step_x* y *mouse_step_y*); éstos, en su signo y magnitud, llevan la información de la dirección y la longitud del movimiento que hará el cursor del *mouse* con respecto a su posición anterior (movimiento relativo). Además de esto, vale observar que siempre que esté dada la orden de ignorar gestos en los ejes X e Y se fijarán estos pasos como cero, lo que generará que el cursor se mantenga quieto; es decir, siempre que se detecte un gesto en Z o un *shake*, el cursor será frenado. Esto sucede porque estas dos acciones son las únicas que tienen injerencia sobre la variable *ignore_gestures_xy*. Vale destacar que sólo se modifican *mouse_step_x* y *mouse_step_y* cuando hay un gesto detectado, por lo cual resulta claro que si se desencadenó cierto comportamiento en el movimiento del cursor, se mantendrá así hasta la detección de un nuevo gesto.

Una vez que el sistema determinó cual es el paso que deberá moverse el cursor, *mouse_control()* envía esa información vía *Bluetooth*, junto con las acciones del *click* en los casos en los que corresponda.

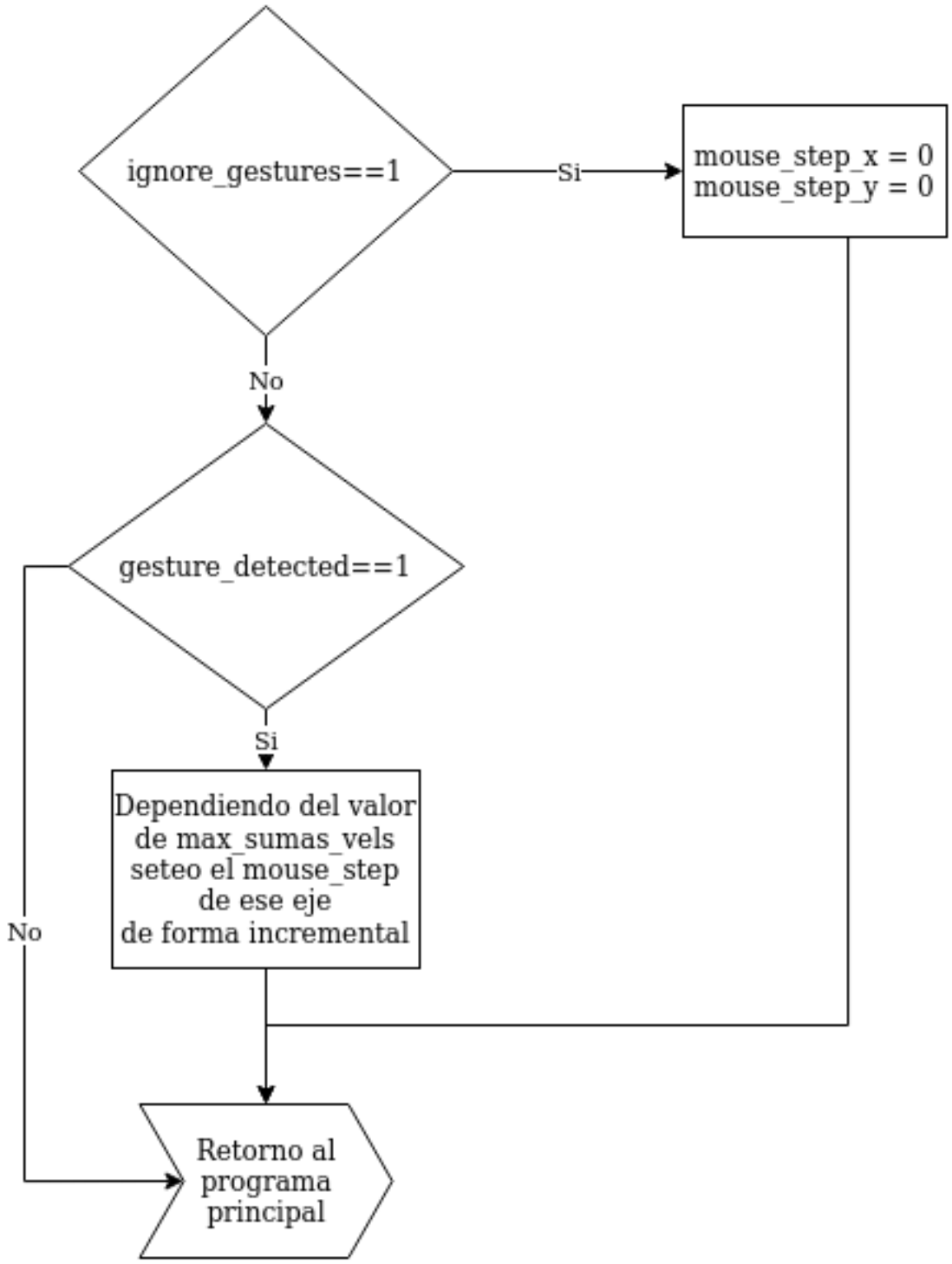

Figura 4.31: Diagrama de flujo de la función *get_mouse_order()*

Capítulo 5

Diseño del *Hardware*

En el siguiente capítulo se explica cómo fue llevado a cabo el diseño del *hardware* del dispositivo, así como también las especificaciones del mismo. En primer lugar se expone el proceso de elección de componentes, y para concluir se presenta el diseño de los circuitos impresos (*PCB* por su sigla en inglés) del dispositivo.

5.1. Prototipo de Desarrollo

Para una primera etapa de implementación, depuración y validación del sistema, se realizó un ensamblaje de distintas placas de desarrollo y circuitos integrados disponibles en el mercado. A continuación se detallará la elección de dichos componentes.

5.1.1. Comunicación Inalámbrica y Procesador

Una parte fundamental del sistema es la comunicación inalámbrica. Para su implementación se consideraron las tecnologías que se manejan en el mercado para este tipo de aplicaciones:

- *SubRF* (Frecuencia $< 1\ GHz$)
- *Bluetooth* (Frecuencia en el entorno de los 2,4 GHz)

Inicialmente se realizó una preselección de componentes eligiendo un representante de cada tecnología (tabla 5.1), basándose en los requerimientos antes mencionados en el Capítulo 1 y poniendo especial énfasis en el transmisor (ya que por ser un dispositivo inalámbrico el consumo puede ser determinante).

Nombre	**Protocolo**	**Frecuencia**	**Consumo(ON)**	**Potencia(Salida)**	**Extras**
CC1150 [20]	*SubRF*	Hasta 500 kBaud	14.1 mA @(0 dBm, 315 MHz)	Hasta 10 dBm	Fifo 64 Bytes, LPM
nRF52832 [30]	*Bluetooth*	Hasta 2 Mbps	5.3 mA corriente de pico @(0 dBm)	Hasta 4 dBm	*uC* ARMCortex-M4,LPM

Tabla 5.1: Comparativa de opciones inalámbricas

Analizando brevemente los componentes mencionados, por un lado vemos que uno de ellos (*CC1150*) se destaca por una potencia de transmisión considerablemente mayor,

reflejándose en un alcance más amplio. La clara desventaja de la tecnología asociada a este componente (*SubRF*) es que se torna necesario la inclusión de un receptor en el dispositivo de destino (*PC* o dispositivo móvil). En materia de consumo se destaca la placa *nRF52832* que brinda un módulo *Bluetooth* de baja energía (o *BLE* por su sigla en ingles). Vale notar que si bien la potencia máxima de transmisión de este integrado es menor a la del *CC1150*, la misma es estándar en el protocolo *Bluetooth*. Dado que esta tecnología es usualmente utilizada en periféricos *HID* comerciales, se considera que la potencia es suficiente. Además esta tecnología brinda cierta flexibilidad ya que es soportada en todos los dispositivos móviles y computadoras actuales (e incluido en la mayoría de ellos), permitiendo prescindir de la inclusión de un receptor externo. A su vez, en este componente mencionado (*nRF52832*) tenemos también un procesador *ARMCortex-M4* [29], lo que le brinda cierta ventaja frente al otro.

Analizando la placa *nRF52832* en detalle, nos encontramos con las siguientes características:

- Procesador *ARMCortex-M4*.
  - 64 *Kbyte* de RAM.
  - 512 *Kbyte* de Flash.
  - Reloj de 64 MHz.
  - Unidad de punto flotante (FPU).
  - División por *hardware*.
- Módulo *Bluetooth* de bajo consumo.

Considerando el gran volumen de operaciones matemáticas que serán requeridas en esta aplicación, se valora que el procesador posea la capacidad de realizar operaciones en punto flotante, así como división por *hardware*.

Es por lo antes expuesto que se utiliza la tecnología *Bluetooth* para realizar las comunicaciones inalámbricas, y la placa *nRF52832* para el desarrollo del *firmware*.

5.1.2. *MARG*

Uno de los componentes principales para el desarrollo del proyecto es el *MARG* (*Magnetic, Angular Rate, and Gravity*), basado en tres sensores: magnetómetro, acelerómetro y giroscopio. Para este caso se analizaron varias placas de desarrollo, realizando una comparativa de las principales características como se puede ver en la tabla 5.2.

Nombre	Fondo de escala			Resolución	Corr. Máx
	Acelerómetro	Giroscopio	Magnetómetro		
9DOF RAZOR IMU [40]	±2/±4/±8/±16 g	±125 a ±2000 DPS	±4,8 mT	16 bits,14 bits magnetómetro	6,8 mA
BMX055 [27]	±2/±4/±8/±16 g	±125 a ±2000 DPS	1,3/1,5 mT	10 bits, 0,3 μT	10 mA
MPU-9250 [23]	±2/±4/±8/±16 g	±125 a ±2000 DPS	±4,8 mT	16 bits,14 bits magnetómetro	3,7 mA
LSM9DS1 [3]	±2/±4/±8/±16 g	±245 a ±2000 DPS	±0,4/±0,8/±1,2/±1,6 mT	16 bits	4,6 mA
LSM6DSOX + LIS3MDL [2]	±2/±4/±8/±16 g	±125 a ±2000 DPS	±0,4/±0,8/±1,2/±1,6 mT	16 bits	0,82 mA

Tabla 5.2: Comparativa placas de desarrollo *MARG*

Como se puede notar, en lo que respecta a los rangos y la resolución no existe demasiada diferencia, a excepción del magnetómetro; considerando la aplicación deseada (campo magnético terrestre $\approx 25 - 65\ \mu T$), se prefieren aquellos con menor fondo de escala. En el ámbito del consumo, la diferencia es un tanto más notable, donde existe un factor mayor a cuatro entre el consumo mínimo y el siguiente ($\frac{3{,}7\ mA}{0{,}82\ mA} = 4{,}5$). Teniendo en cuenta este aspecto, se utiliza la placa *Adafruit LSM6DSOX + LIS3MDL - Precision 9 DoF IMU* [2], compuesta de un *IMU* (por sus siglas en ingles, *Inercial Measurement Unit*) *LSM6DSOX* [44] y un magnetómetro *LIS3MDL* [43]. Las características de estos componentes se detallan a continuación:

- LSM6DSOX
  - Acelerómetro: $\pm 2/\pm 4/\pm 8/\pm 16$ g, con frecuencia de actualización entre 1,6 Hz y 6,7 KHz
  - Giroscopio: $\pm 125/\pm 250/\pm 500/\pm 1000/\pm 2000$ DPS, con frecuencia de actualización entre 12,5 Hz y 6,7 KHz
  - Consumo: 0,55 mA con acelerómetro y giroscopio en *High Perfomance*
- LIS3MDL
  - Magnetómetro: fondo de escala configurable entre $\pm 4/\pm 8/\pm 12/\pm 16$ *gauss*
  - Frecuencia de actualización entre 0,625 Hz y 1000 Hz
  - Generador de interrupciones
  - Consumo: 0,27 mA con el magnetómetro en *Ultra High Resolution* y *ODR* (*Output Data Rate*) de 20 Hz

5.1.3. Botones

Como fue mencionado, algunos de los modos descritos en la Sección 2.2 requieren de la presencia de botones para su funcionamiento. Los botones mecánicos convencionales pueden presentar dificultades para ser presionados, y no cuentan con sensibilidad configurable. Es por esto que para el diseño se define el uso de sensores de proximidad que cumplan la función de interfaz entre el usuario y el sistema.

Debido a que la implementación de los botones se puede realizar de varias formas, previo a comparar distintos sensores de proximidad es necesario realizar una comparativa de tecnologías; en la tabla 5.3 podemos ver una comparación preliminar, mencionando algunas de las características principales.

Como se puede apreciar, algunas de las tecnologías mencionadas requieren que la aproximación al sensor se realice con un material en particular; esto genera una clara desventaja ya que se busca minimizar la complejidad en su uso. Por otra parte, el valor mínimo de detección que brindan los sensores de ultrasonido ($\approx 0{,}1\ m$ [28]) no resulta adecuada para la realización de botones, lo que lleva a descartar esta tecnología.

Teniendo en cuenta esto, se determinó que para el sensor de proximidad deberían utilizarse sensores capacitivos u ópticos. Con esto en mente, en la tabla 5.4 se realiza una comparativa de los dos sensores seleccionados (uno por cada tecnología).

Tecnología	**Distancias detectables**	**Materiales detectados**
Inductivos	$< 80\ mm$	Piezas metálicas
Capacitivos	$< 160\ mm$	Cualquier material que afecte la capacidad (e.g. la piel)
Magnéticos	$< 100\ mm$	Piezas imantadas
Sensor de luz infrarroja de corto alcance	$< 130\ mm$	Cualquier material
Sensor de ultrasonido	$< 15\ m$	Cualquier material

Tabla 5.3: Comparativa de tecnologías de proximidad

Nombre	**Tecnología**	**Consumo**	**Robustez frente a interferencias**	**Extras**
AT42QT1011 [42]	Capacitivo	400 μA	Filtro y compensación	LPM, sensibilidad configurable
MAX44000 [21]	IR	11 μA (promedio)	Rechazo de ruido IR	LPM, sensor de luz

Tabla 5.4: Comparativa de placas de desarrollo de sensores de proximidad considerados

A priori, y considerando las características expuestas, no existe una gran diferencia entre ambos componentes. Como una ventaja notable encontramos que el sensor capacitivo (incluido en la placa de desarrollo *SparkFun Capacitive Touch Breakout - AT42QT1011* [42]) permite definir las dimensiones del electrodo de sensado, consiguiendo así que el botón no deba restringirse únicamente al espacio abarcado por el módulo, lo cual brinda una mayor flexibilidad a la hora de la fabricación de los botones. A su vez, la detección de proximidad está resuelta en el integrado, dando la salida en formato de pin digital. Es por estos dos factores que se selecciona este componente sobre el *MAX44000*.

5.1.4. Memoria Externa

Considerando la necesidad de tener persistencia en algunos datos (tales como calibraciones o configuraciones iniciales) resulta necesaria la disponibilidad de una memoria no volátil. Si bien el procesador seleccionado posee una memoria con esta característica, se optó por la inclusión de una memoria externa. Esto se debe mayoritariamente a una decisión de diseño en pos de preservar la memoria incluida en el procesador (minimizando los ciclos de escritura), incorporando una memoria externa que puede ser removida y remplazada en caso de ser necesario.

Debido a que no existen demasiadas restricciones sobre este componente, se optó por una memoria *EEPROM M95320* [45], cuyas características son:

- Protocolo de comunicación *SPI*.
- Capacidad: 32 $Kbits$.
- Tiempo de escritura: 5 ms.

5.1.5. Componentes Seleccionados

A modo de resumen de lo expuesto anteriormente en esta sección, en la tabla 5.5 se detallan los componentes seleccionados para la realización del prototipo de desarrollo.

Hardware	Componente seleccionado
MARG	LSM6DSOX + LIS3MDL 9 DOF IMU [2]
Com. inalámbrica + microcontrolador	FEATHER NRF52 BLUEFRUIT LE - NRF [30]
Sensor capacitivo	CAPACITIVE TOUCH BREAKOUT AT42QT [42]
Memoria externa	IC EEPROM 32K SPI 20MHZ 8SO [45]

Tabla 5.5: *Hardware* requerido para el prototipo

5.1.6. Resultados

Una vez que se tienen seleccionados los componentes, solo resta integrarlos para obtener un prototipo sobre el cual se realizan las implementaciones y pruebas correspondientes. En la figura 5.1 podemos ver uno de los prototipos ensamblados, remarcando y etiquetando cada uno de los componentes que lo integran. La versión que vemos allí contiene lo necesario para los modos Gestos y Cabeceo, ya que el único sensor que se incluye es el *MARG* y su tamaño permite que este prototipo sea vestible. Además se incluye un botón pulsador y dos conectores de audio *Plug TRS* de 3,5 *mm* (tres señales), para la realización tanto de pruebas de los modos mencionados, como para ensayos preliminares de los modos Barrido y Botonera.

Por otro lado, también se diseñó un prototipo exclusivo para los modos de Botonera y Barrido; en la figura 5.2 podemos ver una foto del dispositivo, que consta únicamente de la placa *nRF52832*, seis conectores *Plug* de 3,5 *mm* con sus señales conectadas a pines digitales, y una batería e interruptor para brindar autonomía. En este caso los conectores de audio están conectados a *VCC*, *GND* y un pin digital.

En la figura 5.3 podemos apreciar los botones diseñados para esta prueba de concepto; como se puede ver, la parte superior de los mismos está cubierta por una hoja de aluminio que hace las veces de electrodo de sensado. Tanto la señal de salida como la alimentación van a el conector de audio. De este modo, utilizando un cable de audio *Jack* macho-macho se interconectan el prototipo de desarrollo y los botones para los modos en que estos sean necesarios.

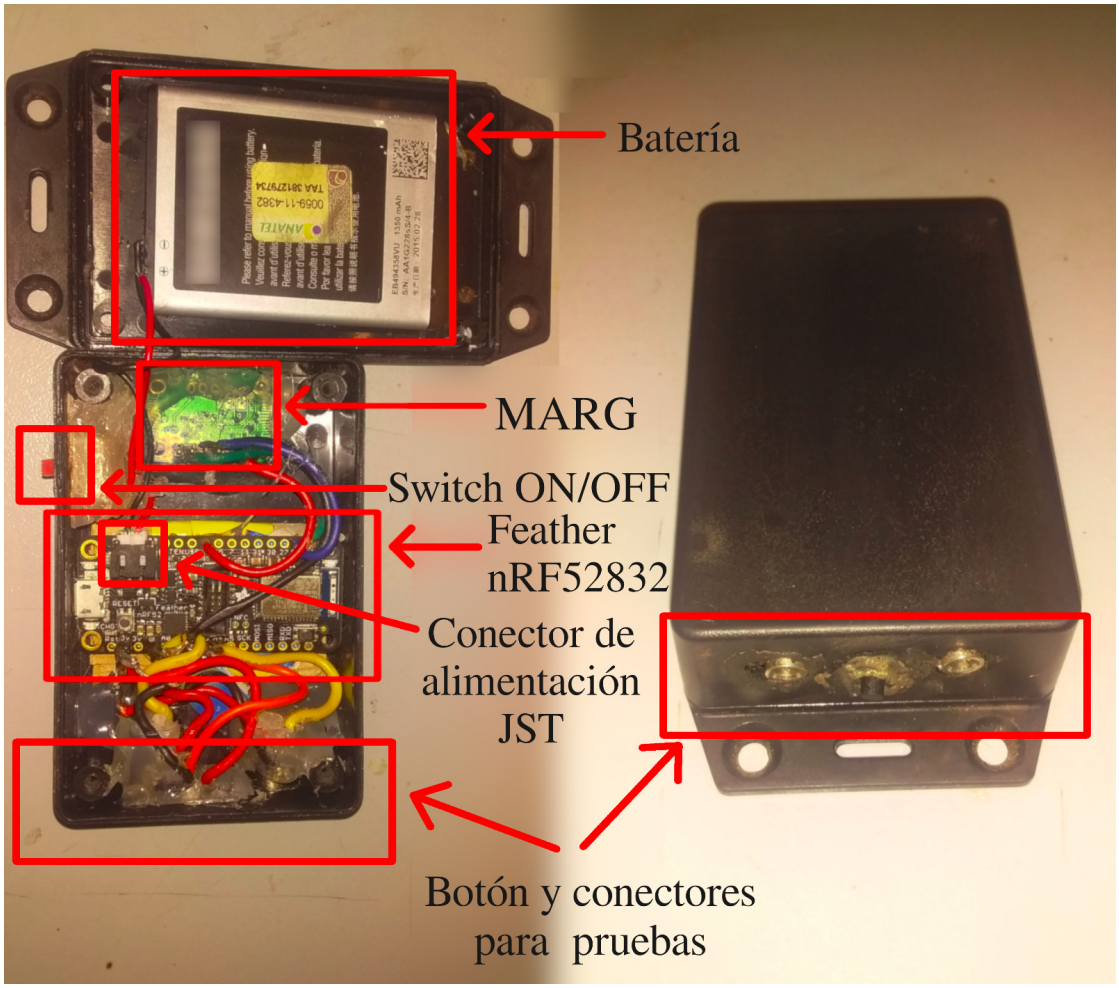

Figura 5.1: Prototipo de desarrollo con *MARG*

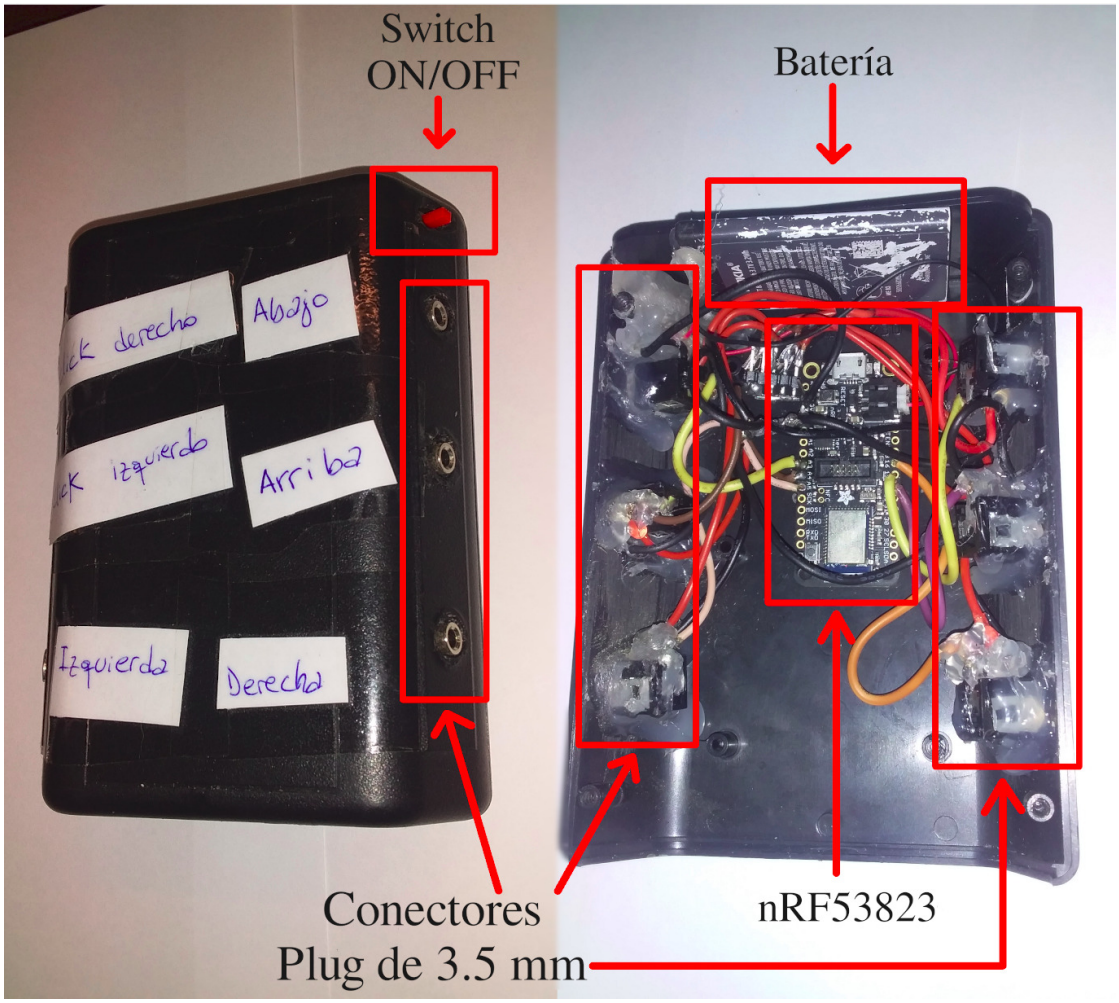

Figura 5.2: Prototipo de desarrollo para botones

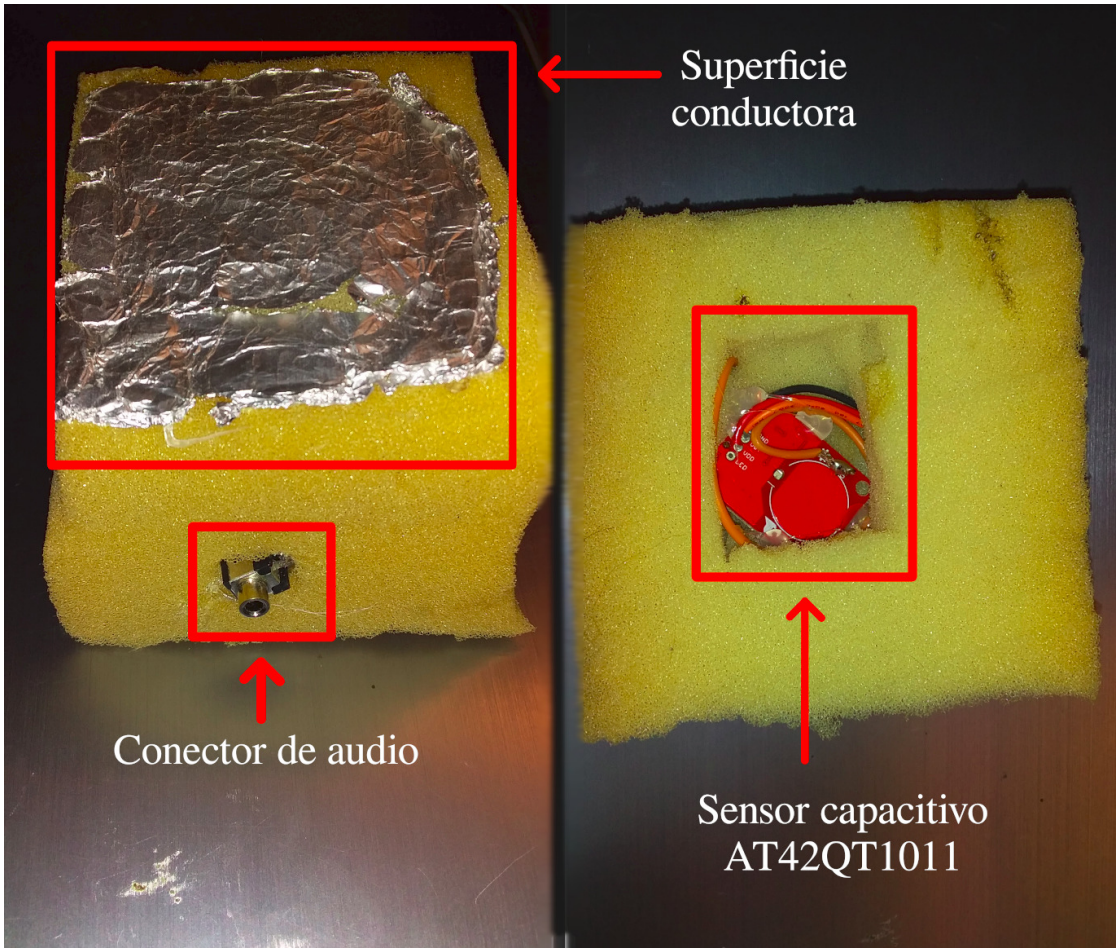

Figura 5.3: Prototipo de botones

5.2. Desarrollo de *PCB*

Una vez validada la prueba de concepto se procede al diseño de placas de circuito impreso (*PCB* por su sigla en ingles) con el fin de disminuir el tamaño del dispositivo final. Como fue descrito en la Sección 2.2, los modos Cabeceo y Gestos requieren que el dispositivo sea vestible, cosa que no ocurre en los modos que utilizan botones para su funcionamiento (Barrido y Botonera). Teniendo en cuenta esto y que para el conexionado de los botones se utilizaron conectores de audio *Plug* de 3,5 *mm* (que ocupan una gran superficie), se opta por separar el desarrollo en dos placas: por un lado, una conteniendo el *MARG*, procesador, memoria, regulador de voltaje y circuito de carga (en resumen, todo lo necesario para los modos de funcionamiento que no requieren botones), a la cual nos referiremos como placa Principal. Por otro lado, atendiendo a las necesidades de los modos que se basan en botones para su funcionamiento, se desarrolla un segundo *PCB* que extiende las capacidades de la placa Principal, para que pueda interactuar con botones. Esta segunda placa cuenta únicamente con los sensores capacitivos y los componentes pasivos para que estos funcionen, pero requiere ser alimentada externamente. Esta placa será referida como Botonera.

A lo largo de esta sección se detallará el proceso de diseño de dichas placas, así como los resultados obtenidos y los problemas encontrados.

5.2.1. Desarrollo de la Placa Principal

Luego de validar el prototipo vestible, el objetivo pasa a ser lograr una versión final de dimensiones reducidas que pueda ser usada en alguna parte del cuerpo. Para esto se parte de la configuración validada con las placas de desarrollo para diseñar un *PCB* que cuente únicamente con los componentes indispensables:

- *IMU*: LSM6DSOX [44].
- Magnetómetro: LIS3MDL [43].
- Microcontrolador+*Bluetooth*: MDBT42Q [35].
- Memoria externa: M95320-W [45].
- Regulador de voltaje lineal de 3,3 V: AP2112 [7].
- Circuito de carga: MCP73831 [26].
- Resistencias, capacitores, inductores.

Teniendo en cuenta el prototipo validado incluía la placa de desarrollo *Feather nRF52832*, se mantiene el chip *MDBT42Q*, el cual será el encargado del procesamiento y la comunicación *Bluetooth*. Con el objetivo de minimizar la superficie de la placa Principal se elimina el conversor *USB-Serial* incluido en la placa de desarrollo *Bluefruit52*; este *chip* es útil en etapas de desarrollo, ya que permite programar la placa mediante *USB*, sin necesidad de utilizar un programador externo. Sin embargo, al contar con un medio para programar el *chip* (dado que el mismo cuenta con una interfaz JTAG/SWD) utilizando J-Link EDU [36], se puede prescindir del mismo; es para esto que se dejan

expuestos en formato de *PAD* los cuatro pines necesarios para programar la placa (*SWD*, *SWCLK*, *VDD*, y *VSS*).

Tanto el *IMU* como el magnetómetro se comunican con el microcontrolador mediante el protocolo I^2C, el cual precisa que tanto la señal de datos como la de reloj tengan una resistencia de *pull-up*. La corriente máxima por estas señales se da cuando las mismas valen cero y vale

$$I_{max} = \frac{3{,}3\ V}{R_{pull-up}}.$$

Considerando que la corriente máxima de los pines de entrada y salida del microprocesador es de 25 mA se fija $R_{pull-up} = 10\ k\Omega$ lo que da una corriente $I_{max} = 330\ \mu A$, lo cual cumple ampliamente con la restricción.

Dado que el dispositivo debe funcionar de forma autónoma, es necesario tener en cuenta ciertas consideraciones para agregar una batería. Para lograr esto se mantiene en el diseño el conector *USB*, circuito de carga y regulador de voltaje originales de la placa de desarrollo *Bluefruit52*. De esta forma, todos los componentes se alimentan con los 3,3 V de salida del regulador de voltaje, a excepción del circuito de carga, que se alimenta de la tensión del *USB* (5 V). El componente que implementa el circuito de carga (MCP73831) permite configurar la corriente a la que se carga el dispositivo según el valor de una resistencia. La corriente de carga queda determinada por

$$I_{carga} = \frac{1000\ V}{R_{PROG}}.$$

A efectos de asegurar compatibilidad con la corriente máxima soportada por la batería a seleccionar se fija R_{PROG} en 10 $k\Omega$, lo que determina una corriente de carga de $I_{carga} = 100\ mA$.

Dado que el máximo de botones que se soporta es seis (en modo Botonera), y considerando que el *PCB* Principal es el que alimenta al destinado a los botones, es necesario conectar ocho señales; para esto se opta por un conector *RJ45*. Si bien hay alternativas a la conexión que ocupan menos superficie (y por lo tanto disminuyen el tamaño final de la placa), se toma esta decisión pensando en la experiencia del usuario, ya que este conector hace imposible conectar de forma errónea las placas, a la vez que ofrece robustez frente a desconexiones involuntarias.

Para determinar el *layout* de los componentes en la placa, es necesario referirse a las hojas de datos de los mismos. Allí se encuentran las consideraciones del fabricante sobre cómo

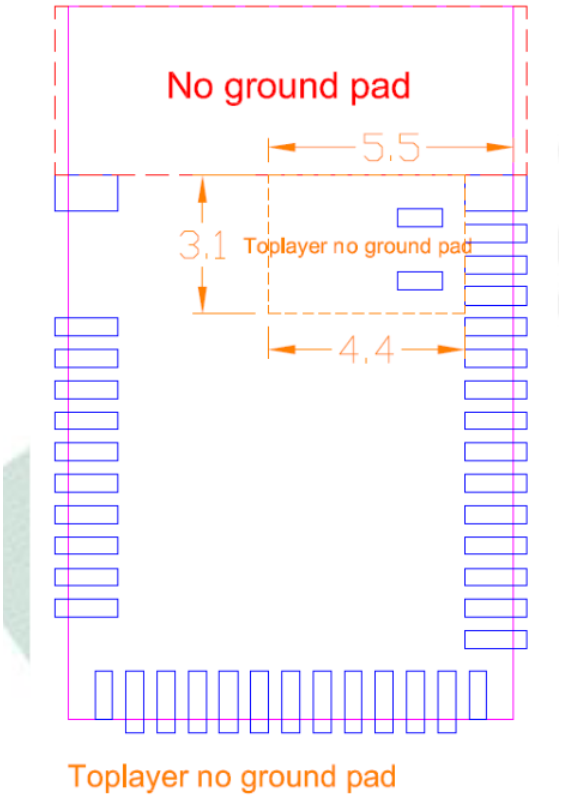

Figura 5.4: *Layout* sugerido para MDBT42Q

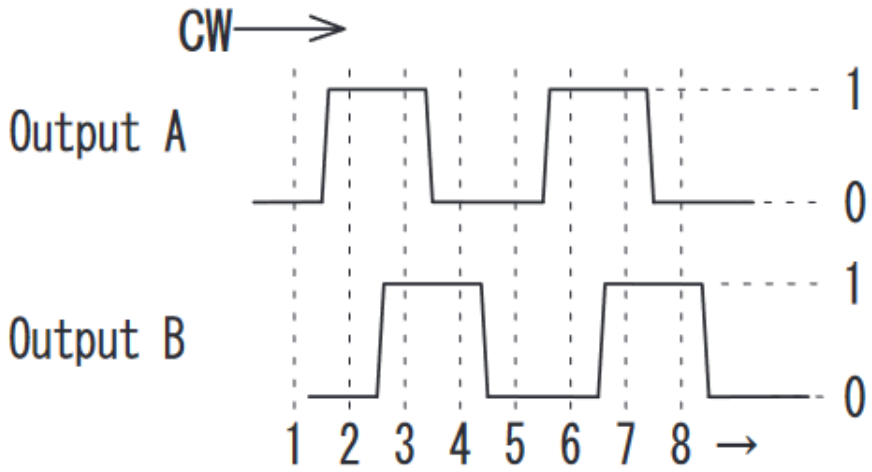

Figura 5.5: Salida de los pines del *encoder* al ser girado en sentido horario

o dónde ubicar los componentes en la
placa, así como su posición relativa a otros componentes. Sobre estas restricciones, el *chip* compuesto por el procesador y la solución inalámbrica (*MDBT42Q*) impone que la antena no se encuentre en un valle de *GND*, tal como se puede ver en la figura 5.4.

Encoder

A efectos de permitir el cambio entre modos, se agrega en esta placa un *encoder* rotatorio *EN11-HSM1BF20* [17]. El mismo cuenta con un botón y además, cada vez que es girado genera en dos de sus pines un pulso cuadrado, de forma tal que si lo consideramos girando continuamente obtendríamos dos señales desfasadas un cuarto de período, como las de la imagen 5.5.

Se puede observar que la salida **A** está adelantada a la salida **B**. Esto vale para todos los giros en sentido horario; mientras que para los giros en sentido antihorario la salida **B** es la que está adelantada.

De esta forma, basta con mirar el valor de la señales en un flanco de una de ellas (puede ser de subida o bajada) para determinar el sentido del giro. Por ejemplo si miramos en los flancos de bajada de la salida **A** y las señales son iguales, implica que el giro fue en sentido antihorario, mientras que si las señales son distintas el sentido fue horario.

Layout y placa fabricada

Antes de hablar del *layout* de la placa Principal corresponde presentar el esquemático correspondiente. Para facilitar la lectura, el mismo se presenta por partes. En la figura 5.6 se puede ver el esquemático del procesador, así como sus señales. Además del oscilador externo se remarca la presencia de un bloque CL entre los pines *DEC4* y *DCC* cuya función es utilizar el conversor DC/DC interno del procesador. Se ve también un

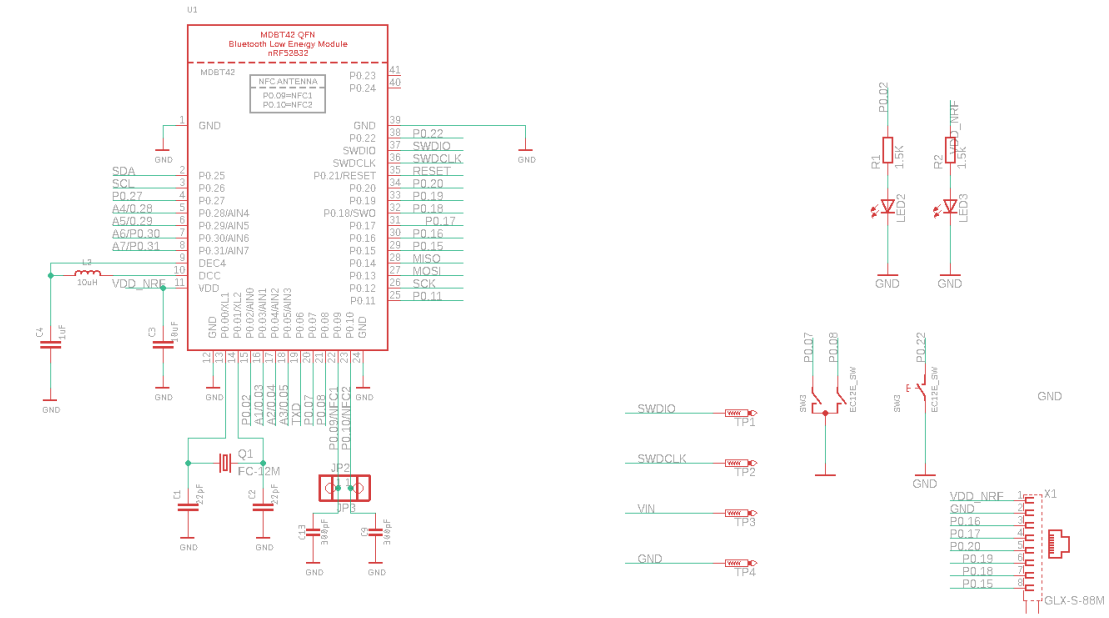

Figura 5.6: Esquemático correspondiente al procesador

capacitor cerámico de desacople (C3); estos capacitores serán un factor común a lo largo de los bloques presentados. También se puede notar el conexionado del *encoder*, el *RJ45* y las señales necesarias para la programación del dispositivo.

En la figura 5.7 se puede observar el esquemático correspondiente al *MARG*, en el mismo se pueden apreciar los capacitores cerámicos de desacople (C5 y C6) así como las resistencias de *pull-up* necesarias para el correcto funcionamiento del protocolo I^2C.

Por último en las figuras 5.8 y 5.9 se pueden ver respectivamente el circuito de carga y el regulador de voltaje. En el circuito de carga vemos el capacitor cerámico de desacople (C10) y la resistencia que permite fijar la corriente de carga (R3). Vale mencionar que dado que este este *chip* es el único que funciona con 5 V, y por lo tanto podría ser conectado a la red eléctrica, se agrega un diodo *Zenner* a la entrada a modo de protección contra sobretensiones. Sobre el regulador de voltaje el único comentario además de los capacitores de desacople viene por el *switch ON/OFF*, el cual está lo más "aguas arriba" posible a efectos de eliminar consumos parásitos a la batería cuando el dispositivo está apagado. Por último, no se agregan esquemáticos de la memoria, ni del conector *USB* ya que simplemente tienen conectadas las señales correspondientes.

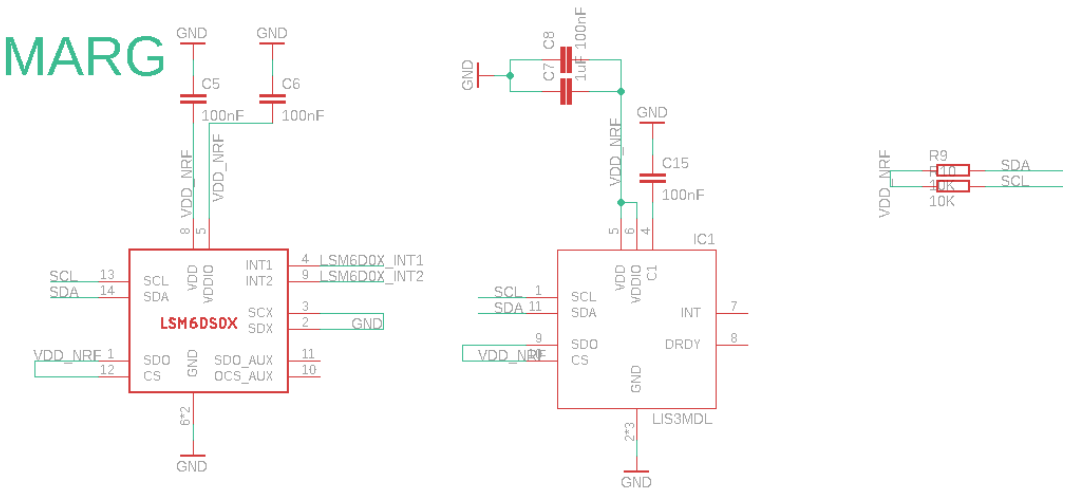

Figura 5.7: Esquemático correspondiente al *MARG*

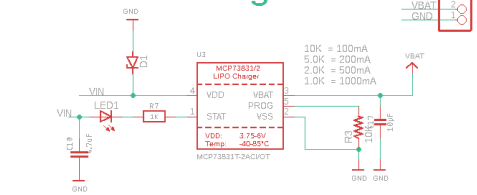

Figura 5.8: Esquemático correspondiente al circuito de carga

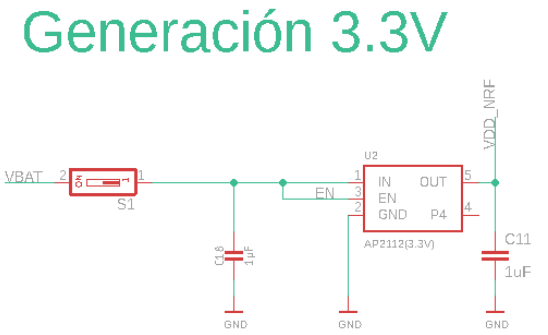

Figura 5.9: Esquemático regulador de Voltaje

Finalmente, en la figura 5.10 se presenta el *layout* final de la placa, donde se puede notar la zona restringida correspondiente a la antena, mientras que en la figura 5.11 se hace lo propio con la placa fabricada, realizando la señalización de los componentes y dimensiones.

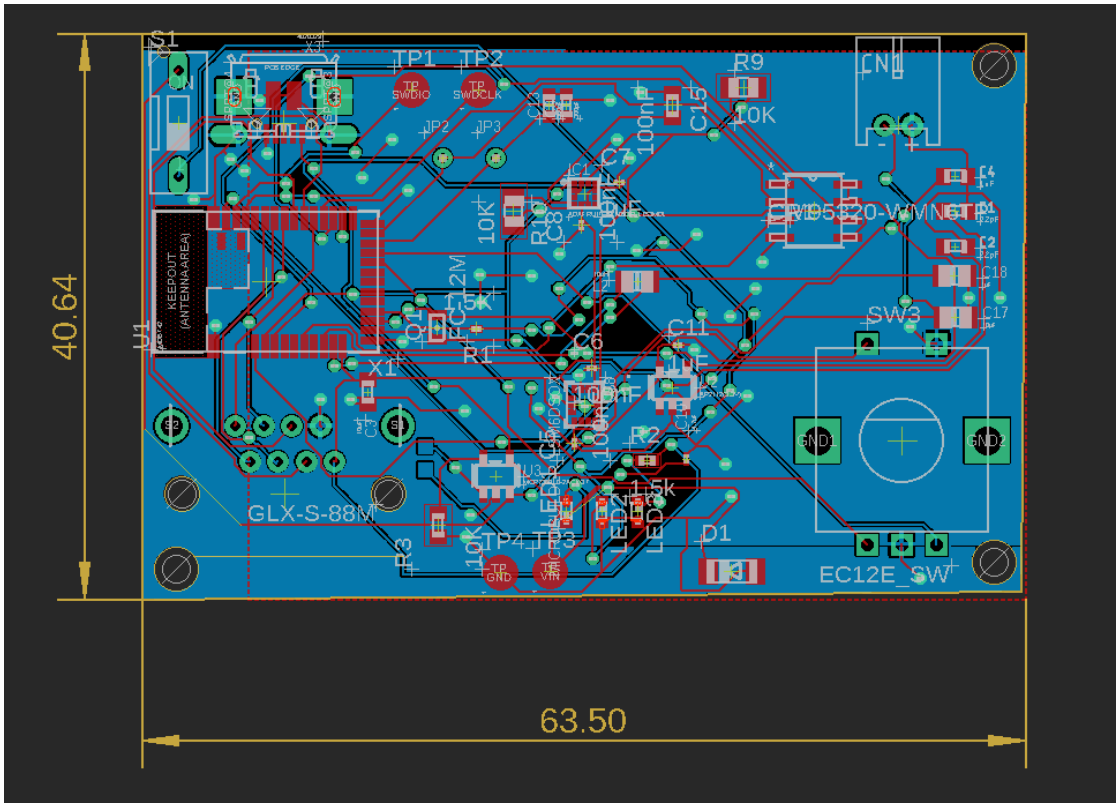

Figura 5.10: *Layout* correspondiente al dispositivo Principal

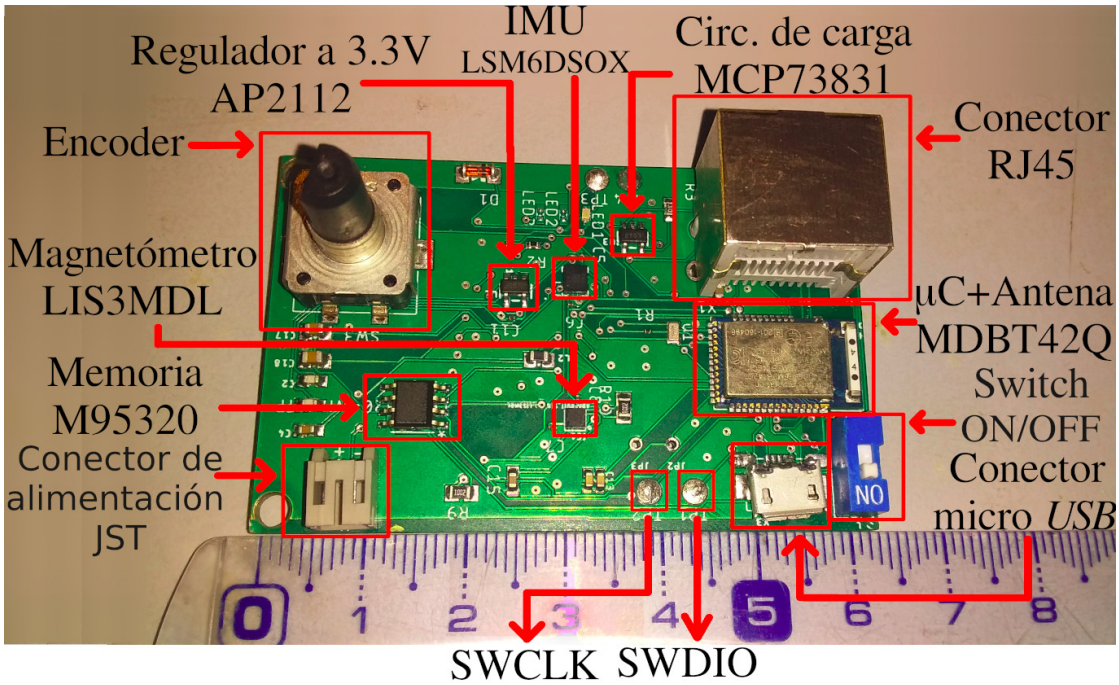

Figura 5.11: *PCB* correspondiente al dispositivo Principal

5.2.2. Desarrollo de la Placa Botonera

La placa Botonera contiene los seis sensores capacitivos (*AT42QT1011*), así como los conectores que vinculan este *PCB* con los botones; en la etapa de validación (particularmente en las interacciones con el cliente) del prototipo se observaron botones cuya conexión se realiza mediante un *Plug* de 3,5 mm, por lo que se decide conservar este criterio, a efectos de ser compatible con soluciones ya existentes.

En la figura 5.12 se presenta el esquemático correspondiente a un botón, en el que se puede ver el integrado encargado del sensor capacitivo, así como los *jumpers* que permiten configurar su sensibilidad. También se pueden apreciar el selector entre botón capacitivo y señal digital (S1) y el conector de audio (DCJ0202 [14])

Para el soporte de botones de diversas formas y tamaños es necesario referirnos a la imagen 5.13, que muestra la configuración básica del sensor capacitivo. La sensibilidad del sensor es proporcional tanto a C_s como a C_x (según los nombres de la figura), siendo la segunda la capacidad del electrodo conectado (botón en este caso). Por lo tanto, para poder calibrar la sensibilidad apropiada según la forma y tamaño del botón, se tienen en cuenta dos factores: por un lado, el rango válido para C_s ($2\ nF - 50\ nF$ [24]) y por otro, en las pruebas realizadas utilizando la placa *SparkFun Capacitive Touch Breakout - AT42QT1011* ($C_s = 10\ nF$) se observó que la sensibilidad era suficiente para el tamaño de botones diseñados. Considerando esto, C_s resulta ser el equivalente de tres capacitores en paralelo, de valores $2\ nF$, $3\ nF$ y $4\ nF$, seleccionables mediante *jumpers*. De esta forma se puede fijar C_s como cualquier combinación de estos tres valores; a mayor capacidad, mayor sensibilidad para un botón dado. Así es posible ajustar la sensibilidad de forma acorde a sus necesidades del usuario.

A su vez, se agregó un *switch* por cada entrada, que permite seleccionar si la misma es un electrodo a ser conectado al sensor capacitivo o un botón digital; en el segundo caso se lleva la señal directamente al microcontrolador y se inhabilita temporalmente el sensor capacitivo.

Considerando que la placa Botonera no fue diseñada para ser vestible, no se priorizó el tamaño de la placa. El *PCB* resultante lo podemos ver en la figura 5.14; en la misma se marca cada componente dentro de un bloque (azul) que está compuesto por la electrónica correspondiente a un solo botón por lo cual, como es natural, se encuentra repetido seis veces en la placa.

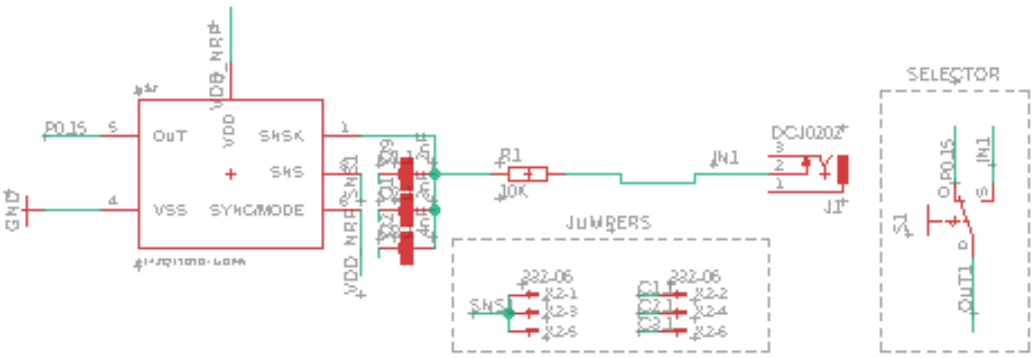

Figura 5.12: Esquemático correspondiente a un botón

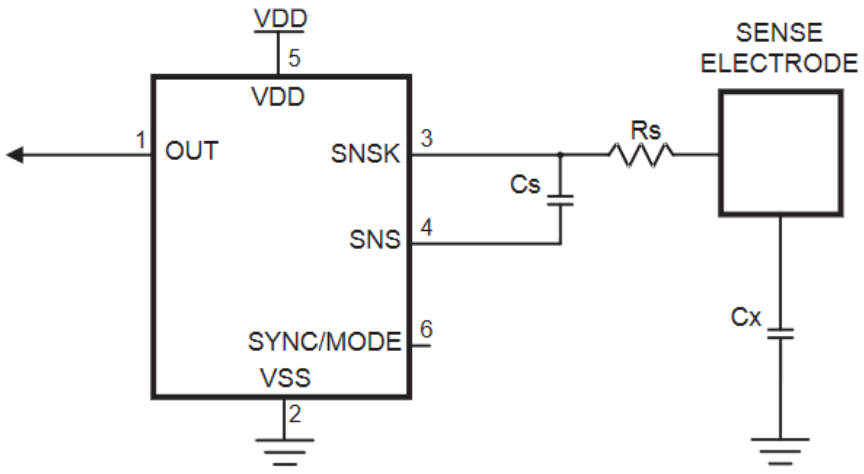

Figura 5.13: Configuración básica sensor capacitivo

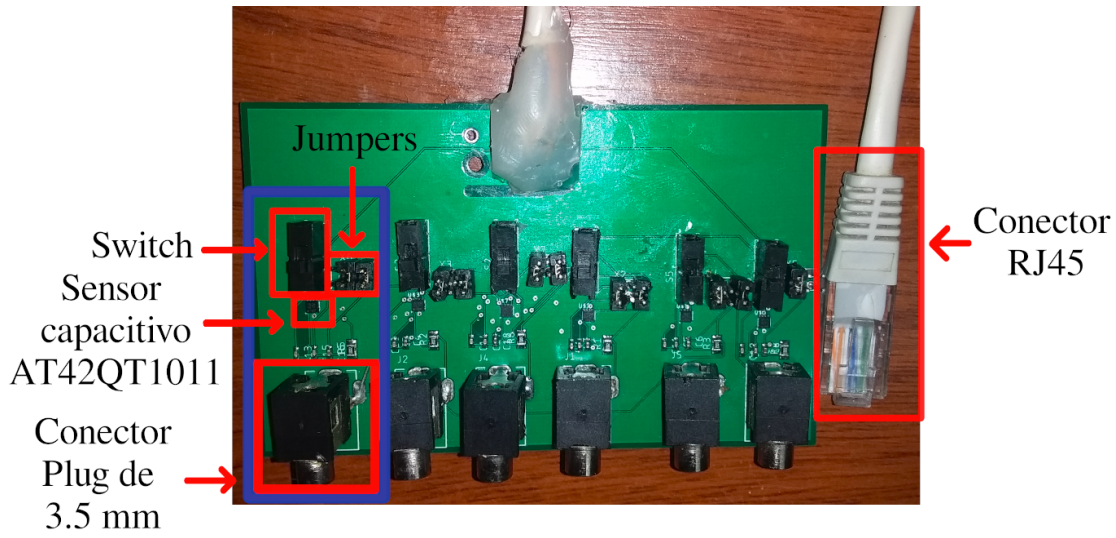

Figura 5.14: *PCB* correspondiente al dispositivo secundario

5.2.3. Errores en los Diseños de los *PCB*

En el proceso de diseño de los *PCB* se cometieron errores debidos a la falta de experiencia en estas tareas. Algunos de ellos fueron resueltos modificando las placas, y los que no pudieron ser solucionados no afectan en demasía la funcionalidad del dispositivo. A continuación se detallan estos errores y qué medidas se tomaron al respecto.

Placa Principal

Como se mencionó en la Subsección 5.2.1 para el integrado *MDBT42Q* se consideraron las sugerencias del fabricante, en particular no colocar la antena en un valle de *GND*, como muestra la figura 5.4. Una sugerencia análoga es dada por el fabricante del *chip MCP73831* (figura 5.15), que propone ubicar vías en un radio del componente a efectos de disipar el calor del mismo. Esto no fue notado hasta luego de fabricar las placas y resulta ser un problema a tener en cuenta, pudiendo generar daños en este componente u otros aledaños.

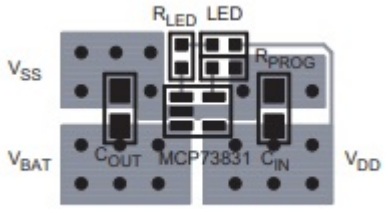

Figura 5.15: *Layout* sugerido por el fabricante para el *chip MCP78321*

Por su parte, el conector de alimentación *JST* está ruteado con la polaridad invertida por lo que previo a conectar el dispositivo con una batería es necesario rotar el conector 180°.

Otro problema encontrado en el diseño de esta placa es que el *footprint* de los *LED* en el *layout* no coincide con las dimensiones de los *LED* comprados; esto se debió a que la librería de la que se obtuvo el *footprint* especifica las dimensiones en el sistema Métrico (Americano) pero sin mencionarlo explícitamente, y en el desarrollo se consideró que todos los *footprints* se daban en sistema Imperial (Británico).

Botonera

Como se puede apreciar en la figura 5.14, el conector *RJ45* hembra de la placa no está presente. Esto se debe a que dicho componente fue orientado de forma errónea, quedando rotado 180° respecto a lo deseado, lo que produce que conectar el extremo macho sea imposible. Por esto, se optó por aprovechar las vías destinadas a fijar el conector para soldar los cables correspondientes a las señales. Además en esta placa faltó conectar *VCC* y *GND* a los *Plug* de 3,5 *mm*, por lo que según el diseño que se fabricó no se podrían alimentar botones digitales; sin embargo esto se solucionó fácilmente ya que estos conectores son *Through-hole* lo que simplifica soldar un cable desde las señales de interés. Otro problema encontrado en esta placa es que faltó agregar agujeros para montar la

misma al encapsulado; esto se solucionó realizando los agujeros de forma manual. Por último resta realizar un comentario referente a los bloques por botón; contrario a lo deseado, no en todos los casos el conector, *switch* y banco de capacitores alineados corresponden al mismo sensor. Si bien esto no tiene impacto sobre el funcionamiento del sistema, dificulta la configuración de sensibilidad de los botones.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 6

Diseño Mecánico del Dispositivo

En este capítulo se detalla el diseño de los encapsulados que integran al dispositivo. Durante el proceso de diseño de estos encapsulados la estructura general se mantuvo sin grandes modificaciones, pero si requirió de varios ajustes debido a la necesidad de que la placa se acople correctamente con el encapsulado, y que además permita la accesibilidad de los componentes que sean necesarios (*encoder*, interruptores y conectores). Para realizar los diseños 3D, se utilizó el programa de diseño de licencia libre *FreeCad* [19].

6.1. Encapsulado Móvil: Caja para la Placa Principal

Como se mencionó en la Sección 5.2, el dispositivo está compuesto por dos placas; en esta sección se abordará el diseño del encapsulado de la placa Principal. Se comentarán sus características y se presentará el resultado final.

6.1.1. Diseño e Impresión

El encapsulado está conformado por una estructura principal en donde se coloca la placa, y se cierra mediante una tapa que se ajusta utilizando cuatro tornillos M2,5 $\times$ 12 (según el sistema de Rosca Métrica). Para fijar la placa se diseñaron pilares cilíndricos huecos autorroscantes, con la finalidad de ajustar la misma mediante tornillos M1,5 $\times$ 3.

Dado que la placa Principal tiene dos conectores (*micro USB* y *RJ45*) que deben ser accesibles para los usuarios, el encapsulado necesita tener dos agujeros que permitan su conexión. Los mismos se encuentran en las caras laterales del diseño. Análogamente se incluye un orificio para el *switch ON/OFF* de la placa en la cara superior del encapsulado. Además la placa tiene un *encoder* rotatorio que permite al usuario seleccionar los distintos modos del dispositivo. Para este componente es necesario diseñar un hueco en la tapa del encapsulado. Dado que la altura del *encoder* es comparable a la del encapsulado, para facilitar el acceso se realiza el agujero de forma cónica. Por último se incluyen cuatro "alas" con ranuras a efectos de sujetar el dispositivo mediante, por ejemplo, elásticos.

Para diseñar e imprimir este encapsulado se tuvieron en cuenta los requerimientos exigidos, que se especificaron previamente en la Subsección 1.3.1. A continuación se

detallan las características para este encapsulado:

- Impresión 3D.
- Material: PLA (ácido poliláctico) [39].
  - Inodoro.
  - Aprobado por la FDA [33].
  - Biodegradable [46].
  - Temperatura de fusión: $120 - 150\ ^{\circ}C$ [47].
- Tamaño (cm): $5{,}0 \times 7{,}8 \times 2{,}1$.
- Peso total: 27 *gramos*.
- Tapa ajustable mediante tornillos M$2{,}5 \times 12$.
- Pilares para sostener y atornillar la placa electrónica mediante tornillos M$1{,}5 \times 3$.
- Espacio para conectores *micro USB* y *RJ45*.
- Espacio para un *encoder* y un *switch*.
- Bordes curvos para reducir posibles accidentes.
- Adaptación para sujeción a varias parte del cuerpo.

En la figura 6.1 se puede ver el plano del encapsulado, conteniendo la estructura de la caja y su tapa, mientras que en la figura 6.2 se muestra una foto del encapsulado logrado.

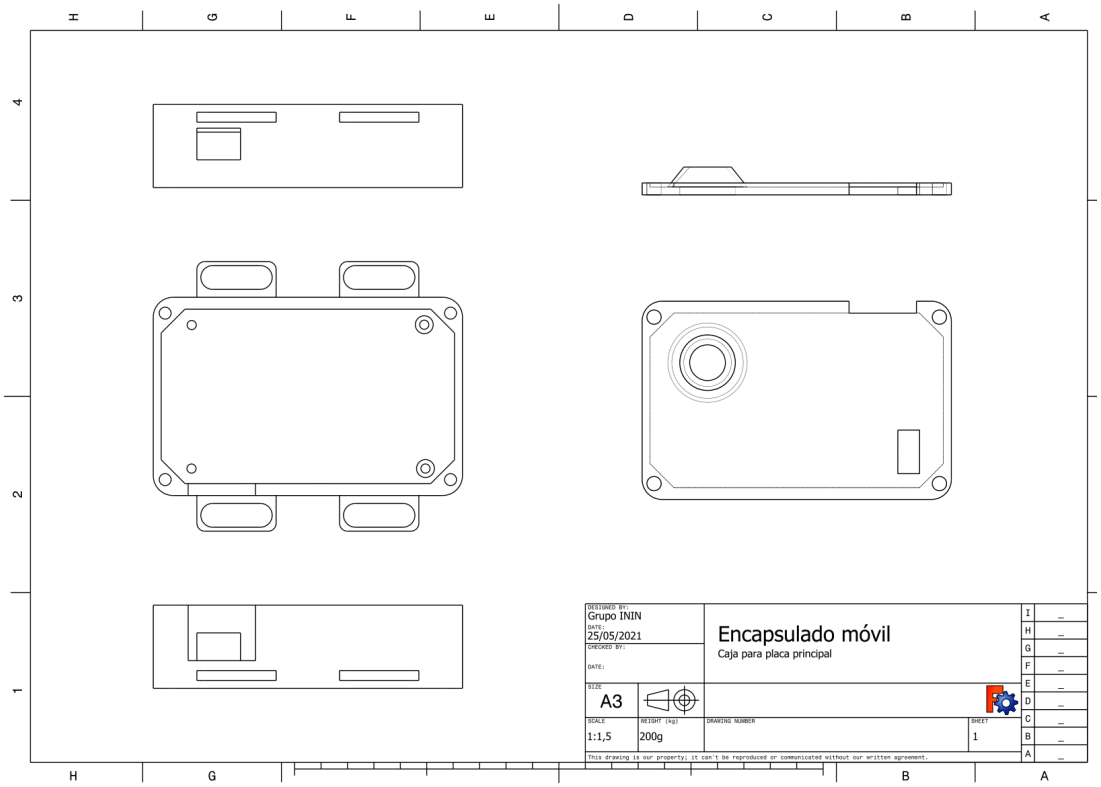

Figura 6.1: Plano para encapsulado de la placa Principal

Otro factor a diseñar es el método de sujeción; como fue mencionado, este encapsulado está pensado para ser vestible, e incluye "alas" laterales con este fin. Para lograr el objetivo de que pueda ser colocado en distintas partes del cuerpo, se utilizan elásticos con hebillas de bloqueo doble. Estas últimos permiten ajustar el largo del elástico, ajustándose así a la medida necesaria según la situación. En la figura 6.3 podemos ver una foto de lo antes descrito.

No obstante, el elástico que no es utilizado queda remanente y puede llegar a incomodar al momento de utilizar el dispositivo. Por otro lado se debe tener en cuenta que el largo del elástico necesario para la utilización del dispositivo en la cabeza es mucho mayor a la necesaria para la muñeca. Es por la suma de estos factores que se opta por diseñar un método de expansión (figura 6.4) que permite anexar un conjunto de elásticos y hebillas de doble bloqueo cuando se quiere utilizar en dispositivo en la cabeza.

Por último, en la figura 6.5 se puede apreciar el dispositivo completamente ensamblado, integrando el encapsulado y la placa Principal vista en la Subsección 5.2.1.

Figura 6.2: Encapsulado para la placa Principal

Figura 6.3: Método de sujeción del encapsulado Principal

Figura 6.4: Expansión del método de sujeción

Figura 6.5: Versión final de la placa Principal y su encapsulado

6.2. Encapsulado Botonera

Este encapsulado está diseñado para contener a la placa Botonera (la cual se vio en la Subsección 5.2.2). El mismo no tiene requerimientos previos, ya que la idea de realizar este segundo dispositivo surge luego de la etapa de planificación. Es por esto que se mantienen los criterios de diseño aplicados en la Subsección 6.1.1, con dos excepciones: en este caso el tamaño del dispositivo no es una limitante, ya que el mismo no es vestible; y considerando que el usuario no tendrá contacto directo con el encapsulado no se hizo foco en lograr bordes curvos.

6.2.1. Diseño e Impresión

Este encapsulado está diseñado de forma análoga al descrito en la Subsección 6.1.1, cuenta con una estructura principal en donde se coloca la placa Botonera, y una tapa que se ajusta mediante tornillos M2,5 $\times$ 12. También contiene pilares cilíndricos huecos en su interior de manera que se pueda ajustar la placa utilizando tornillos M1,5 $\times$ 3.

En una de sus caras laterales el encapsulado tiene seis agujeros que permiten conectar cables de audio a los conectores *Plug TRS* de 3,5 mm. A su vez en la tapa tiene seis orificios para dejar accesibles los interruptores que tiene la placa Botonera. Por último en la otra cara lateral se puede encontrar un abertura para el conector *RJ45*.

En la siguiente lista se muestran las características principales de este encapsulado.

- Impresión 3D.
- Material: PLA(ácido poliláctico).
- Tamaño (cm): 12,0 $\times$ 6,4 $\times$ 3,5.
- Peso total: 75 *gramos*.
- Tapa ajustable mediante tornillos M2,5 $\times$ 12.
- Pilares para sostener y atornillar la placa electrónica mediante tornillos M1,5 $\times$ 3.
- Espacio para conector *RJ45* y seis conectores de audio de 3,5 mm.
- Espacio para seis interruptores.

En la figura 6.6 se muestra el plano del diseño realizado y en la figura 6.7 una foto del encapsulado logrado. Por último en la figura 6.8 se presenta el encapsulado logrando conteniendo la placa Botonera, y los botones.

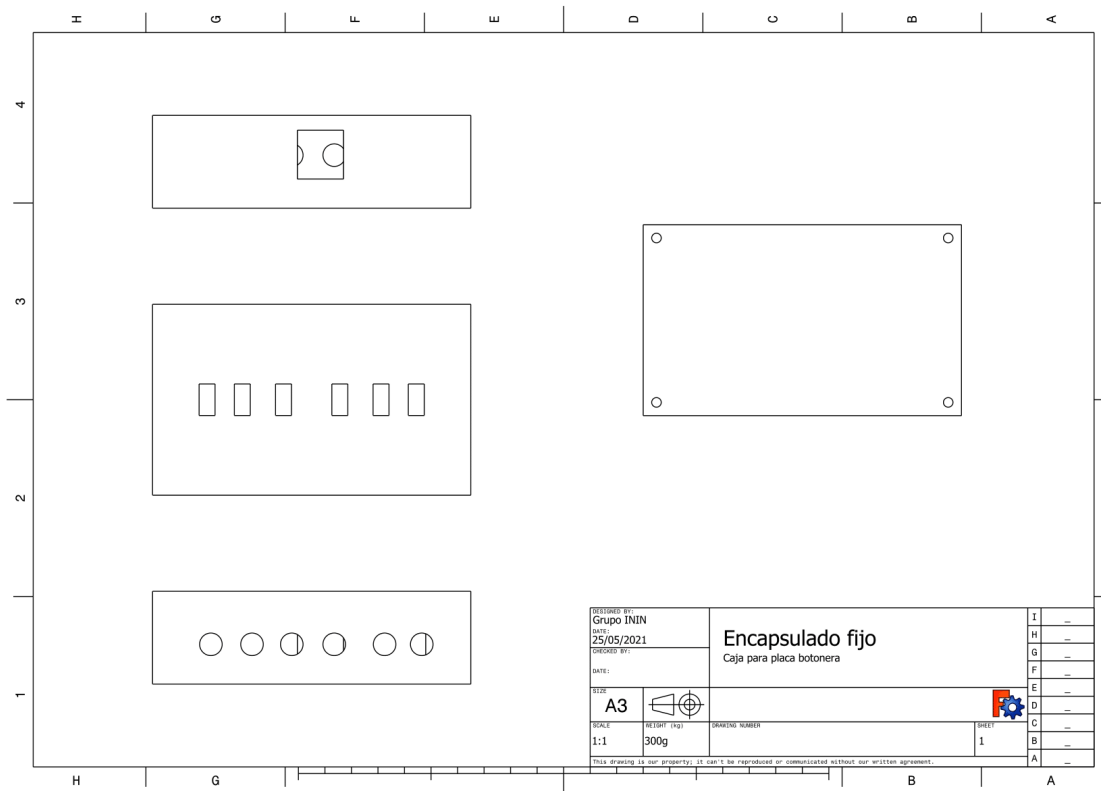

Figura 6.6: Plano de encapsulado para la placa Botonera

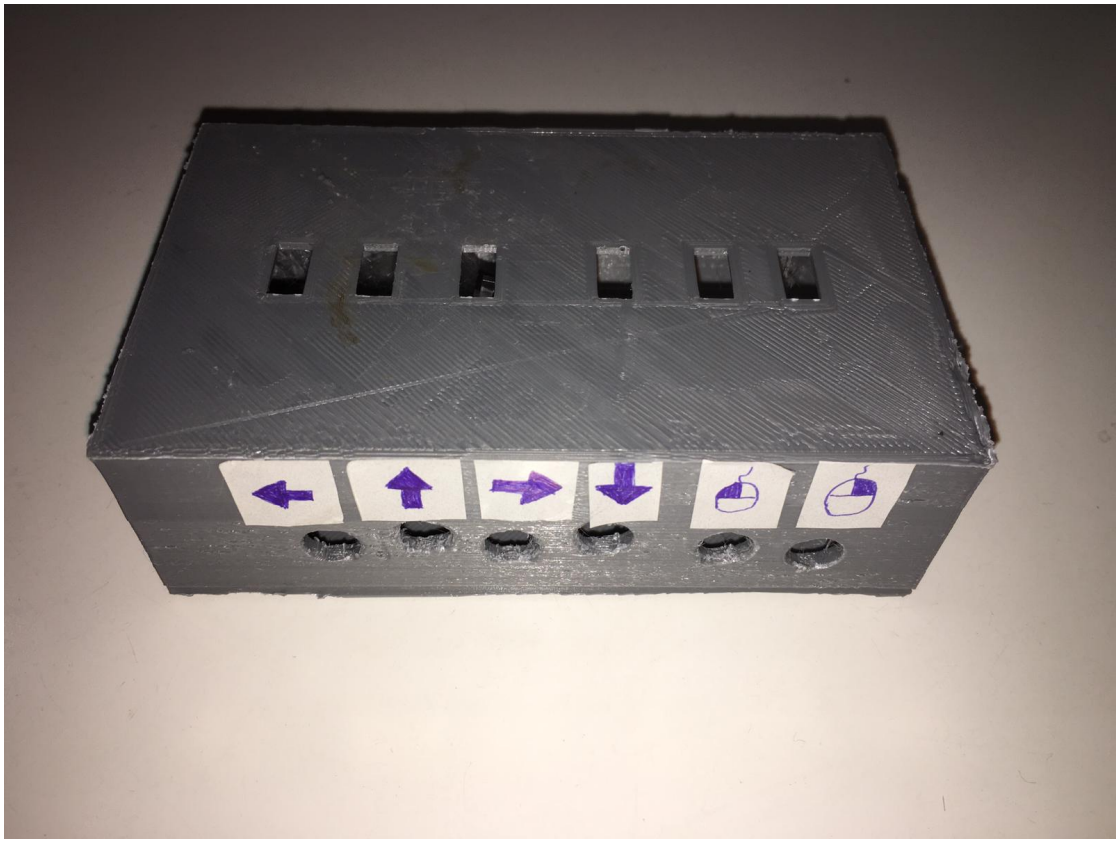

Figura 6.7: Foto del encapsulado para la placa Botonera

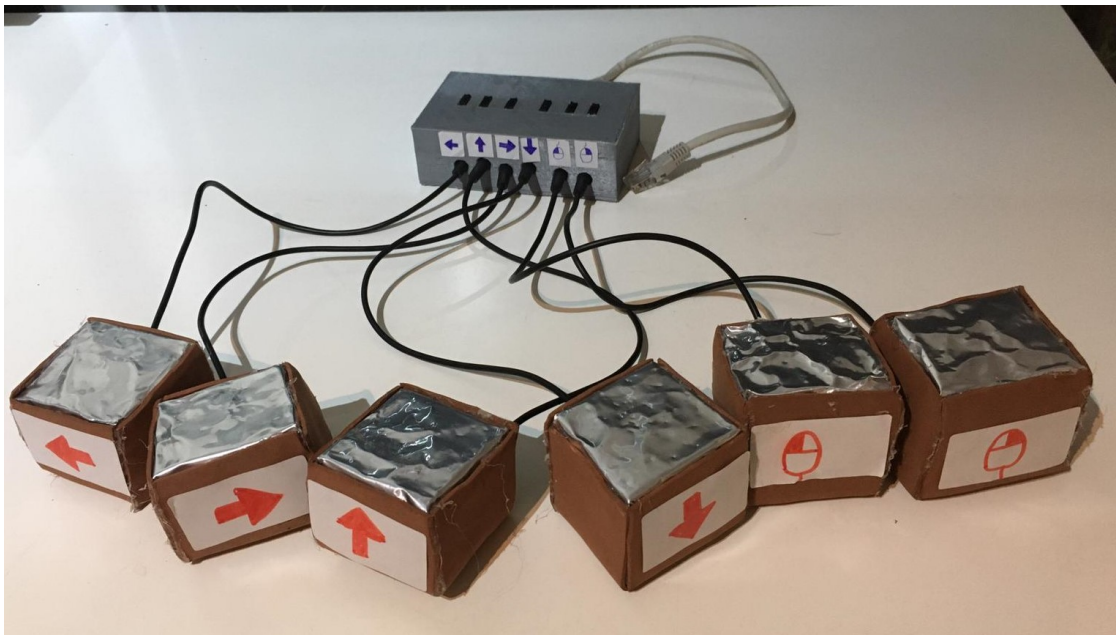

Figura 6.8: Versión final de la placa Botonera y su encapsulado, en conjunto con los botones

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 7

Consumo

En este capítulo se detallan los consumos del dispositivo en sus distintos modos, así como los cálculos para el dimensionamiento de la batería.

7.1. Cálculo del Consumo

En esta sección se realizará una primera estimación del consumo del sistema. Como fue explicado en la Sección 2.2, el dispositivo cuenta con cuatro modos. Los modos Gestos y Cabeceo utilizan las señales provenientes del conjunto de sensores *MARG*, mientras que los modos Barrido y Botonera utilizan las señales de los sensores capacitivos para funcionar. Es por esto que el análisis del consumo se realizará de a pares: por un lado los modos Gestos y Cabeceo, y por otro Barrido y Botonera.

Para comenzar, se detallará el consumo del microcontrolador presente en los cuatro modos, explicitando el consumo de los protocolos de comunicación usados por el mismo. También se presenta el consumo correspondiente a la comunicación *Bluetooth*.

7.1.1. Microprocesador

Si bien el *ARMCortex-M4* [29] puede configurarse para funcionar en modos de bajo consumo, en una primera aproximación se analiza el funcionamiento por defecto. En dicho modo el procesador está siempre operativo y el reloj opera a 64 MHz. En base a los resultados obtenidos se determinará si este modo de operación se ajusta a las requisitos de autonomía. A continuación se detallan los consumos nominales considerados para las condiciones mencionadas:

- Oscilador (64 MHz): 250 μA.
- I^2C: 50 μA@100 $kbps$.
- SPI: 50 μA@2 $Mbps$.
- CPU: Modo de consumo Normal 58 $\mu A/MHz \times 64\ MHz = 3{,}7\ mA$.

	Time	Source	Destination	Protocol	Length	Info
923	56.295711	ca:16:a1:0a:9e:08 (…	localhost ()	ATT	17	Rcvd Handle Value Notification, Handle: 0x0021 (Human Interface Device:
924	56.325713	ca:16:a1:0a:9e:08 (…	localhost ()	ATT	17	Rcvd Handle Value Notification, Handle: 0x0021 (Human Interface Device:
925	56.355751	ca:16:a1:0a:9e:08 (…	localhost ()	ATT	17	Rcvd Handle Value Notification, Handle: 0x0021 (Human Interface Device:
926	56.385748	ca:16:a1:0a:9e:08 (…	localhost ()	ATT	17	Rcvd Handle Value Notification, Handle: 0x0021 (Human Interface Device:
927	56.415754	ca:16:a1:0a:9e:08 (…	localhost ()	ATT	17	Rcvd Handle Value Notification, Handle: 0x0021 (Human Interface Device:
928	56.445713	ca:16:a1:0a:9e:08 (…	localhost ()	ATT	17	Rcvd Handle Value Notification, Handle: 0x0021 (Human Interface Device:
929	56.475704	ca:16:a1:0a:9e:08 (…	localhost ()	ATT	17	Rcvd Handle Value Notification, Handle: 0x0021 (Human Interface Device:
930	56.505740	ca:16:a1:0a:9e:08 (…	localhost ()	ATT	17	Rcvd Handle Value Notification, Handle: 0x0021 (Human Interface Device:
931	56.535740	ca:16:a1:0a:9e:08 (…	localhost ()	ATT	17	Rcvd Handle Value Notification, Handle: 0x0021 (Human Interface Device:
932	56.565704	ca:16:a1:0a:9e:08 (…	localhost ()	ATT	17	Rcvd Handle Value Notification, Handle: 0x0021 (Human Interface Device:
933	56.595702	ca:16:a1:0a:9e:08 (…	localhost ()	ATT	17	Rcvd Handle Value Notification, Handle: 0x0021 (Human Interface Device:
934	56.625705	ca:16:a1:0a:9e:08 (…	localhost ()	ATT	17	Rcvd Handle Value Notification, Handle: 0x0021 (Human Interface Device:
935	56.655744	ca:16:a1:0a:9e:08 (…	localhost ()	ATT	17	Rcvd Handle Value Notification, Handle: 0x0021 (Human Interface Device:
936	56.700742	ca:16:a1:0a:9e:08 (…	localhost ()	ATT	17	Rcvd Handle Value Notification, Handle: 0x0021 (Human Interface Device:
937	56.730713	ca:16:a1:0a:9e:08 (…	localhost ()	ATT	17	Rcvd Handle Value Notification, Handle: 0x0021 (Human Interface Device:
938	56.761132	ca:16:a1:0a:9e:08 (…	localhost ()	ATT	17	Rcvd Handle Value Notification, Handle: 0x0021 (Human Interface Device:
939	56.790743	ca:16:a1:0a:9e:08 (…	localhost ()	ATT	17	Rcvd Handle Value Notification, Handle: 0x0021 (Human Interface Device:
940	56.820725	ca:16:a1:0a:9e:08 (…	localhost ()	ATT	17	Rcvd Handle Value Notification, Handle: 0x0021 (Human Interface Device:
941	56.850758	ca:16:a1:0a:9e:08 (…	localhost ()	ATT	17	Rcvd Handle Value Notification, Handle: 0x0021 (Human Interface Device:
942	56.880741	ca:16:a1:0a:9e:08 (…	localhost ()	ATT	17	Rcvd Handle Value Notification, Handle: 0x0021 (Human Interface Device:
943	56.910728	ca:16:a1:0a:9e:08 (…	localhost ()	ATT	17	Rcvd Handle Value Notification, Handle: 0x0021 (Human Interface Device:
944	56.940747	ca:16:a1:0a:9e:08 (…	localhost ()	ATT	17	Rcvd Handle Value Notification, Handle: 0x0021 (Human Interface Device:

Figura 7.1: Captura de la transmisión de paquetes *Bluetooth* del dispositivo en modo Gestos, utilizando el programa *Wireshark*

7.1.2. Comunicación *Bluetooth*

El dispositivo cuenta con una interfaz *Bluetooth* que permite establecer la comunicación con un PC o con un dispositivo móvil; esta interfaz se configura para transmitir con una potencia de 4 dBm lo cual garantiza una cobertura de 16 metros [10], que se considera más que suficiente para todos los casos de uso. A continuación se detallan los consumos asociados a la comunicación *Bluetooth* [41]:

- TX: 7,5 mA@4 dBm.
- RX: 5,4 mA@1 $Msps$.

Para calcular el consumo de la antena es necesario conocer cada qué período de tiempo la misma se activa, así como también la duración de cada transmisión. Para esto se toma el escenario considerado como más restrictivo, el cual se da cuando el dispositivo está enviando constantemente, esto es, moviendo el cursor del *mouse* en cada iteración del programa.

Bajo estas condiciones, para estimar el consumo es necesario saber cuántos *bytes* se envían, y cada cuánto. Para esto se capturan los paquetes *Bluetooth* cuando está establecida la comunicación con la PC y se está moviendo el cursor. Esta captura se realiza utilizando el programa $Wireshark$ [49]. Para hacer esto es necesario capturar con la interfaz correspondiente, esto varía según se quiera capturar en *Windows* [1] o *Linux* [48].

En la figura 7.1 se muestra la captura mencionada. Se puede observar que el origen (*source*) de todos los paquetes coincide, este es el dispositivo enviando desplazamientos del *mouse*; dado que de la captura no se desprenden datos de transmisiones desde la PC, no se considera el consumo en recepción. Vale aclarar que, en caso de existir trasmisiones desde la PC al microcontrolador, serían propias del protocolo ya que la solución implementada utiliza una comunicación unidireccional.

También se nota que el período resulta ser aproximadamente $\tau_{Tx} = 30\ ms$ (este tiempo está determinado por el sistema operativo [25]), en donde se envían 17 *bytes*. Con estos datos, se puede obtener el consumo medio de la transmisión gracias a una herramienta disponible en la página web del fabricante [38]. En las figuras 7.2 y 7.3 se

presenta el resultado de la simulación, así como la configuración. Los valores correspondientes a *Master sleep clock accuracy* y *Slave sleep clock accuracy* se desconocen, por lo que se toman los valores que determinan la corriente más alta. Con esto, se obtiene una corriente media de transmisión de 186 μA.

Test setup	
Chip	nRF52832 QFAAE0
Softdevice	s132 6.1.0
Voltage	3.3 V
Regulator	DCDC
BLE event details	
Interval	30.00 ms
Length	2.52 ms
Number of packets	1
Master sleep clock accuracy	500 ppm
Slave sleep clock accuracy	500 ppm
Data transmission	
LL PDU size	27 Byte
TX payload per event	17 Byte
TX LL throughput	4.53 kbps
RX payload per event	0 Byte
RX LL throughput	0 bit/s
On air data rate	1 Mbps

Current consumption	
BLE event total charge	5.51 µC
Idle current	2.0 µA
Total average current	186 µA

Figura 7.2: Configuración de la antena para transmitir, y corriente media estimada

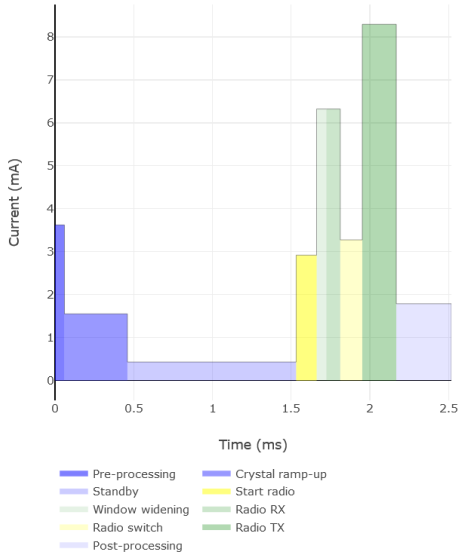

Figura 7.3: Gráfica corriente en función del tiempo para una transmisión

7.1.3. Regulador de Voltaje

El dispositivo funciona con una batería de Litio de voltaje 3,7 V, y dado que los sensores y el microcontrolador se alimentan de un voltaje de 3,3 V fue necesario incluir en la placa Principal un regulador de voltaje lineal a 3,3 V (ver Subsección 5.2.1). En términos del consumo este integrado impacta de forma directa a través de su eficiencia, la cual depende de la relación entre el voltaje de entrada y el voltaje de salida del integrado, y se puede estimar como [37]:

$$\nu_{REG} \approx \frac{V_{salida}}{V_{entrada}}. \tag{7.1}$$

Utilizando la ecuación 7.1 podemos obtener la eficiencia del regulador presente en la placa principal, donde el voltaje de entrada es $V_{bat} = 3{,}7\ V$, y el voltaje de salida es $V_{nRF} = 3{,}3\ V$. De esta forma la eficiencia del regulador de la placa Principal resulta $\nu_{REG} \approx \frac{V_{nRF}}{V_{bat}} = 0{,}89$.

Teniendo esta eficiencia se puede calcular la corriente total consumida a la batería, a partir de la corriente que entrega el regulador lineal de la siguiente forma: $I_{bat} = \frac{I_{REG}}{\nu_{REG}}$. Análogamente se puede obtener la corriente que consumen los integrados conociendo la

corriente suministrada por la batería.

7.1.4. Comunicación I^2C

El conjunto de sensores *MARG* se comunica con el microcontrolador a través del protocolo de comunicación I^2C. Para su correcto funcionamiento este protocolo exige que los pines que utiliza estén en nivel alto mediante resistencias de *pull-up*, de forma que se puedan generar las señales llevando el voltaje a cero. Para el caso particular implementado, estas resistencias involucradas son de 10 $k\Omega$ como se explicó en la Subsección 5.2.1. Con respecto al consumo la corriente que circula al trasmitir un cero se calcula de la forma:

$$i_{I^2C} = \frac{V_{nRF}}{R} = 330\ \mu A \tag{7.2}$$

donde $V_{nRF} = 3{,}3\ V$ y $R = 10\ k\Omega$.

Por otro lado podemos determinar cuantos *bytes* se transmiten por lectura al *MARG*. Cada lectura a un sensor está determinada por el valor correspondiente a cada eje en punto flotante (4 *bytes*), a lo cual se le suman dos *bytes* de direccionamiento a los *chips*. Cada transferencia está dada por:

- Un *bit* de comienzo.
- Un *byte* de direccionamiento más un *bit* de reconocimiento.
- *Bytes* de información, cada uno con su *bit* de reconocimiento.
- Un *bit* de parada.

De esta forma, la cantidad total de *bits* por lectura al *MARG* es

$$(3\ sensores \times 3\ ejes \times 4\ bytes \times 9\ bits) + 2 \times (1\ byte \times 9\ bits + 2\ bits) = 346\ bits$$

.

Dado que la velocidad a la que se configura la comunicación I^2C es de 100 *kbps*, y se envían 346 *bits* por cada ciclo, se tiene que la comunicación está activa durante 3,48 *ms*. Teniendo este tiempo de activación se puede calcular el consumo medio de cada pin de comunicación I^2C, de la siguiente forma:

$$I_{I^2C} = \frac{1}{T} \times \int_{t=0}^{t=t_{trans}} i_{I^2C} \cdot dt \times D.$$

Donde T es el periodo entre transmisiones, t_{trans} es el tiempo que dura la transferencia y D es el porcentaje de tiempo que la señal está baja. Para nuestro caso particular se toma que $t_{trans} = 3{,}48\ ms$ y $T = \frac{1}{frecuencia\ de\ muestreo} = \frac{1}{104\ Hz} = 9{,}6\ ms$ ya que se considera que se leen los datos a la frecuencia de muestreo.

Por otro lado, el valor de D se diferenciará según el caso: para el caso de la señal de reloj se tomará un valor de 50 %, mientras que para la señal de datos se considerará el peor caso tomando $D = 100\,\%$.

$$I_{I^2C_{total}} = \frac{1}{9{,}6\ ms} \times \int_{t=0}^{t=3{,}48\ ms} 330\ \mu A \cdot dt \times 0{,}5 + \frac{1}{9{,}6\ ms} \times \int_{t=0}^{t=3{,}48\ ms} 330\ \mu A \cdot dt \times 1 = 179\ \mu A.$$

Integrado	Consumo nominal estimado
LSM6DSOX (IMU) [44]	0,55 mA promedio en modo alto rendimiento
LIS3MDL (Magnetómetro) [43]	0,27 mA promedio en en modo alto rendimiento
nRF52832 CPU (Microcontrolador) [30]	3,70 mA ON mode @64 MHz
nRF52832 *SPI* (Microcontrolador) [30]	50 μA@2 $Mbps$
nRF52832 I^2C (Microcontrolador) [30]	50 μA@100 $kbps$
nRF52832 Antena (Bluetooth) [35]	186 μA@4 dB
M95320 (EEPROM) [45]	5 mA de pico en modo de escritura o lectura y 3 μA en modo *Stand-By*
Canal I2C	179 μA
AP2112 (regulador de voltaje) [7]	661 μA
Consumo total	5,60 mA

Tabla 7.1: Consumos de integrados y sensores utilizados en el modo Gestos o Cabeceo

7.1.5. Modo Cabeceo y Gestos

El consumo teórico de estos dos modos se calculará en conjunto, ya que ambos utilizan los mismos sensores y el mismo microcontrolador con las mismas configuraciones. En la tabla 7.1 se muestran los componentes, considerando su funcionamiento nominal.

Para obtener el consumo del regulador lineal que se aprecia en la tabla, alcanza con sumar las corrientes consumidas por los otros componentes y dividir este valor por la eficiencia calculada en la Subsección 7.1.3, se suma además la corriente *quiescent* del regulador (55 μA). Por otra parte, el consumo de la memoria será despreciado, ya que solo se lee cuando se enciende el dispositivo; de igual manera se desprecia el consumo del protocolo SPI que solo es utilizado para comunicar el microcontrolador con esta memoria. Entonces sumando el consumo de todos los integrados se puede obtener que el consumo teórico nominal en el modo Gestos o Cabeceo es: 5,60 mA.

7.1.6. Modo Barrido y Botonera

Dado que estos modos utilizan la misma configuración de *hardware* se espera que tengan un consumo similar, por lo que se los analiza en conjunto. Para poder hacer esto, es necesaria una salvedad previa: como fue mencionado en la Sección 4.2, el modo Barrido no transmite en todas las iteraciones del bucle, por lo que el consumo de las transmisiones *Bluetooth* en este modo es menor al de los otros modos; sin embargo, dado se quiere analizar el caso más restrictivo y a efectos de simplificar el análisis, se considerará $I_{TxBarrido} = I_{TxBotonera} = 186\ \mu A$.

En la tabla 7.2 se muestran los consumos promedios de los integrados involucrados

Integrado	Consumo nominal
AT42QT1011 (Sensor capacitivo) [24] ×6	34 μA Low mode
nRF52832 CPU (Microcontrolador) [30]	3,70 mA ON mode @64 Mhz
nRF52832 Interfaz *Bluetooth* 7.1.2	186 μA@4 $dB, 1$ $Mbps$
AP2112 (regulador de voltaje) [7]	510 μA
Consumo total	4,60 mA

Tabla 7.2: Consumos de integrados utilizados en el modo Barrido o en el modo Botonera

en el modo Barrido o en el modo Botonera, considerando su funcionamiento nominal. De forma análoga al desarrollo para Cabeceo y Gestos se desprecia el consumo de la memoria. En este caso, al considerar los seis sensores capacitivos se obtiene una corriente de 4,60 mA.

7.2. Medición del Consumo

Previo a realizar las mediciones de consumo se realiza un breve análisis de la forma de onda de la corriente consumida por el dispositivo al realizar trasmisiones *Bluetooth*. Para esto se alimenta el dispositivo a la entrada de la batería con un voltaje de 5 V, y se conecta una resistencia de 15 Ω en serie con esta entrada. Con la ayuda de un osciloscopio se mide el voltaje en bornes de esta resistencia, lo que permite de manera indirecta observar los picos de corriente que consume el dispositivo. En la figura 7.4 se muestra la captura realizada con el osciloscopio en los bornes de la resistencia, que corresponde al canal 2 ($CH2$) del mismo. Por otro lado en el canal 1 ($CH1$) se muestra el voltaje en un pin digital del microcontrolador, el cual se utiliza como una señal de control; previo a ejecutar el código correspondiente a las transmisiones *Bluetooth* el voltaje se pone en alto, y se baja cuando el mismo termina.

Lo más relevante a destacar en la captura es que durante el tiempo en que el microcontrolador ejecuta el código correspondiente a las transmisiones *Bluetooth*, ($CH1$ en nivel alto) la amplitud de los picos de voltaje en la resistencia disminuyen, a excepción del momento en que efectivamente se transmite. Esto lleva a concluir que el consumo medio cuando el microcontrolador está en este estado es menor a cuando deja de ejecutar este código. Esto contradice la hipótesis utilizada en el cálculo de la corriente (que el consumo máximo se da al transmitir en cada iteración del bucle). Esto puede deberse a que la implementación de las transmisiones por parte de la antena disminuyan el consumo durante los 30 ms que dura la misma.

Este resultado debe ser teniendo en cuenta para analizar las mediciones del consumo de los distintos modos, ya que implica que el mismo disminuirá cuando se estén transmitiendo órdenes del *mouse*. Esta diferencia de consumo se analizará solo para el modo Barrido, ya que alcanza para observar la diferencia mencionada.

Para medir el consumo se alimenta el dispositivo con una fuente de voltaje a 3,7 V (emulando el uso típico) en la entrada de la batería, y se conecta un amperímetro de precisión en serie con la fuente. La disposición utilizada para realizar esta medición se presenta en la figura 7.5.

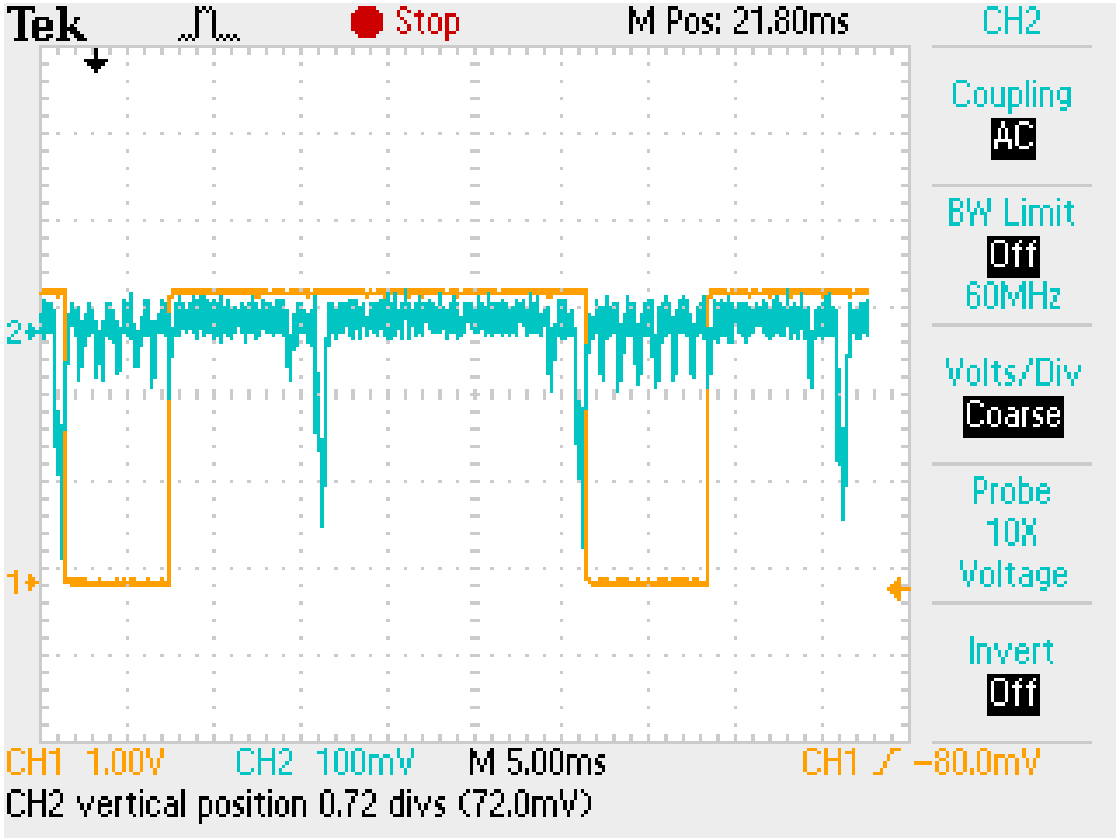

Figura 7.4: Captura del voltaje en bornes de la resistencia

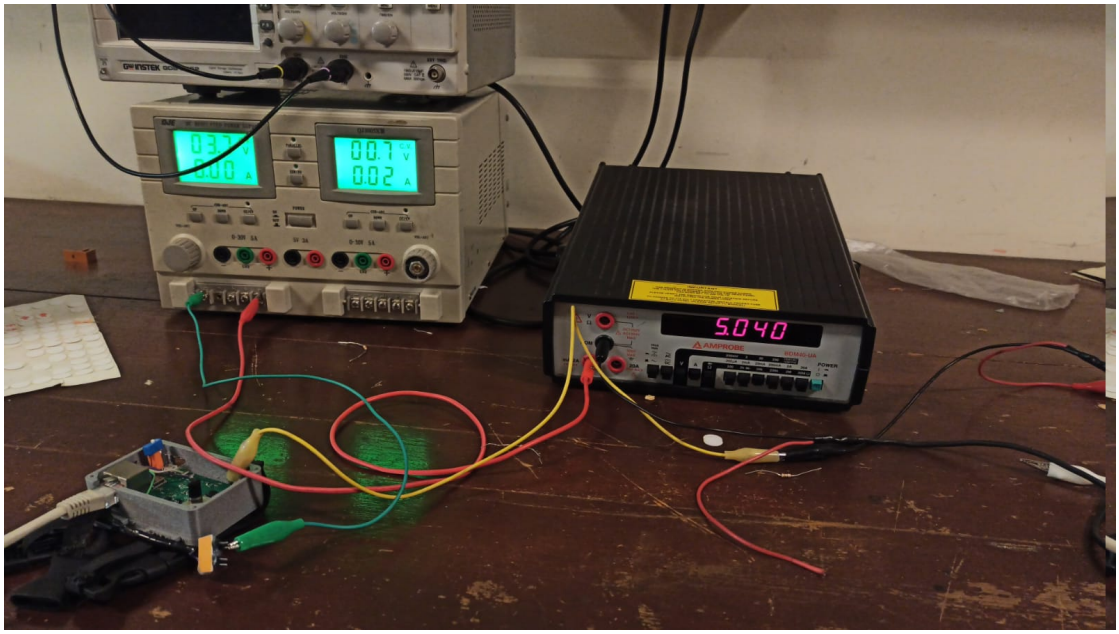

Figura 7.5: Configuración para medición de corriente, utilizando un amperímetro en serie con el dispositivo

Las mediciones obtenidas se presentan en las figuras 7.6 y 7.7, en donde la primera muestra el consumo del dispositivo en modo Barrido cuando se realizan transmisiones *Bluetooth* (se está moviendo el cursor), y en la segunda se muestra el consumo cuando estas transmisiones cesan (cursor frenado).

Como se puede observar en las figuras el modo Barrido consume significativamente menos cuando se están realizando transmisiones. Esto reafirma la conclusión de que cuando el dispositivo no está ejecutando el código correspondiente a transmitir, el consumo aumenta. Por esta razón para el resto de los modos se presentarán las mediciones del consumo cuando el dispositivo no se encuentre transmitiendo, ya que es aquí cuando se observa la condición más restrictiva en términos de autonomía.

Figura 7.6: Corriente medida por un amperímetro en serie con el dispositivo en modo Barrido cuando se realizan transmisiones *Bluetooth*

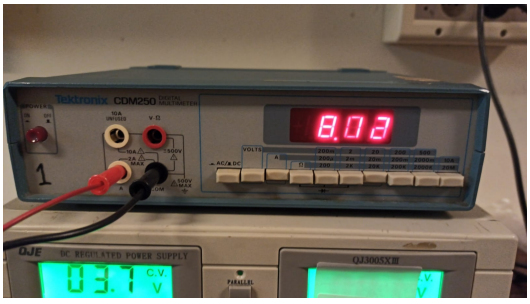

Figura 7.7: Corriente medida por un amperímetro en serie con el dispositivo en modo Barrido cuando no se están realizando transmisiones *Bluetooth*

7.2.1. Modo Gestos y Cabeceo

Utilizando la configuración mostrada en la figura 7.5, se realizan las mediciones para los modos Gestos y Cabeceo, que son presentadas en las figuras 7.8 y 7.9 respectivamente. Como ya fue comentado anteriormente, las mediciones de consumo para estos modos corresponden a los casos en los que el dispositivo no se encuentra transmitiendo.

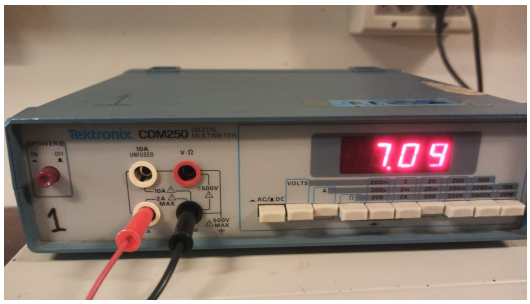

Figura 7.8: Corriente medida por un amperímetro en serie con el dispositivo en modo Gestos

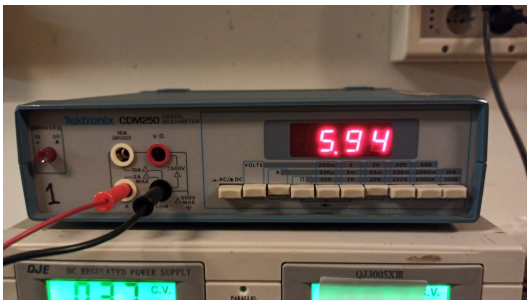

Figura 7.9: Corriente medida por un amperímetro en serie con el dispositivo en modo Cabeceo

Se puede observar que la corriente que consume el dispositivo en el modo Gestos es $I_{GestosMED} = 7{,}09\ mA$, mientras que el consumo en el modo Cabeceo es $I_{CabeceoMED} = 5{,}94\ mA$. Como era de esperarse estos valores son muy similares entre si, dado que utilizan los mismos integrados.

7.2.2. Modo Barrido y Botonera

Análogamente al caso anterior, para la medicion del consumo de los modos Barrido y Botonera se utilizó la configuración mostrada en la figura 7.5. La medida obtenida cuando el dispositivo es configurado en modo Barrido ya fue introducida en esta sección y el valor de corriente es de $I_{BarridoMED} = 8{,}12\ mA$. Por otra parte, en la figura 7.10 se presenta el resultado obtenido con el modo Botonera, donde se tiene una corriente de $I_{BotoneraMED} = 6{,}88\ mA$. Ambas medidas corresponden a casos en los que el dispositivo no está transmitiendo.

Si comparamos estos dos valores con el calculado teóricamente, se puede notar una diferencia cercana al doble para el caso del Barrido y de un 35 % aproximadamente para el caso del modo Botonera. Esta diferencia si bien es notable, no es determinante para calcular la autonomía del dispositivo, ya que para esto es razonable utilizar el consumo medido.

Figura 7.10: Corriente medida por un amperímetro en serie con el dispositivo en modo Botonera

7.3. Batería

En esta sección se detallan las características de la batería seleccionada, así como la autonomía obtenida con la misma.

7.3.1. Características de la Batería

Dado que la capacidad máxima de la batería está determinada por sus dimensiones, simplemente se selecciona una del voltaje apropiado teniendo en cuenta el tamaño disponible en el encapsulado.

- Voltaje: 3,7 V
- Capacidad: 300 mAh
- Batería de polímero de litio recargable
- Dimensiones (cm): $3{,}4 \times 2{,}0 \times 0{,}7$

7.3.2. Autonomía del Dispositivo

Para calcular la autonomía del dispositivo se utilizará el consumo en modo Barrido cuando el mismo no está realizando transmisiones *Bluetooth*, ya que es el que más restringe la autonomía. El consumo medido para este modo es 8,12 mA, y el dispositivo resulta tener una autonomía en el peor caso de:

$$\frac{300\ mAh}{8{,}12\ mA} = 36{,}9\ hs. \tag{7.3}$$

Considerando el peor caso para el consumo del dispositivo se supera el requerimiento planteado en la Subsección 1.3.2; en caso de que se deseara disminuir la corriente consumida, pueden configurarse los modos de bajo consumo para los sensores y el microcontrolador presentes en la placa Principal.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 8

Evaluación Final del Dispositivo

Este capítulo se enfocará en mostrar cómo se llevó a cabo el proceso de validación del dispositivo. En el mismo se evalúa si se cumplieron los objetivos establecidos para el proyecto teniendo en cuenta el alcance definido.

8.1. Evaluación General del Proyecto

Habiendo hablado en detalle del "qué" y "cómo" del proyecto, es momento de realizar la evaluación de lo obtenido. El objetivo principal de este proyecto fue desarrollar un dispositivo de interfaz humana capaz de sustituir un *mouse* o panel táctil convencional, tal como fue definido en la Sección 1.2. En pos de alcanzar este objetivo fue necesario realizar un desglose en distintos objetivos parciales, los cuales fueron alcanzados en gran medida.

Por un lado se realizó un desarrollo integral tanto a nivel de *firmware* como de *hardware*, y se obtuvo un prototipo final que cumple con los requerimientos impuestos. En primera instancia, su tamaño se considera adecuado teniendo en cuenta las aplicaciones para las que fue concebido. Para apoyar esta afirmación, en la figura 8.1 podemos ver el caso particular en el que el dispositivo es colocado en la muñeca del usuario. Su tamaño es considerablemente más grande que otros dispositivos vestibles en la muñeca (por ejemplo, relojes), por el hecho de que la placa del dispositivo posee un conector $RJ45$ y un *encoder* que obligan a aumentar el volumen del encapsulado. Sin embargo el tamaño del dispositivo permite un correcto movimiento del mismo sin generar demasiada incomodidad. Cuando el dispositivo es colocado en la cabeza (figura 8.2) el tamaño también resulta adecuado, permitiendo utilizar el dispositivo de manera cómoda.

El método de sujeción cumple con el objetivo de ser ajustable a varias partes del cuerpo de manera simple. Sin embargo, posee algunos problemas; por un lado, el uso de broches grandes brinda cierta robustez (debido a que son menos propensos a romperse y desabrocharse) pero resulta un tanto incomodo en su uso. Esto se debe a que, como se ve en la figura 8.3, el tamaño de los broches es comparable con el ancho de la muñeca (más si se tiene en cuenta que puede ser utilizado por niños), pudiendo generar alguna molestia tras un uso prolongado. Asociado también a esto, en algunos casos el agarre no resulta ser del todo firme y se pierde tensión en los elásticos, lo que genera que el

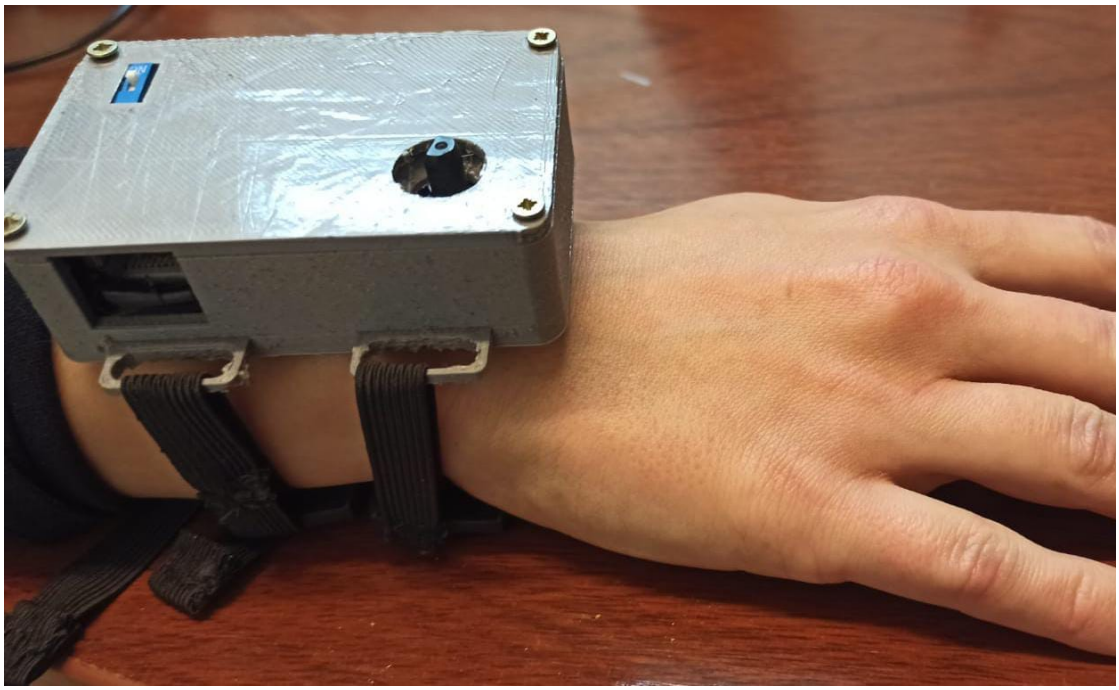

Figura 8.1: Dispositivo final colocado en la muñeca

dispositivo quede "flojo" pudiendo dificultar su uso.

En cuanto a lo que refiere a la usabilidad, los varios modos implementados logran que el dispositivo se adapte a distintas necesidades de los usuarios, cumpliendo con una de las principales metas del proyecto. Esta variedad de modos permite que sea usado tanto mediante movimientos del usuario (como el caso de los modos Cabeceo y Gestos), como por el accionado de botones que permiten mover el cursor (como en los casos Barrido y Botonera).

Con respecto a los botones se logró que el diseño sea muy versátil, ya que pueden tener forma y tamaño adaptable. Además no es necesario realizar presión sobre los mismos, ya que se activan al ubicar la mano a una distancia determinada. Esta distancia mínima se fija mediante *jumpers* tal como se explicó en la Subsección 5.2.2. La configuración de esto trae una desventaja intrínseca; para realizarla se torna necesario agregar o sacar *jumpers* en la placa Botonera, lo que puede resultar poco práctico debido a la necesidad de abrir el encapsulado y manipular componentes pequeños (como es el caso de los *jumpers*).

Si bien la tecnología de los botones evita la necesidad de presionar los mismos para activarlos, tiene la limitante de que deben ser accionados con la piel. Esto implica que si bien a priori pueden ser activados con cualquier parte del cuerpo, es necesario que la misma esté descubierta.

La mayoría de estas evaluaciones se sustentan en apreciaciones subjetivas realizadas por los integrantes de este proyecto. Durante el proceso se realizaron algunas pruebas y validaciones, tanto con el cliente (Escuela Nº 200) como con personas con mayor conocimiento sobre el público objetivo. Sin embargo no fue posible mostrar la versión final a potenciales usuarios ni a las personas capacitadas para realizar una validación

Figura 8.2: Dispositivo final colocado en la cabeza

Figura 8.3: Método de sujeción del dispositivo final colocado en la muñeca

más profunda del dispositivo y su funcionalidad. Esto impide realizar una evaluación más categórica del dispositivo.

Su configuración e instalación inicial resultan simples, logrando así que no sean necesarios demasiados conocimientos previos para su uso. El protocolo utilizado para la conexión (*Bluetooth*) cumple con el cometido de que el dispositivo sea inalámbrico, así como también permite su uso en una gran variedad de dispositivos, debido a que actualmente está integrado en la gran mayoría de las computadoras personales y dispositivos móviles. Esta facilidad de conexionado implica que una vez configurado el dispositivo, al ser energizado se conectará automáticamente al sistema operativo con el que esté emparejado. Esto puede representar una dificultad si se desea utilizar en otra PC o dispositivo móvil, pero se puede solucionar fácilmente apagando el dispositivo y desemparejándolo desde el sistema operativo. Por otro lado, a lo largo de los meses de uso se observaron problemas en la conexión *Bluetooth*; en algunos casos, se presentan dificultades al intentar conectarse con un dispositivo (el mismo se conecta y desconecta repetidamente). Esta situación puede variar según el *host*, y no necesariamente se replica al cambiar de uno a otro.

Al comienzo de este documento (Capítulo 1) se realizaron algunos comentarios referidos al estado del arte y, en particular, a las distintas características de los dispositivos disponibles en el mercado (englobados en la tabla 1.1). Habiendo ya desglosado las ca-

racterísticas y diseño del dispositivo *ININ*, resulta pertinente realizar una comparativa con esos dispositivos; en la tabla 8.1 se puede ver al dispositivo resultante cotejado con algunas soluciones comerciales.

Nombre	Precio aproximado	PlugPlay	Click izquierdo	Click derecho	Movimiento del cursor	Click y movimiento simultáneo	Usable solo con botones	Usable solo con movimientos
Quha Zono [34]	USD 1000	No	Si	Si	Si	No	No	Si
5 Switch [12]	USD 129	Si	Si	Si	Si	Si	Si	No
enPathia [18]	227 €	Si	No	Si	Si	Si	No	Si
ININ	USD169	Si[1]	Si	Si[2]	Si	Si[2]	Si	Si

Tabla 8.1: ININ comparado con soluciones disponibles en el mercado

[1]De manera inalámbrica.
[2]Únicamente en modos Barrido y Botonera.

El precio asociado al dispositivo ININ está dado por el costo de la fabricación de las placas y la compra de los componentes. A continuación se realiza un análisis en detalle:

- USD 140 de placas (Precio por unidad en base a dos placas fabricadas).
  - USD 1 por *PCB* ININ.
  - USD 75.5 de ensamblado y algunos componentes ININ.
    - USD 60.5 de componentes.
    - USD 15 de ensamblado.
  - USD 1 por *PCB* Botonera.
  - USD 62.5 de ensamblado y algunos componentes Botonera.
    - USD 18.5 de componentes.
    - USD 44 de ensamblado.
- USD 28.99 asociado a otros componentes.

8.2. Evaluaciones Específicas y Usabilidad

Esta sección está dedicada a la usabilidad del dispositivo logrado, así como a las evaluaciones especificas de cada uno de los modos implementados. Recordando la Sección 2.2 estos son: Barrido, Botonera, Cabeceo y Gestos. A efectos de poder evaluar de forma razonable los modos de uso, se realizaron una serie de pruebas que se entienden representativas de lo que se pretende de un periférico *HID*. Las pruebas realizadas fueron las siguientes:

- Escribir utilizando un teclado en pantalla.
- Explorador de archivos: navegar, cambiar ubicación de archivos, etc.
- Juego “Buscaminas”.

Si bien este juego no es en si mismo representativo del uso de una PC o dispositivo móvil, la prueba se considera relevante ya que introduce un factor de no predictibilidad sobre los movimientos a realizar; esto permite eliminar el sesgo intrínseco de las otras dos pruebas, en las que se corre el riesgo de evaluar el modo únicamente para una secuencia dada de movimientos.

8.2.1. Modo Barrido

Como fue mencionado, el objetivo principal de este modo es permitir al usuario controlar el cursor del *mouse* con la mínima interacción posible. En base a lo desarrollado en la Subsección 2.2.1, y a las pruebas mencionadas, este objetivo se alcanzó de forma satisfactoria siendo sencillo tanto escribir en pantalla como jugar al "Buscaminas". En lo referente a la navegación de archivos, se resalta la capacidad de realizar acciones compuestas, como puede ser la selección múltiple. Todas estas tareas llevan necesariamente más tiempo que con un *mouse* convencional debido al comportamiento semi autónomo de este modo. En las pruebas fue necesario un breve período de adaptación para lograr desarrollar las tareas con fluidez. Pese a esto se concluye que este modo se desempeña acorde a lo esperado.

8.2.2. Modo Botonera

El funcionamiento de este modo fue presentado en la Subsección 2.2.2. El mismo funciona mediante seis botones, correspondientes a las cuatro direcciones y los dos *clicks*, que al ser activados disparan la acción correspondiente del *mouse*.

En términos de usabilidad el modo Botonera resulta muy útil e intuitivo, obteniendo un muy buen resultado en las pruebas de usabilidad comentadas al inicio de esta sección. Este modo se destaca al usarlo en el Explorador de archivos, ya que permite realizar acciones complejas utilizando más de un botón, por ejemplo: *click*&*drag*, movimientos en diagonal, etc.

8.2.3. Modo Cabeceo

Según fue explicado en la Subsección 2.2.3, este modo traduce posiciones angulares respecto a la vertical (*pitch* y *roll*) en movimientos en los ejes de la pantalla.

En las pruebas realizadas, se logró controlar el cursor de forma satisfactoria, pudiendo utilizarse el teclado en pantalla sin inconvenientes. En cuanto a la exploración de archivos, la misma resulta natural; tanto la acción de *click* como la de doble *click* son muy sencillas de realizar. Sin embargo, a raíz de no contar con la acción de *click* secundario o la opción de mantener presionado el *click*, no es posible arrastrar archivos.

Un problema de este modo puede darse cuando se desea detener el cursor, ya que para esto es necesario que el ángulo correspondiente esté por debajo de cierto umbral. Esta condición puede no darse inmediatamente cuando el usuario quiere frenar, debido a que los movimientos con la cabeza son lentos; esto genera una demora en el frenado del cursor, lo que dificulta detener el movimiento en el lugar exacto (por ejemplo una celda en "Buscaminas"). Luego de un tiempo de usar el dispositivo en este modo, dicho

efecto fue paulatinamente menos apreciable. Un factor menos cuantificable, pero que puede generar incomodidad en el uso de este modo, es la necesidad de mover la cabeza manteniendo la mirada fija en la pantalla, principalmente al subir o bajar el mentón.

El hecho de que el dispositivo defina el origen una única vez al principio de la ejecución, determina que si su posición relativa a la cabeza del usuario cambia, también lo hace la referencia de quietud. Recordar que la posición es comparada con este valor para mover o detener el cursor, por lo que si cambia esta referencia (por ejemplo por aflojarse el elástico que fija el dispositivo a la cabeza) es necesario reiniciar el dispositivo para volver a controlar el cursor de forma confiable. Dado que este problema no fue habitual a lo largo de su uso y que es fácilmente solucionable, no afecta la evaluación del modo, que se considera funcional.

8.2.4. Modo Gestos

Este modo se basa en detectar los gestos del usuario, para luego traducirlos en acciones del mouse como fue explicado en la Subsección 2.2.4.

Para evaluar este modo se realizaron las pruebas de usabilidad definidas al inicio de esta sección. Tanto en el juego "Buscaminas" como en la prueba del teclado en pantalla se observaron resultados satisfactorios, ya que este modo permite frenar rápidamente el cursor y además hacer un *click* izquierdo en el mismo gesto, que resulta muy ventajoso para estas dos pruebas.

Con respecto a la prueba relacionada al Explorador de archivos se pudo navegar por los directorios sin muchos inconvenientes, asi como abrir archivos. Sin embargo se notaron algunas deficiencias del modo; la ausencia del *click* derecho, y la imposibilidad de realizar acciones complejas como *click&drag* limitan mucho las acciones que se pueden realizar en esta prueba. A su vez como fue explicado en la Subsección 4.5.5, en este modo se observan dificultades al cambiar de dirección de forma ágil, lo que dificulta el manejo del cursor.

Las pruebas antes mencionadas se realizaron con el dispositivo sujeto a la muñeca, y con esta apoyada sobre una mesa. Al realizar las mismas pruebas con movimientos en el aire se observaron algunas diferencias en el desempeño. Principalmente se notó que algunos gestos realizados no generaban movimientos del cursor. Esto se debe a la dificultad de ejecutar movimientos en un plano horizontal, que pueden venir acompañados de movimientos en el eje Z o un *shake*.

Otro problema notado en todas las pruebas se da al realizar movimientos "lentos", como se explicó en la Subsección 4.5.5. Cuando esto sucede el gesto puede no ejecutar ninguna acción del cursor o mover el mismo en una dirección incorrecta.

A pesar de estas limitantes se considera que el dispositivo resulta útil, ya que permite al usuario controlar el cursor de forma cómoda, siempre teniendo en cuenta que es necesario un proceso de adaptación y aprendizaje de cómo funciona este modo.

8.3. Mejoras

Si bien los objetivos del proyecto fueron alcanzados en gran medida, aún restan algunas mejoras a realizar sobre el prototipo logrado antes de pensar en trabajos futuros;

tanto a nivel de *firmware* como de *hardware*.

A nivel de *firmware*, todavía queda por salvar la limitante introducida por la fusión de los sensores, que impone una restricción sobre el tiempo de duración de los gestos. Si bien los gestos individuales no se ven afectados por esta limitante, la concatenación fluida de los mismos sí. Esto hace que sea necesaria una pausa entre gesto y gesto (o cada dos gestos) que si bien no convierte al dispositivo en inutilizable, lo aleja bastante de lo deseado. Sería ventajoso extender a los modos Cabeceo y Gestos la capacidad de mantener el *click* presionado, así como de ejecutar la acción de *click* derecho, logrando a la vez que estas acciones se puedan hacer en simultáneo con un movimiento del cursor.

Si bien lo analizado en la Subsección 7.3.2 muestra que la autonomía del dispositivo es suficiente, y que las dimensiones de la batería no limitan a las del dispositivo, esto no implica que no se pueda disminuir el consumo. En este aspecto tanto el microprocesador como el *MARG* admiten modos de bajo que consumo que no fueron tenidos en cuenta en este desarrollo, sin embargo esto impacta en la corriente total consumida.

También sería deseable reducir el tamaño del dispositivo, tanto por comodidad de los usuarios como pensando en masificar el producto. Esto se puede lograr fácilmente desarrollando un *PCB* de cuatro capas y/o eliminando componentes excesivamente grandes (*encoder*, conector *RJ45*, y conector *micro USB*).

En lo referente al encapsulado logrado, la terminación presenta una rugosidad que puede no ser la ideal para un dispositivo de esta naturaleza. Al estar (en algunos modos) en contacto con la piel o cuero cabelludo, sería apropiado garantizar que sus características físicas sean tales que no le ocasionen al usuario ninguna molestia o lastimadura tras el uso prolongado. Además, sería necesario asegurar que el método de sujeción no se afloje una vez colocado el dispositivo, así como garantizar que no le genere incomodidad al usuario.

Capítulo 9

Conclusiones y Trabajos Futuros

En esta sección se dará un cierre a la presente documentación, dejando las conclusiones realizadas sobre el proyecto y su ejecución, así como también remarcando una serie de lineamientos en dirección a profundizar lo hasta aquí realizado.

9.1. Conclusiones

Se logró diseñar y construir un dispositivo inalámbrico de interfaz humana capaz de sustituir un *mouse* convencional. El mismo tiene la ventaja de ser *Plug&Play*, y de no necesitar la instalación de ningún tipo de *software* externo para utilizarlo.

Se diseñó e implementó el *firmware* que permite controlar el cursor del *mouse* con mayor o menor interacción del usuario, ya sea mediante el accionado de botones o movimientos del dispositivo. Para esto último se lograron dos métodos para detectar y clasificar distintas acciones del usuario. Debido a esta variedad de modos de uso es posible cubrir diversas necesidades y/o preferencias de los usuarios.

Se diseñó el *hardware* del dispositivo de manera tal que el mismo pueda ser vestible, seleccionando los componentes necesarios y diseñando un *PCB* para esto. Para agregar el soporte de botones se diseñó un segundo *PCB*, cuyo conexionado es opcional. Además se logró ofrecer flexibilidad tanto en la forma como en el tamaño de los botones, así como en su fabricación. Se completó este diseño fabricando un encapsulado para cada *PCB* y proponiendo un método de sujeción para el vestible.

Se generaron instancias de dialogo con profesionales de la Escuela Nº 200 y Casa de Gardel, lo que permitió dar un primer paso en la comprensión de las necesidades de los potenciales usuarios.

A lo largo del proyecto se emplearon conocimientos adquiridos en la carrera. A su vez, se obtuvo experiencia en el proceso de desarrollo de *PCB* y encapsulado, así como familiaridad con herramientas de diseño *CAD*, tanto para el diseño del circuito impreso como para el modelado 3D. Además se adquirió experiencia en el trato profesional en

grupos interdisciplinarios.

9.2. Trabajos Futuros

Este proyecto da pie a diversos trabajos a realizar, pensando en el largo plazo. En pos de lograr un mejor desempeño del dispositivo se requiere un profundo análisis de la señales involucradas en la detección y clasificación de acciones del usuario. Teniendo en cuenta esto, se puede enfocar el problema desde una perspectiva de tratamiento de señales o aprendizaje automático.

Apuntando a conseguir que el dispositivo satisfaga las necesidades de una mayor cantidad de usuarios es preciso apuntar a un proyecto mucho mas integral, que abarque profesionales de varias áreas y especialidades. Con esto se podrán encontrar, atacar y resolver una serie de problemas que quedan fuera de los conocimientos específicos de la ingeniería eléctrica, consiguiendo un producto final que allane un poco el camino rumbo a una mayor inclusión de todas las personas.

Apéndice A

Proceso de Desarrollo e Interacción con el Cliente

En este apéndice se presentará el proceso de desarrollo realizado para definir los prototipos.

Inicialmente se pensó en diseñar un dispositivo vestible para cualquier parte del cuerpo, que tuviese la capacidad de detectar gestos (movimientos predefinidos), y que tradujera estos gestos a movimientos del cursor del *mouse*. Para esto se definieron requerimientos primarios de *hardware*, con la motivación de seleccionar componentes para validar el concepto.

A efectos de enriquecer este proceso se tuvo en cuenta la opinión y validación del cliente, lo cuál introdujo cambios tanto en el *hardware* a utilizar como en la solución desarrollada. Parte de este proceso se detallará más adelante.

A.1. Estado del Arte

Con el objetivo de tener un punto de partida, y para entender qué resultados era razonable esperar, se hizo una investigación sobre el estado del arte en lo referente a soluciones similares a la que se pretende; los resultados obtenidos se pueden observar en la tabla A.1.

Nombre	Precio aproximado	Plug&Play	Click izquierdo	Click derecho	Movimiento del cursor	Click y movimiento simultáneo	Usable solo con botones	Usable solo con movimientos
Quha Zono [34]	USD 1000	No	Si	Si	Si	No	No	Si
5 Switch [12]	USD 129	Si	Si	Si	Si	Si	Si	No
enPathia [18]	227 €	Si	No	Si	Si	Si (con accesorio)	No	Si
Integraouse [32]	USD 2500	Si	Si	Si	Si	Si	No	Si
The New PCEye [16]	1249 €	No	Si	Si	Si	Si	No	Si
IntelliGaze [22]	8420 €	No	Si	Si	Si	Si	No	Si
SAM-Joystick [15]	USD 229	No	Si	Si	Si	No	No	No
BJOY Ring Wireless [9]	753.50 €	No	Si	No	Si	No	No	Si
Jouse 3 [13]	USD 1495	Si	Si	Si	Si	No	No	Si
BJOY Chin [8]	753.50 €	Si	Si	Si	Si	No	No	No

Tabla A.1: Tabla extensiva comparativa de soluciones disponibles en el mercado

A.2. Cambios Introducidos a Raíz de Reuniones con Funcionarios de la Escuela Nº 200

A efectos de validar las decisiones tomadas en el desarrollo, contamos con el apoyo de funcionarios de la Escuela N°200 y de Casa de Gardel, con quienes tuvimos reuniones en las distintas etapas del proyecto. A continuación se resumen lo puntos más importantes de estas reuniones.

Primera reunión

En una primera instancia, y debido a lo delicado de la situación sanitaria, el contacto con la Escuela N° 200 se realizó virtualmente. En esa reunión se encontraban presentes autoridades de la escuela, así como maestras de la misma. Los objetivos de dicha reunión eran esencialmente:

- Validar la idea del proyecto.
- Conocer las herramientas que poseían para facilitar el acceso a las computadoras.
- Escuchar las propuestas y opiniones que la gente de la escuela pudiera aportarnos.

Con respecto a la validación de la idea, las opiniones preliminares pueden considerarse como positivas, sin embargo no permitieron sacar grandes conclusiones debido a las complejidades de transmitir la idea sin entrar en tecnicismos, sumado a las dificultades propias que agrega una reunión virtual. No obstante, en dicha reunión surgió la idea de agregar la funcionalidad de botonera para mover el cursor, sustentado principalmente en las necesidades y apreciaciones de las maestras de la escuela. Finalmente, se pudo sacar información general sobre alguno de los dispositivos y herramientas que la escuela utilizaba. Dentro de los mencionados, los más destacados fueron:

- Enable Viacam (Emulador de ratón vía webcam).
- Tobii Eye Tracker.
- Pulsadores.

Soluciones como el seguidor de ojos o el emulador de *mouse* vía cámara web distan bastante de nuestra propuesta, pero nos permitieron integrar alternativas como el barrido del cursor, basándonos en su forma de uso.

Segunda reunión

Una vez que consideramos que teníamos una versión lo suficientemente madura, decidimos presentarla ante nuestro cliente.

Amparados en el aplaque momentáneo del avance de la pandemia en Uruguay, y aprovechando la reanudación de las actividades en las escuelas, tuvimos la oportunidad de reunirnos presencialmente en la Escuela Nº 200 a mediados del mes de octubre de 2020.

Ese primer contacto presencial fue sumamente enriquecedor, ya que nos permitió adentrarnos un poco más en la realidad de la escuela y de sus alumnos. Dirigidos por las maestras y autoridades de la institución, pudimos tener contacto con algunos de los niños que asisten a la escuela y observar en detalle las dificultades que se les presentan al momento de utilizar una computadora.

A su vez, pudimos tener contacto y ver en acción algunas de las herramientas que poseen, con la finalidad de analizar las ventajas y desventajas de dichas soluciones.

Por último tuvimos una instancia donde el grupo mostró algunos prototipos muy preliminares de las soluciones propuestas, con la finalidad de recaudar comentarios, opiniones y criticas de los mismos. Las alternativas presentadas fueron:

- Movimiento del cursor mediante gestos.
- Movimiento del cursor accionado por inclinación.
- Barrido de la pantalla accionado por botones.

Además se presentó, a través de un vídeo, el funcionamiento de una versión preliminar del modo Botonera.

En una primera instancia, la solución del movimiento accionado por inclinación no fue del todo bien recibida, ya que a los docentes se les dificultó imaginarse un escenario donde esa propuesta les resultara realmente útil. Luego de algún intercambio de opiniones, se concluyó que podía ser útil para los casos en los que se acoplara a un dispositivo u objeto externo, como puede ser una palanca o un joystick. No obstante, por parte de las maestras no se le vio demasiado potencial para ser usado en la cabeza, siendo esta la forma en la que esa idea había sido concebida inicialmente.

La versión del dispositivo accionado por gestos presentada consistía en que un movimiento en una de las cuarto direcciones principales (arriba, abajo, izquierda y derecha) accionaba un movimiento del cursor en esa dirección, mientras que un movimiento en la dirección contraria generaba el frenado del mismo. A pesar de ser una versión muy preliminar y precaria, fue en general bien recibida. Incluso fue utilizada por alguno de los presentes, y surgieron algunos escenarios posibles donde esa solución podría ser útil en el trabajo diario con los niños.

Finalmente, las soluciones del barrido y los botones fueron consideradas prácticas y útiles; con respecto a esto, generó interés y atrajo la idea de poder fabricar y adaptar de manera sencilla nuevos botones a nuestra solución propuesta.

Debido al fin de año lectivo, sumado a las medidas adoptadas por el aumento de casos de COVID-19 en el país, esta fue la última reunión que tuvimos con funcionarias de la Escuela N° 200.

A.3. Cambios Introducidos a Raíz del Uso del Dispositivo

Una vez definidos los cuatro modos del dispositivo se trabajó en implementar y mejorar la usabilidad de cada uno de ellos. Para esto fue crucial tener *feedback* por parte de los tutores y de una especialista que trabaja con potenciales usuarios del dispositivo.

El modo que más iteraciones necesitó fue el modo Gestos. En un comienzo se planteó que este modo se dividiera en dos submodos: el discreto y el continuo. Este último

pretendía mapear directamente los movimientos del dispositivo con los movimientos del cursor (de forma similar a como hace un *mouse* convencional), mientras que el modo discreto inicia el movimiento del cursor al detectar un gesto, y mantiene este movimiento de forma autónoma. Al recibir devoluciones negativas sobre la usabilidad del modo continuo se decidió eliminarlo y continuar trabajando con el discreto.

Una vez enfocados en el modo discreto, se realizaron cambios principalmente en los umbrales de detección, y en las acciones que debe realizar el usuario para frenar el dispositivo. Para frenar el cursor se planteó la idea de utilizar los mismos gestos. Si el usuario realizaba un gesto mientras el cursor estaba en movimiento lo frenaba, y para que se comenzara a mover nuevamente debía realizar otro gesto. A través de las devoluciones recibidas se llegó a la conclusión de que esta solución no era practica y que implicaba realizar demasiados gestos para mover el cursor. Finalmente se optó por definir gestos particulares tanto para el frenado, como para el frenado acompañado de un *click* izquierdo.

Con respecto al modo Botonera el cambio más importante fue en la sensibilidad con la que los sensores capacitivos pueden detectar los cambios de capacidad. Dicha sensibilidad se logró disminuir al agregar capacitores en paralelo a los electrodos que funcionan como botones.

Sobre el modo Barrido no se recibieron muchos comentarios, únicamente se planteó que se diera la posibilidad de controlar la velocidad con la que se mueve el cursor, lo cual se solucionó con la integración del *encoder*.

Por último el modo Cabeceo también requirió muy pocas iteraciones para mejorarlo, la principal devolución fue respecto al funcionamiento, en donde se planteó definir mejor la zona de frenado, lo cual se logró modificando los umbrales.

Apéndice B

Sobre el *Hardware*

B.1. *Hardware* Necesario para la Prueba de Concepto

A continuación se detallan las placas de desarrollo y componentes utilizados para la prueba de concepto.

Hardware	Componente seleccionado
MARG	LSM6DSOX + LIS3MDL 9 DOF IMU
Com. inalambrica + microcotrolador	FEATHER NRF52 BLUEFRUIT LE - NRF
Sensor capacitivo	CAPACITIVE TOUCH BREAKOUT AT42QT
Memoria externa	IC EEPROM 32K SPI 20MHZ 8SO

Tabla B.1: *Hardware* requerido

B.2. Lista de Materiales de las Placas Desarrolladas

A continuación se detallan los componentes usados en los *PCB* desarrollados; se menciona tanto el nombre del componente, cómo su valor (cuando corresponde), y su ubicación en la placa.

Part	Value	Device
C1	22 pF	C-EUC0603K
C2	22 pF	C-EUC0603K
C3	10 uF	C-EUC0201
C4	1 uF	C-EUC0603K
C5	100 nF	C-EUC0201
C6	100 nF	C-EUC0201
C7	1 uF	C-EUC0201
C8	100 nF	C-EUC0201
C9	300 pF	CAP_CERAMIC_0603MP
C10	4.7 uF	C-EUC0201
C11	1 uF	C-EUC0201
C13	300 pF	CAP_CERAMIC_0603MP
C15	100 nF	C-EUC0603K
C17	10 µF	CAP_CERAMIC0805-NOOUTLINE
C18	1 µF	CAP_CERAMIC0805-NOOUTLINE
CN1	S2B-PH-KL	JST_2PIN-SMT
CR1	M95320-WMN6TP	M95320-WMN6TP
D1	TZS4688	ZENER-DIODESOD80C
IC1		ADAFRUIT_SENSOR_LIS3MDL
JP2		HEADER-1X12MM
JP3		HEADER-1X12MM
L2	10uH	INDUCTOR0805-NO
LED1		LED
LED2		LED
LED3		LED
Q1	CM8V-T1A-32.768KHZ-12.5PF-20PPM-TB-QA	FC-12M
R1	1.5 $k\Omega$	R-EU_R0201
R2	1.5 $k\Omega$	R-EU_R0402
R3	10 $k\Omega$	R-US_R0603
R7	1$k\Omega$	RESISTOR_0603_NOOUT
R9	10$k\Omega$	R-EU_R0805
R10	10$k\Omega$	R-EU_R0805
S1		DS01
SW3	EC12E2424407	EC12E_SW
U$8		MICROBUILDER_LSM6DSOX
U1		NRF52832_MODULE_MDBT42
U2	AP2112K-3.3TRG1	VREG_SOT23-5
U3	MCP73831T-2ACI/OT	MCP73831/2
X1	GLX-S-88M	GLX-S-88M
X3		USB_MICRO_20329_V2

Tabla B.2: BOM correspondiente al dispositivo principal

Part	Value	Device
C1	3 nF	C-EUC0402K
C2	4 nF	C-EUC0402K
C3	3 nF	C-EUC0402K
C4	4 nF	C-EUC0402K
C5	3 nF	C-EUC0402K
C6	4 nF	C-EUC0402K
C7	3 nF	C-EUC0402K
C8	4 nF	C-EUC0402K
C9	2 nF	C-EUC0402K
C10	2 nF	C-EUC0402K
C11	2 nF	C-EUC0402K
C12	2 nF	C-EUC0402K
C13	2 nF	C-EUC0402K
C14	2 nF	C-EUC0402K
C15	3 nF	C-EUC0402K
C16	4 nF	C-EUC0402K
C17	3 nF	C-EUC0402K
C18	4 nF	C-EUC0402K
R1	$10k\Omega$	R-EU_R0805
R2	$10k\Omega$	R-EU_R0805
R3	$10k\Omega$	R-EU_R0805
R4	$10k\Omega$	R-EU_R0805
R5	$10k\Omega$	R-EU_R0805
R6	$10k\Omega$	R-EU_R0805
U$2	AT42QT1010 - UDFN	AT42QT1010UDFN
U$3	AT42QT1010 - UDFN	AT42QT1010UDFN
U$4	AT42QT1010 - UDFN	AT42QT1010UDFN
U$5	AT42QT1010 - UDFN	AT42QT1010UDFN
U$6	AT42QT1010 - UDFN	AT42QT1010UDFN
U$7	AT42QT1010 - UDFN	AT42QT1010UDFN

Tabla B.3: BOM correspondiente al dispositivo secundario

Esta página ha sido intencionalmente dejada en blanco.

Apéndice C

Herramientas y Tecnologías

C.1. Definiciones, Nomenclatura y Comentarios sobre *Bluetooth*

C.1.1. Funcionamiento

El protocolo *Bluetooth* opera en la frecuencia de 2,4 GHz, en la que también funcionan otros protocolos de radiofrecuencia como *WiFi* y *ZigBee*.

Las redes de *Bluetooth* funcionan en un modelo maestro/esclavo para definir cuando los dispositivos pueden comunicarse. En este modelo, un único dispositivo maestro se puede conectar con hasta siete esclavos. Cada esclavo puede conectarse únicamente a un maestro.

El maestro coordina las comunicaciones a través de la red, por lo que puede enviar y solicitar datos de cualquiera de los esclavos. Los esclavos solo tienen permitido transmitir y recibir del maestro; no pueden hablar con otros esclavos de la red.

C.1.2. Direcciones y Nombres

Cada dispositivo *Bluetooth* tiene una dirección de 48 *bits*, comúnmente referida como *BD_ADDR*. Esta dirección típicamente se presenta como doce dígitos hexadecimales. Los 24 *bits* más significativos, componen un identificador de organización, el cual identifica al fabricante. La mitad menos significativa corresponde a la parte única de la dirección.

Los dispositivos *Bluetooth* también pueden tener nombres más amigables con el usuario. Estos nombres son los que en general se muestran al listar dispositivos desde una interfaz de sistema operativo. Las reglas para estos nombres son mucho menos restrictivas; se componen de hasta 248 *bytes* y pueden duplicarse entre distintos dispositivos.

C.1.3. Procedimiento de Conexión

Crear una conexión *Bluetooth* entre dos dispositivos es un proceso compuesto de una progresión de tres etapas.

- Consulta: Si dos dispositivos *Bluetooth* no conocen su existencia entre sí, uno de ellos debe enviar una petición para descubrir al otro. Un dispositivo transmite en el medio un mensaje de consulta y, cualquier dispositivo escuchando por este tipo de mensaje le responde con su dirección y, opcionalmente, su nombre u otra información.
- Conectando: Es el proceso de establecer una conexión.
- Conectado: Una vez establecida la conexión, se entra al estado conectado. Mientras esté conectado un dispositivo puede participar activamente en una comunicación, o puede ser puesto en uno de los modos de bajo consumo.
    - Modo activo: Es el modo regular de conexión, cuando el dispositivo está enviando o recibiendo datos activamente.
    - Modo *sniff*: En este modo un dispositivo va a escuchar a intervalos predefinidos (e.g. cada 100 ms).
    - Modo *hold*: Cuando un dispositivo está en este modo dejará de escuchar por un período de tiempo, tras el cual volverá automáticamente a modo activo. Un maestro puede imponer este modo en un esclavo.
    - Modo *park*: Este es el modo de menor consumo; en este modo un esclavo quedará inactivo hasta que el maestro le envíe un comando para volver al modo activo.

C.1.4. Emparejamiento

Cuando se desea que dos dispositivos se conecten entre sí siempre que sea posible, pueden ser emparejados. Los dispositivos emparejados establecen una conexión siempre que estén los suficientemente cerca entre sí.

El emparejamiento se crea mediante un proceso que se realiza una sola vez. Esto produce que los dispositivos además de compartir su nombre y dirección en la etapa de consulta, guarden lo obtenido en la memoria para usarlo en un futuro. Además, ambos dispositivos intercambian una clave, que será utilizada para establecer futuras conexiones.

C.1.5. Perfiles *Bluetooth*

Los perfiles *Bluetooth* son protocolos adicionales que se construyen sobre el protocolo estándar para definir qué tipo de datos está transmitiendo el dispositivo. Mientras que la especificación de *Bluetooth* define como funciona la tecnología, los perfiles definen cómo es usada.

El (los) perfil(es) que soporta un dispositivo, determina(n) para que aplicación(es) está diseñado. Por ejemplo unos auriculares manos libres usarán el perfil HSP (*Headset Profile*), mientras que un *mouse* utilizará el perfil HID (*Human Interface Device*). Para que dos dispositivos sean compatibles deben soportar el mismo perfil. A continuación se nombran algunos de los perfiles más utilizados.

Serial Port Profile (SPP)

SPP es un perfil para enviar ráfagas de datos entre dos dispositivos, tal como si fuera RS-232 o UART. Es uno de los perfiles más básicos.

Hands-Free Profile (HFP) y *Headset Profile* (HSP)

Estos perfiles están diseñados para enviar y recibir señales de audio; en particular están pensados para realizar llamadas telefónicas en modo "manos libres". La diferencia entre ambos perfiles, es que el primero, además de soportar la transmisión y recepción de llamadas, permite las interacciones típicas de una llamada telefónica (atender, cortar, rechazar llamadas, etc).

Advanced Audio Distribution Profile (A2DP)

Este perfil define cómo transmitir audio de un dispositivo a otro. A diferencia de los perfiles anteriores que establecían un flujo bidireccional de audio, este perfil sólo permite la transmisión en un sentido; esta "limitación" es el resultado de que la calidad de audio que se puede transmitir en este perfil es mucho mayor.

A/V Remote Control Profile (AVRCP)

Permite el control de otro dispositivo *Bluetooth*; este perfil en general es implementado junto con A2DP para aumentar las funcionalidades de auriculares Bluetooth (cambiar canción, pausar, etc.).

Human Interface Device (HID)

Es el perfil utilizado para tomar entradas del usuario (mouse, teclado, joysticks). El mismo es una extensión del perfil HID definido para USB. De la misma manera que el perfil SPP pretendía sustituir el RS-232, HID pretende hacerlo con USB. Este es el perfil que se utilizó a lo largo del proyecto, por lo que fue en el que se profundizó más.

Una comunicación entre el sistema operativo y un dispositivo que se presente como HID tiene dos etapas esencialmente. En la primer etapa, de emparejamiento, el dispositivo declara qué parámetros van a estar incluidos en los envíos y en qué formato. De esta forma, en los siguientes envíos, solo se tienen que transmitir los valores en el orden indicado, disminuyendo así significativamente el tamaño de los envíos. En nuestro caso en particular, debemos especificar si los valores de posición enviados son absolutos o relativos a la posición actual del cursor, y en qué rango están estos valores. De esta forma el sistema operativo mapea cualquier valor en el rango especificado en una posición en la pantalla.

Esto presenta un desafío, ya que precisamos que el dispositivo sea capaz de funcionar en modo relativo y absoluto debido a que el barrido tiene que ir de extremo a extremo de la pantalla (y no se sabe a priori el tamaño de la misma), mientras que los otros modos requieren que los movimientos sean relativos.

Una posible solución a esto sería desvincular el dispositivo de la computadora previo a ponerlo en modo Barrido, lo cual resulta poco práctico. La solución que se implementó,

aprovechando que el procesador seleccionado cuenta con RAM suficiente, fue presentarse al sistema operativo con dos perfiles HID distintos, y hablar desde uno u otro según corresponda.

C.1.6. Versiones típicas de *Bluetooth*

Bluetooth ha estado en constante evolución desde su invención en 1994 siendo la última versión *Bluetooth* v5.2. Sin embargo muchas de las versiones anteriores siguen en uso en distintas aplicaciones. Por esto se lista a continuación las versiones más comunes.

- *Bluetooth* v1.2: Fue la última versión 1.X; soporta velocidades de hasta 1 $Mbps$ y hasta 10 m de rango máximo.
- *Bluetooth* v2.1+EDR: Esta versión introdujo una mejora en la tasa de datos (EDR por las siglas en inglés) la cual permite velocidades de hasta 3 $Mbps$. Se agregó además, SSP (*Simple Secure Pairing*) lo que permitió aumentar el uso de seguridad en las conexiones. Esta versión es todavía usada por microcontroladores de baja velocidad.
- *Bluetooth* v3.0+HS: Esta versión usa Bluetooth para establecer y manejar la conexión, pero la transmisión se hace sobre WiFi (802.11) de esta forma se alcanzan velocidades de hasta 24 $Mbps$.
- *Bluetooth* v4.0+BLE: Esta versión dividió la especificación en tres categorías: clásica, alta velocidad y bajo consumo. Las primeras dos categorías se remiten a las versiones v2.1+EDR y v3.0+HS respectivamente, mientras que la última hace referencia al modo bajo consumo (BLE).

C.2. ROS y Plotjuggler

Para el correcto análisis de las señales (en particular para poder visualizarlas en tiempo real) y posterior depuración del código se utilizaron dos herramientas que fueron de suma utilidad: *ROS* y *Plotjuggler*. A continuación se explicará un poco más sobre estas dos utilidades, y como interactúan entre ellas.

C.2.1. ROS

Robot Operating System, más conocido por su sigla *ROS* es un entorno de trabajo pensado para el desarrollo de software para robots, que brinda una serie de librerías y funcionalidades que simplifican esta tarea. *ROS* posee compatibilidad con *C++* y *Python*, pudiendo ser usado en una gran variedad de microcontroladores. A su vez, la comunidad de usuarios se encuentra actualizando y desarrollando continuamente librerías y paquetes. Dos de los conceptos fundamentales para la utilización de ROS son los "nodos" y los "tópicos", que se explicarán brevemente a continuación.

Nodos

Se denominan como "nodos" a aquellos procesos encargados de realizar cálculos computacionales. En esencia, los nodos se encargan de ejecutar ciertas acciones y se comunican con otros nodos a través principalmente de "tópicos". En general, cada funcionalidad es implementada mediante la realización de un nodo, generando que existan muchos nodos interconectados e intercambiando información entre ellos.

Tópicos

Una parte fundamental en el funcionamiento de ROS son los denominados "tópicos". A través de ellos es que se realizan todas las comunicaciones entre nodos, ya sean internas o externas. Estos "tópicos" se caracterizan por una etiqueta, que es esencialmente el nombre que se le brinda al momento de crearlo. Cualquier nodo tiene la capacidad de publicar en un tópico, y los nodos que estén en la red podrán suscribirse a él y recibir los mensajes que allí se publiquen.

C.2.2. Comunicación PC-Microcontrolador

Un caso típico de uso es la interconexión de una *PC* y un microcontrolador; teniendo en cuenta que las computadoras tienen un gran poder de procesamiento y mayores recursos, es normal que gran parte de la lógica "pesada" se ejecute en un nodo corriendo en la computadora, y sea notificada mediante tópicos a otros nodos ejecutándose en el/los microcontroladores, alivianando así la carga computacional de estos últimos. Para el caso particular del proyecto, únicamente se utiliza esta comunicación para la visualización de señales en tiempo real, así como para la depuración del sistema.

C.2.3. Plotjuggler

Plotjuggler es una herramienta de visualización de datos compatible con *ROS*. Al ejecutarlo, genera un nodo que se suscribe a los tópicos correspondientes y, siempre que los datos sean numéricos, permite graficarlos en tiempo real a medida que son recibidos. Además de visualizarlos, tiene funcionalidades que permiten procesar los datos y observar los resultados.

Apéndice D

Manual de Usuario

D.1. Carga del Dispositivo

El dispositivo cuenta con una batería recargable, la cual se recarga por la entrada *micro USB*. A continuación se explican los pasos para cargarla.

D.1.1. Instrucciones para Cargar el Dispositivo

1. Conecte el conector *micro USB* del cable *USB* en el conector *micro USB* del dispositivo.
2. Conecte el conector *USB* del cable (el otro extremo del cable) a una *PC*, o a un cargador con entrada *USB*.
3. La batería comenzará a cargarse. Su carga se completará luego de aproximadamente 4 horas, pero si desea puede cargarla durante menos tiempo.
4. Una vez que decida finalizar la carga desconecte el cable *USB* del dispositivo.

D.2. Encendido

Advertencia: para que el dispositivo encienda debe tener un mínimo de carga. Si no enciende cargue el dispositivo según explica la Sección D.1.1.

D.2.1. Instrucciones para Encender el Dispositivo

Mueva el interruptor del dispositivo (figura D.1, recuadro 1) hacia el lado ON del mismo.

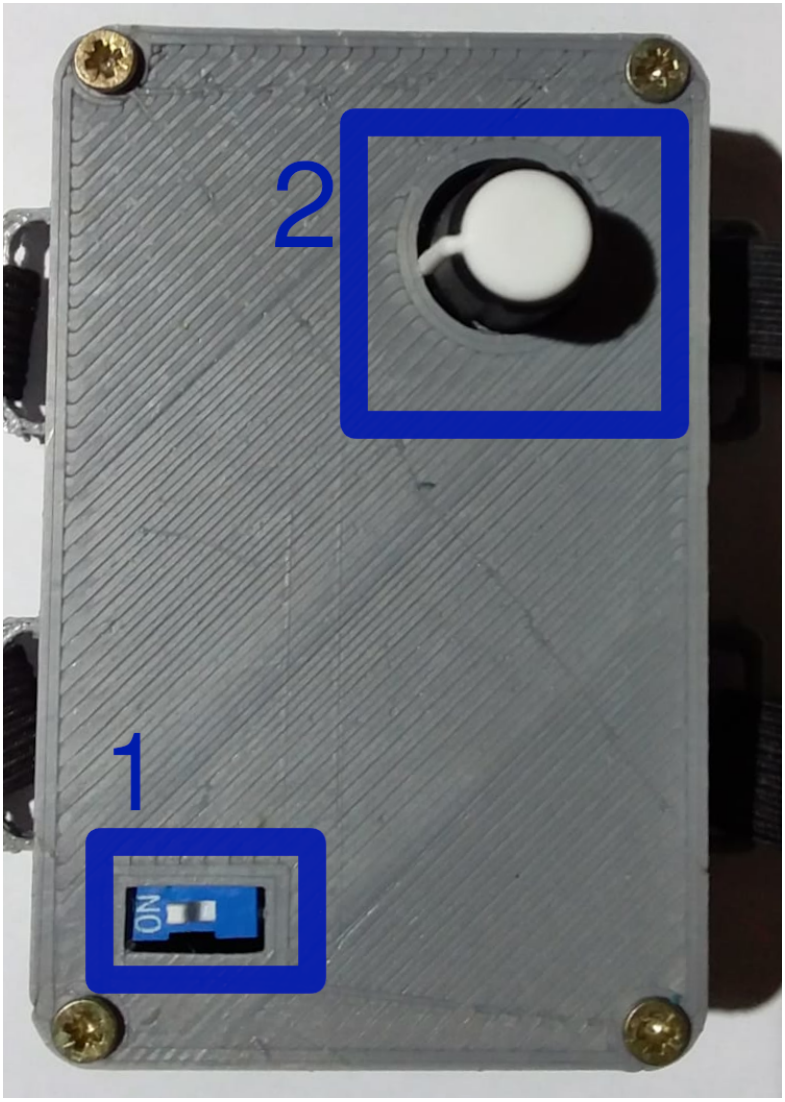

Figura D.1: Vista superior del dispositivo

D.3. Conexión con PC o Dispositivo Móvil

El dispositivo se comunica con la *PC* o el dispositivo móvil que elija mediante *Bluetooth*. A continuación se explica el proceso para agregar dispositivos *Bluetooth* tanto en *Windows 10*, como en *Ubuntu 18.04*.

D.3.1. *Windows 10*

Para agregar un dispositivo *Bluetooth* en *Windows 10* es necesario abrir las opciones correspondientes a "*Bluetooth* y otros dispositivos" (figura D.2). Para esto basta con entrar a las configuraciones del sistema, escribir *"Bluetooth"* y elegir la opción correspondiente. Luego es necesario activar el *Bluetooth* si este no estuviera activado (recuadro

1 de la figura). El siguiente paso es hacer *click* en "Agregar *Bluetooth* u otro dispositivo" (recuadro 2). Una vez hecho esto se abrirá una ventana como la mostrada en la figura D.3; donde se debe hacer *click* en la opción recuadradada *"Bluetooth"*.

Una vez seleccionado esto se abrirá una nueva ventana, similar a la de la imagen D.4, en la que se irán listando los dispositivos encontrados. El dispositivo deseado tiene el nombre "ININ", seleccionarlo y el proceso de conexión habrá terminado.

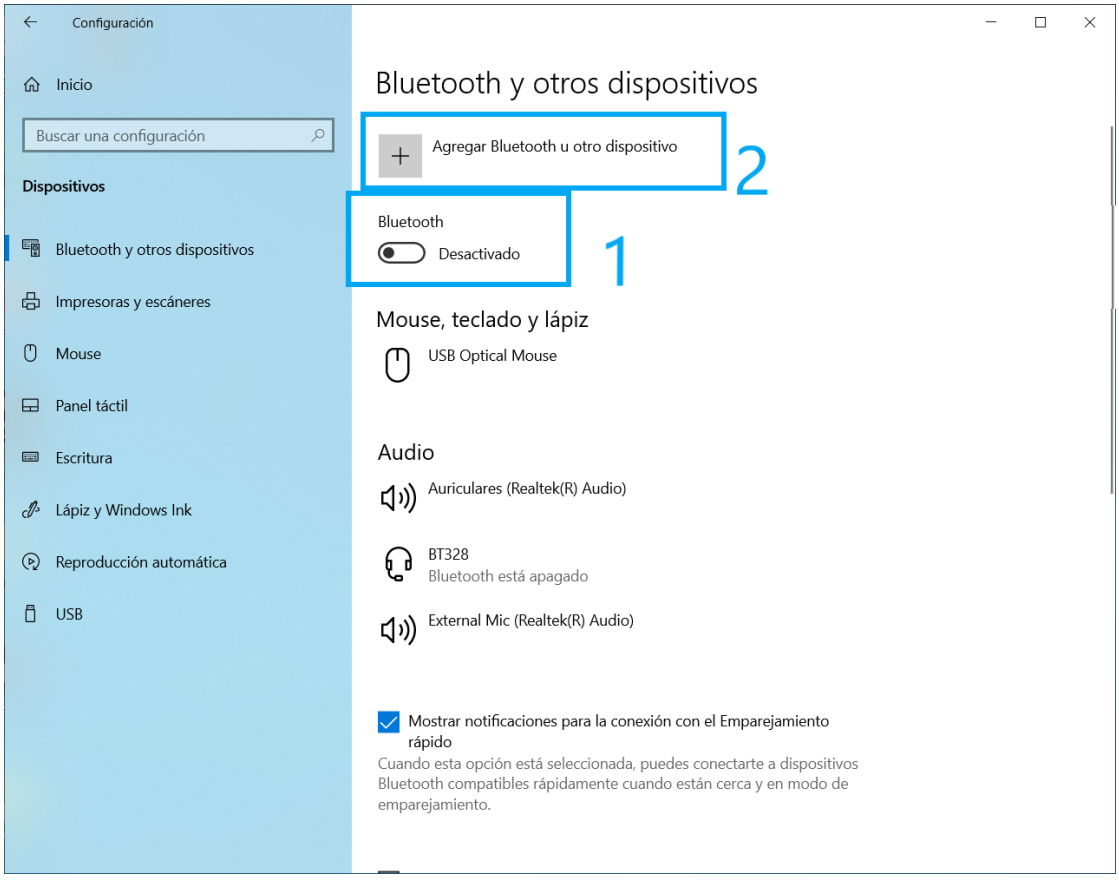

Figura D.2: Encendido de *Bluetooth* en Windows 10

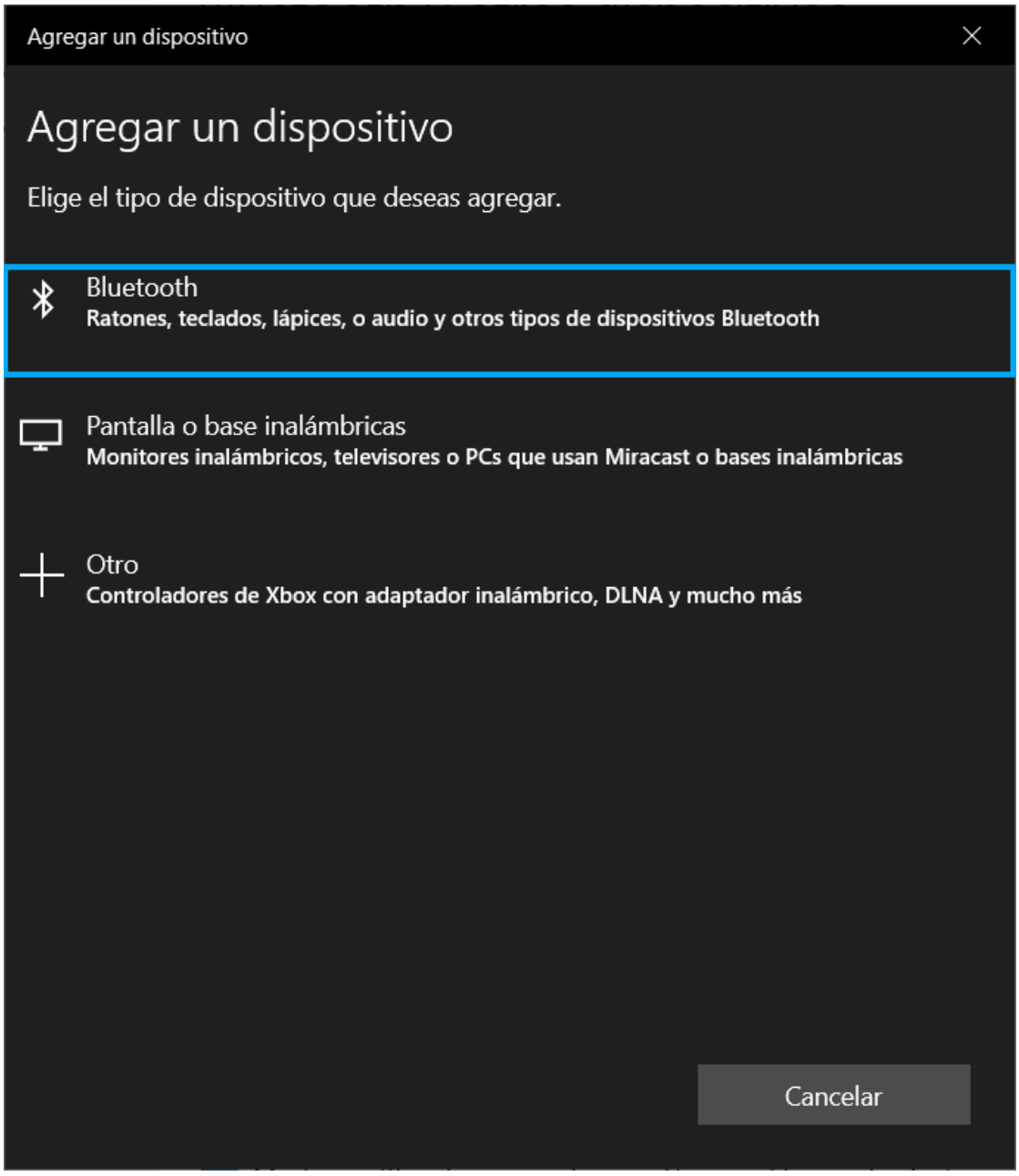

Figura D.3: Agregar dispositivo en *Windows 10*

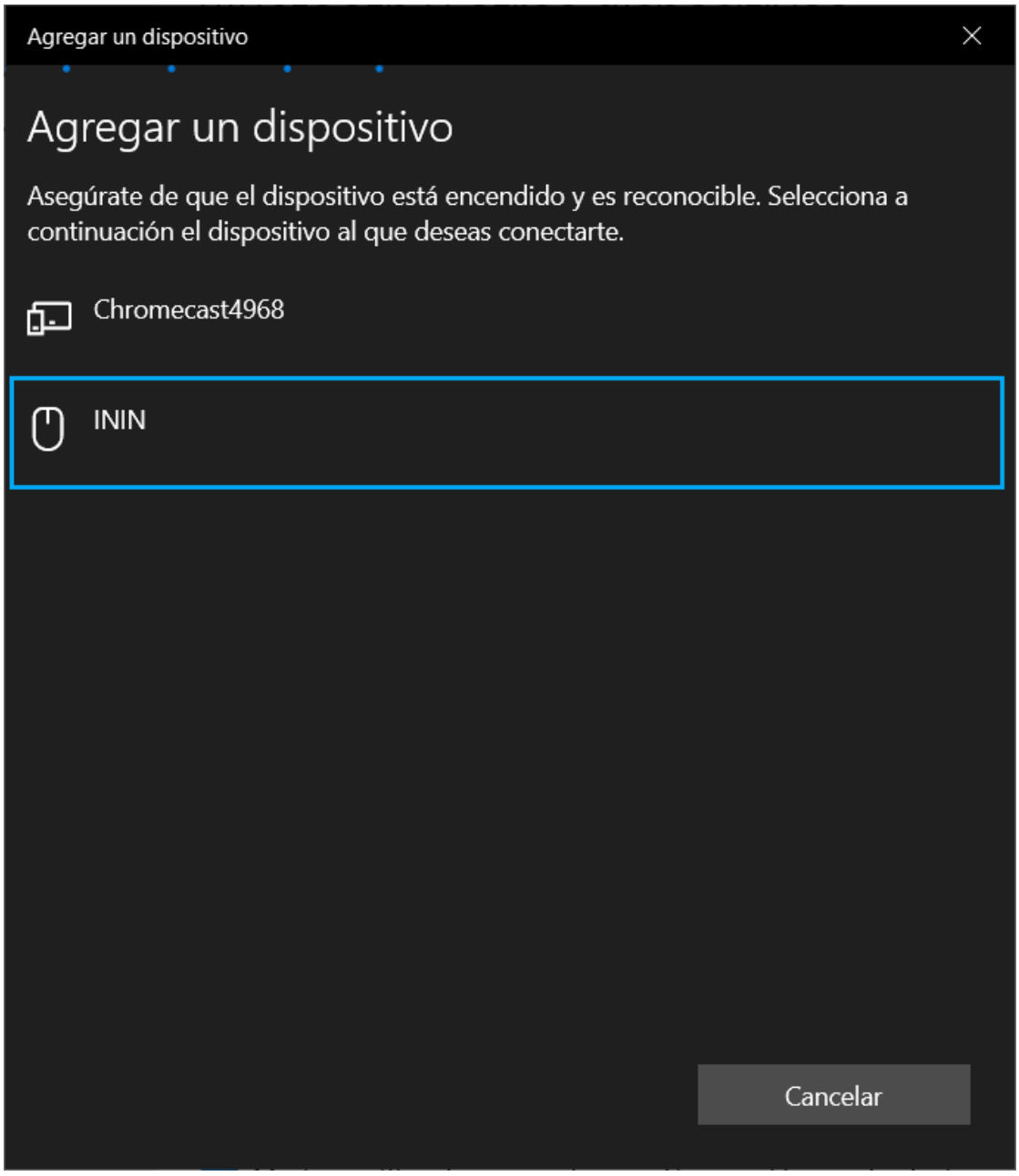

Figura D.4: Conexión de dispositivo ININ en *Windows 10*

D.3.2. Ubuntu

Para agregar un dispositivo *Bluetooth* en *Ubuntu* es necesario activar el *Bluetooth* si este no estuviera activado (figura D.5).

El siguiente paso es hacer *click* en "Configuración de *Bluetooth*" (figura D.6). Una vez hecho esto se abrirá una ventana como la mostrada en la figura D.7; donde se debe hacer *click* en el dispositivo que se desea conectar ("ININ"), luego de hacer esto, el proceso de conexión habrá terminado.

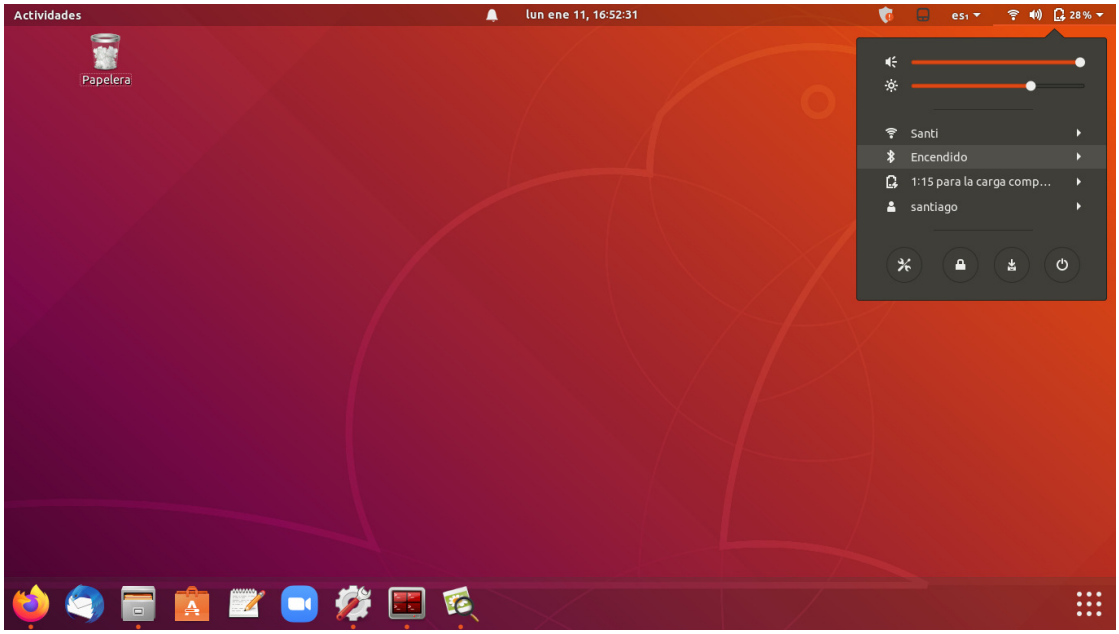

Figura D.5: Encendido de *Bluetooth* en *Ubuntu*

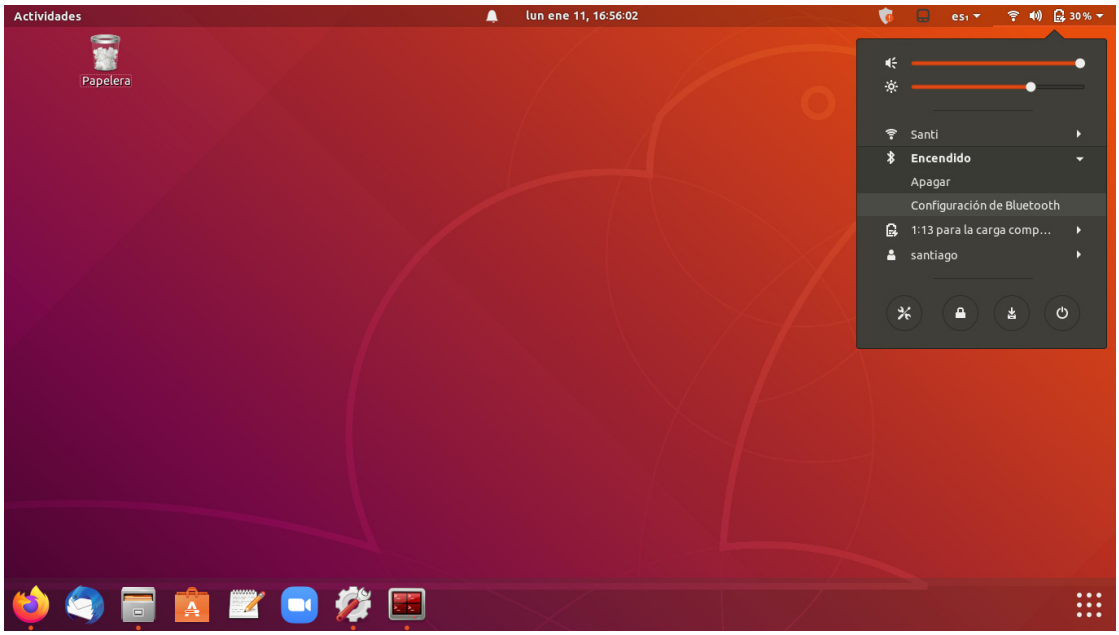

Figura D.6: Configuración de *Bluetooth* en *Ubuntu*

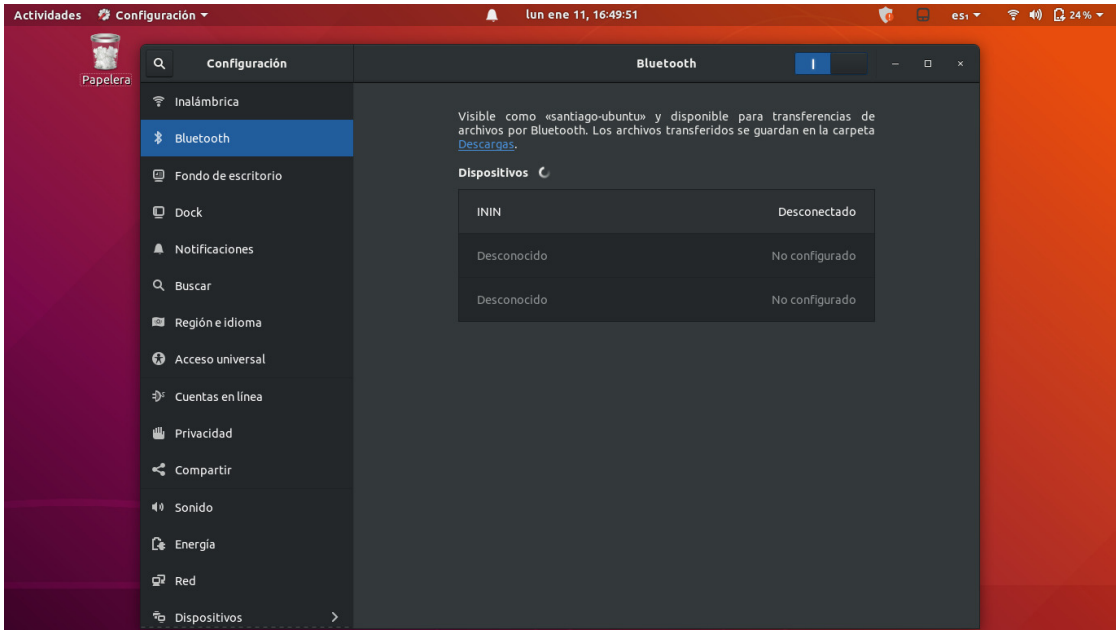

Figura D.7: Agregar dispositivo ININ en *Ubuntu*

D.4. Modos del Dispositivo

D.4.1. Selección de Modo

El dispositivo permite al usuario elegir entre cuatro modos, modo Gestos, modo Cabeceo, modo Botonera, y modo Barrido. En esta sección se dan las instrucciones para que el usuario pueda cambiar entre estos cuatro modos.

1. Encender el dispositivo según se explica en la Sección D.2.1.
2. Al encender el dispositivo por defecto se inicia en el último modo en el que fue utilizado. Los modos se pueden seleccionar en el orden siguiente: modo Barrido, modo Botonera, modo Cabeceo, y modo Gestos.
3. Para cambiar de modo presione el *encoder* (figura D.1, recuadro 2) durante tres segundos, al soltarlo verá el cursor del *mouse* moviéndose según un patrón predefinido; el patrón mostrado corresponde al modo seleccionado actualmente. Al girar el *encoder* el patrón cambiará; una vez se llegue al patrón correspondiente al modo que se desea, este se selecciona presionando el *encoder* una vez más. En la figura D.8 se presenta la correspondencia entre patrones del cursor y modos del dispositivo.

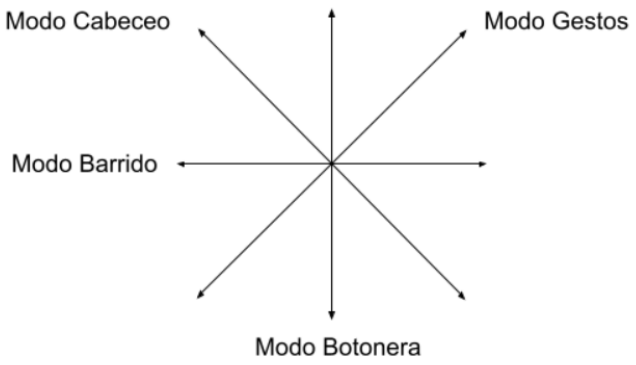

Figura D.8: Mapeo de direcciones-modos en el modo de selección

D.4.2. Configuración de Parámetros

Para cada uno de los modos se definió un parámetro para ser modificado mediante el uso del *encoder* incluido en el dispositivo. A continuación de detalla como realizar esto.

1. Seleccione el modo en el que desea utilizar el dispositivo como se explicó en la Subsección D.4.1.
2. Presionar el botón del *encoder* (figura D.1, recuadro 2) por un tiempo menor a 3 segundos.

3. Girar el *encoder* en sentido horario para aumentar el valor del parámetro, y en sentido antihorario para disminuirlo. El cambio del parámetro se verá reflejado en el funcionamiento del dispositivo inmediatamente luego de realizar un giro, por lo que puede ajustar el valor según lo considere acorde.

4. Para confirmar el cambio del parámetro, presione el botón del *encoder* para guardar la configuración.

D.4.3. Modo Barrido: Instrucciones de Uso

El Modo Barrido acciona los movimientos del cursor de forma semi autónoma y continua. Los movimientos en el eje vertical se comenzarán presionando el botón con la flecha hacia arriba (o hacia abajo), mientras que los movimientos en el eje horizontal lo harán con el botón indicado con una flecha hacia la izquierda (o derecha). También se incluye un botón para el *click* izquierdo y otro para el *click* derecho.

Para utilizar el dispositivo en modo Barrido, siga los siguientes pasos:

1. Conectar cuatro botones a la Botonera: uno con una flecha con dirección vertical (arriba o abajo), otro con una flecha con dirección horizontal (izquierda o derecha), y por último los correspondientes a *click* izquierdo y derecho. Para esto utilice el cable de audio. Cada botón tiene un lugar asignado en la Botonera, los cuales se indican utilizando auto-adhesivos con flechas o palabras. Al conectar un botón con el dispositivo los auto-adhesivos de referencia deben coincidir. En la figura D.9 se muestra la conexión de los botones con la Botonera; en rojo se marca la coincidencia que deben tener los auto-adhesivos antes nombrados.

2. Conectar la Botonera al dispositivo mediante el cable Ethernet.

3. La Botonera cuenta con un interruptor para cada botón. Mover los interruptores correspondiente a los botones que conectó hacia la posición de “PAD”.

4. Encienda el dispositivo, en caso que el mismo estuviera apagado.

5. Conectarlo mediante *Bluetooth* a una PC o con el dispositivo electrónico con el que desee usarlo. Para esto ver Sección D.3.

6. Seleccione el modo Barrido según se explica en la Sección D.4.1.

7. Para iniciar el barrido presionar el botón marcado con la flecha correspondiente. El cursor comenzará a moverse en esa dirección. Cuando llegue al borde de la pantalla cambiará de sentido.

8. Si desea frenar el cursor presione una vez el botón de dirección. No es necesario dejar el botón presionado.

9. Si desea cambiar la dirección del movimiento presione el correspondiente al otro eje.

10. Si desea hacer un *click*, presionar el botón correspondiente al *click* derecho o izquierdo según corresponda.

11. Si desea cambiar la velocidad con la que se mueve el cursor ver la Sección D.4.2.

D.4.4. Modo Botonera: Instrucciones de Uso

Para poder utilizar el dispositivo en este modo es necesario utilizar la Botonera con los seis botones conectados. A continuación, las instrucciones para utilizarlo:

1. Conectar los botones a la Botonera. Para esto conecte el botón a la Botonera usando el cable de audio. Cada botón tiene un lugar asignado en la Botonera, los cuales se indican utilizando auto-adhesivos con flechas o palabras. Al conectar un botón con el dispositivo los auto-adhesivos de referencia deben coincidir. En la figura D.9 se muestra la conexión de los botones con la Botonera; en rojo se marca la coincidencia que deben tener los auto-adhesivos antes nombrados.
2. Conectar la Botonera al dispositivo mediante el cable Ethernet.
3. La Botonera cuenta con un interruptor para cada botón. Mover los interruptores correspondiente a los botones que conectó hacia la posición de “PAD”
4. Encienda el dispositivo, en caso que el mismo estuviera apagado.
5. Conectarlo mediante *Bluetooth* a una PC o con el dispositivo electrónico con el que desee usarlo. Para esto ver Sección D.3.
6. Cambiar el dispositivo al modo Botonera. Para esto ver la Sección D.4.1
7. Cada botón realiza una acción con el cursor del *mouse*; movimiento hacia la izquierda, derecha, arriba, abajo, *click* izquierdo o derecho. Para activar los botones toque la parte metálica del mismo. Si mantiene presionado un botón el cursor se moverá continuamente o hará *click* hasta que deje de presionarlo.
8. Si desea cambiar la velocidad con la que se mueve el cursor ver la Sección D.4.2.

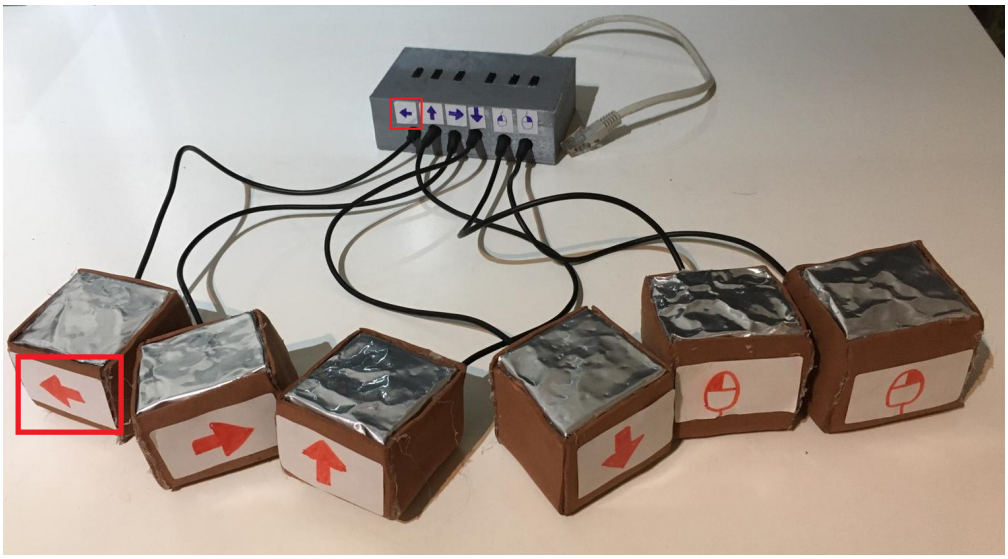

Figura D.9: Conexión de los botones con la Botonera

D.4.5. Modo Cabeceo: Instrucciones de Uso

Procedimiento para utilizar el dispositivo en modo Cabeceo:

1. Encienda el dispositivo, en caso que el mismo estuviera apagado.
2. Colocar el dispositivo en la zona u objeto deseado. Este modo está diseñado para usarse en la cabeza o en un objeto para simular un Joystick. En la figura D.10 se muestra el dispositivo correctamente sujeto a la cabeza.
3. Conectarlo mediante *Bluetooth* a una PC o con el dispositivo electrónico con el que desee usarlo. Para esto ver Sección D.3.
4. Seleccione el modo Cabeceo según se explica en la Sección D.4.1.
5. Para mover el cursor inclinar el dispositivo en el sentido y dirección deseados. En la figura D.11 se muestran las rotaciones posibles, y las acciones del cursor que activa cada una. Mientras el dispositivo siga inclinado el movimiento del cursor continuará en la dirección y sentido correspondiente.
6. Para frenar el dispositivo debe volver a la posición inicial. Si usted tiene el dispositivo sujeto a su cabeza debe mover la cabeza a su posición natural.
7. Existen dos velocidades para el movimiento del cursor, dependiendo de cuanto incline el dispositivo. Cuanto más inclinado esté más rápido se mueve el cursor.
8. Para hacer *click* es necesario rotar la cabeza en el eje correspondiente. La acción del *mouse* se genera por la velocidad del giro, no por el largo del movimiento, por lo que ante la dificultad de realizar un *click* considerar hacer movimientos más cortos y rápidos.

9. Si desea cambiar las velocidades con la que se mueve el cursor ver la Sección D.4.2.

Figura D.10: Dispositivo final colocado en la cabeza

Consideraciones a Tener en Cuenta y Situaciones a Evitar

Para realizar un doble *click* es necesario hacer una pequeña pausa luego del primer *click* a efectos de que el segundo sea tenido en cuenta.

Por otro lado, es necesario asegurarse que el dispositivo esté bien sujeto, de forma que siempre pueda volver a la posición en la que se inició el modo; si este no fuera el caso puede resultar en movimientos involuntarios del cursor o la imposibilidad de frenar un movimiento. Ante esta situación basta con seguir los siguientes pasos para solucionarlo:

1. Entrar a modo selección de modos (según lo expuesto en la Subsección D.4.1).
2. Corregir la posición del dispositivo y asegurarse que quede bien sujeto.
3. Seleccionar modo Cabeceo.

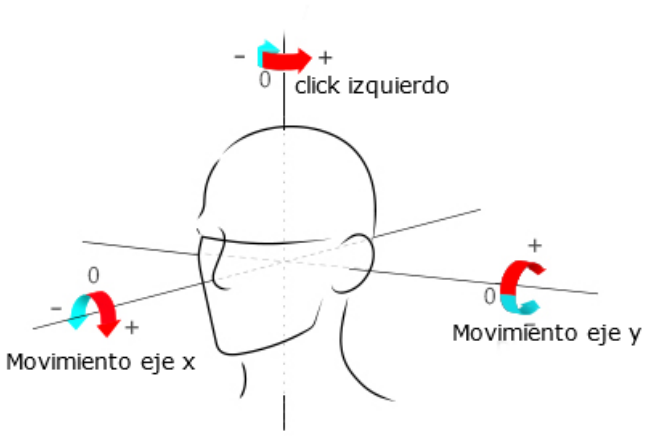

Figura D.11: Ejes principales de rotación de un cuerpo en el espacio

D.4.6. Modo Gestos: Instrucciones de Uso

Llamaremos gesto a todo movimiento paralelo al suelo que determine una acción del cursor. El gesto tendrá una dirección y sentido dados (izquierda, derecha, adelante, atrás). A continuación se listan los pasos para usar el dispositivo en modo Gestos.

1. Colóquese el dispositivo en la muñeca o en la zona del cuerpo elegida. En la figura D.12 se puede ver el dispositivo correctamente colocado en la muñeca.

2. Encienda el dispositivo, en caso que el mismo estuviera apagado.

3. Conéctelo mediante *Bluetooth* a una PC o con el dispositivo electrónico con el que desee usarlo. Para esto ver Sección D.3.

4. Seleccione el modo Gestos según se explica en la Sección D.4.1.

5. Desplace el dispositivo en el sentido y dirección (izquierda, derecha, adelante, atrás) en la que desea mover el cursor del *mouse*.

6. Para cambiar la dirección o sentido en el movimiento del cursor realice nuevamente lo detallado en el punto Nº 5.

7. Para frenar el cursor levante el dispositivo (moverlo en la dirección perpendicular al suelo).

8. Si desea realizar un *click* izquierdo, puede sacudir del dispositivo. Además esto genera que el cursor se detenga si tiene un movimiento en curso.

9. Si desea cambiar la sensibilidad de detección de los gestos ver la Sección D.4.2.

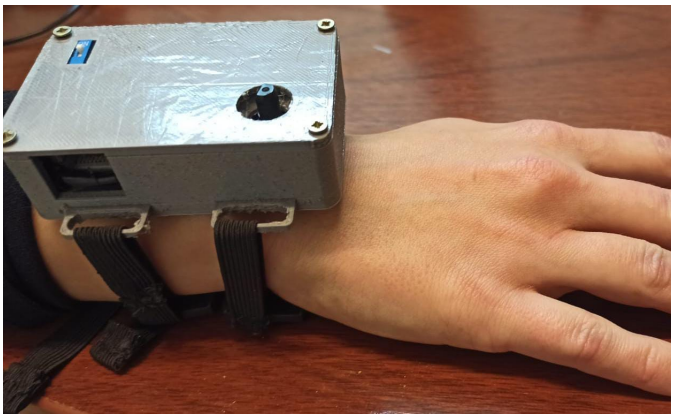

Figura D.12: Dispositivo final colocado en la muñeca

Consideraciones a Tener en Cuenta y Situaciones a Evitar

Para asegurar el correcto funcionamiento del dispositivo en este modo es necesario tener en cuenta ciertas consideraciones al realizar movimientos.

Para evitar movimientos del cursor en algún sentido distinto al deseado se debe intentar que el desplazamiento del dispositivo se realice (en lo posible) en el mismo eje en que se quiere mover el cursor.

Al realizar un gesto, es necesario considerar que para que el mismo sea identificable como tal, debe tener una velocidad mínima. Si esta velocidad no se alcanza, puede suceder que el cursor no se mueva, o incluso que se mueva en un sentido distinto al deseado. De notar este problema, aumentar la velocidad con la que se realizan gestos puede solucionarlo.

La concatenación de gestos consecutivos no está soportada; es necesario dejar el dispositivo quieto durante un tiempo entre gesto y gesto. En caso que al concatenar gestos solo se detecten correctamente los primeros, aumentar el tiempo de pausa entre gestos puede solucionar el problema (este tiempo puede crecer considerablemente al utilizar el dispositivo en el aire). Notar que en muchos de los casos de uso la pausa se da naturalmente al esperar que el cursor alcance la posición deseada.

Si se desea utilizar el dispositivo en el aire es necesario tener en cuenta consideraciones adicionales. Por un lado, evitar sacudir el dispositivo al realizar un gesto, ya que puede generar la realización de un *click*. Además, es necesario realizar los movimientos en un plano paralelo al suelo; de no tomar esta precaución, podría detectarse que se levantó el dispositivo y detener el *mouse*.

Referencias

[1] Analyze Bluetooth protocols on Windows using Wireshark. https://tewarid.github.io/2020/08/20/analyze-bluetooth-protocols-on-windows-using-wireshark.html. [Captura de Bluetooth en Windows].

[2] Adafruit. Adafruit LSM6DSOX + LIS3MDL - Precision 9 DoF IMU - STEMMA QT / Qwiic. https://media.digikey.com/pdf/Data%20Sheets/Adafruit%20PDFs/4517_Web.pdf. [Sensor MARG].

[3] Adafruit. Adafruit LSM9DS1 Accelerometer + Gyro + Magnetometer 9-DOF. https://cdn-learn.adafruit.com/downloads/pdf/adafruit-lsm9ds1-accelerometer-plus-gyro-plus-magnetometer-9-dof-breakout.pdf. [Sensor MARG].

[4] Adafruit. Magnetometer Calibration. https://learn.adafruit.com/adafruit-sensorlab-magnetometer-calibration?view=all. [Calibración del magnetómetro].

[5] Adafruit. Why Calibrate? https://learn.adafruit.com/calibrating-sensors/why-calibrate. [Calibración de sensores].

[6] AIP. Comparison of complementary and Kalman filter based data fusion for attitude heading reference system. https://aip.scitation.org/doi/pdf/10.1063/1.5018520. [Utilización de acelerómetro, giroscopio y magnetómetro para la obtención de la orientación].

[7] BCD. Datasheet Regulador de voltaje AP2112. https://www.diodes.com/assets/Datasheets/AP2112.pdf. [Regulador de voltaje lineal].

[8] BJOY. BJOY Chin. https://bjadaptaciones.com/ratones-bjoy/211-bjoy-chin.html. [BJOY Chin].

[9] BJOY. BJOY Ring Wireless. https://bjadaptaciones.com/ratones-bjoy/241-bjoy-ring-wireless.html. [BJOY Ring Wireless].

[10] Bluetooth. Understanding Bluetooth Range. https://www.bluetooth.com/learn-about-bluetooth/key-attributes/range/#estimator. [Calculadora de alcance Bluetooth].

[11] Chionotech. Accelerometer Calibration II: Simple Methods. https://chionophilous.wordpress.com/2011/08/26/accelerometer-calibration-ii-simple-methods/. [Calibración de seis puntos para el acelerometro].

[12] Compusult. 5-switch. https://compusult-limited.myshopify.com/products/5-switch. [Dispositivo para controlar el cursor del mouse con botones].

[13] Compusult. Jouse3. https://www.compusult.com/assistive-technology/our-at-products/jouse3. [Jouse 3].

[14] Conec. Datasheet DCJ0202. https://uy.mouser.com/datasheet/2/81/C0504R06A-1160352.pdf. [Conector de audio de 3.5mm].

[15] RJ CooperAssociates. SAM-Joystick. https://store.rjcooper.com/products/sam-joystick?variant=47792320079. [SAM-Joystick].

[16] Tobii Dynavox. The New PCEye. https://www.tobiidynavox.com/devices/eye-gaze-devices/pceye-5-31ad2875/#Specifications. [Tobii The New PCEye].

[17] TT Electronics. Rotary Encoder. https://www.ttelectronics.com/TTElectronics/media/ProductFiles/Encoders/Datasheets/EN11.pdf. [Datasheet Encoder Rotatorio EN11].

[18] Eneso. 5-enPathia. https://www.eneso.es/shop/product/enpathia. [Dispositivo para controlar el cursor del mouse con la cabeza].

[19] Freecad. Programa de diseño Freecad. https://www.freecadweb.org/. [Programa de diseño Freecad].

[20] Texas Instruments. Datasheet nRf52832 con antena. https://www.ti.com/lit/ds/swrs037b/swrs037b.pdf?ts=1625009433848&ref_url=https%253A%252F%252Fwww.google.com%252F. [Transmisor RF].

[21] Maxim integrated. Datasheet MAX4400. https://datasheets.maximintegrated.com/en/ds/MAX44000.pdf. [Sensor óptico MAX44000].

[22] IntelliGaze. IntelliGaze. https://www.intelligaze.com/en/IntelliGaze/overview. [IntelliGaze].

[23] InvenSense. MPU-9250 Product Specification Revision 1.1. https://media.digikey.com/pdf/Data%20Sheets/Seeed%20Technology/Grove_IMU_9DOF_v2.0_Web.pdf. [Sensor MARG].

[24] Microchip. AT42QT1011 Data Sheet. https://ww1.microchip.com/downloads/en/DeviceDoc/40001947A.pdf. [Sensor Capacitivo].

[25] Microchip. Bluetooth Low Energy Connection Process. https://microchipdeveloper.com/wireless:ble-link-layer-connections. [Restricción sobre intervalo mínimo de conexión].

[26] Microchip. Datasheet MCP73831. https://ww1.microchip.com/downloads/en/DeviceDoc/MCP73831-Family-Data-Sheet-DS20001984H.pdf. [Datasheet Circuito de carga].

[27] Mouser. BMX055 Shuttle Board. https://www.mouser.com/datasheet/2/783/BST-DHW-FL021-1509587.pdf. [Sensor MARG].

[28] AkeemWhitehead MubinaToa. UltrasonicSensingBasics. https://www.ti.com/lit/an/slaa907c/slaa907c.pdf?ts=1625080043098&ref_url=https%253A%252F%252Fwww.google.com%252F. [Documento de TI sobre sensado básico con ultrasonido].

[29] Nordic. Datasheet ARM Cortex-M4. https://infocenter.nordicsemi.com/pdf/nRF52832_PS_v1.4.pdf. [Datasheet ARM Cortex-M4].

[30] Nordic. Datasheet nRf52832. https://cdn-learn.adafruit.com/downloads/pdf/bluefruit-nrf52-feather-learning-guide.pdf. [Microcontrolador].

[31] NXP. NXPSensor Fusion Library for KinetisMCUs. https://www.nxp.com/docs/en/data-sheet/NSFK_DS.pdf. [Algoritmo de fusión de sensores].

[32] Network of Care. Integramouse. https://howard.md.networkofcare.org/veterans/assistive/detail.aspx?id=7625. [Dispositivo Integramouse].

[33] Rajendra Pawar, Sunil Tekale, Suresh Shisodia, Jalinder Totre, and Abraham Domb. Biomedical applications of poly(lactic acid). *Recent Patents on Regenerative Medicine*, 4, 05 2014.

[34] QUHA. Quha zono. https://www.quha.com/products-2/zono/. [Dispositivo para controlar el cursor del mouse con la cabeza].

[35] Raytac. Datasheet MDBT42Q. https://www.raytac.com/upload/download_files/38a8a4a0aff945d8484507d60058109b.pdf. [Datasheet Módulo Bluetooth].

[36] Segger. J-Link / J-TraceUser Guide. https://www.segger.com/downloads/jlink/UM08001. [J-Link].

[37] Rohm Semiconductor. Basics of linear regulators. https://www.mouser.com/pdfdocs/linear_reg_basic_appli-e.pdf. [Eficiencia de un regulador lineal].

[38] Nordic Semiconductors. Online Power Profiler for BLE. https://devzone.nordicsemi.com/nordic/power/w/opp/2/online-power-profiler-for-ble. [Calculadora de consumo antena Bluetooth].

[39] Robert Langer Shady Farah, Daniel G. Anderson. Physical and mechanical properties of PLA, and their functions in widespread application. https://dspace.mit.edu/bitstream/handle/1721.1/112940/Anderson_Physical%20and%20mechanical%20properties.pdf?sequence=1&isAllowed=y. [Estrcutra del PLA].

[40] Sparkfun. 9 Degrees of Freedom - Razor IMU. https://www.sparkfun.com/products/retired/10736. [Sensor MARG].

[41] Sparkfun. nRF52832 - Product Specification. https://cdn.sparkfun.com/datasheets/Wireless/Bluetooth/nRF52832_PS_v1.0.pdf. [Datasheet nRF5282].

[42] SparkFun. SparkFun Capacitive Touch Breakout - AT42QT1011. https://media.digikey.com/pdf/Data%20Sheets/Sparkfun%20PDFs/SEN-14520_Web.pdf. [Placa de desarrollo del Sensor Capacitivo].

[43] STMicroelectronics. Datasheet LIS3MDL. https://www.st.com/resource/en/datasheet/lis3mdl.pdf. [Sensor Magnetómetro].

[44] STMicroelectronics. Datasheet LSM6DSOX. https://www.st.com/resource/en/datasheet/lsm6dsox.pdf. [Sensor IMU].

[45] STMicroelectronics. Datasheet M95320. https://www.st.com/resource/en/datasheet/m95320-a125.pdf. [EEPROM].

[46] Yutaka Tokiwa and Buenaventurada Calabia. Biodegradability and biodegradation of poly(lactide). *Applied microbiology and biotechnology*, 72:244–51, 10 2006.

[47] Ana Valerga, Moises Batista Ponce, Jorge Salguero, and Franck Girot. Influence of pla filament conditions on characteristics of fdm parts. *Materials*, 11, 07 2018.

[48] Wireshark. Bluetooth capture setup. https://gitlab.com/wireshark/wireshark/-/wikis/CaptureSetup/Bluetooth. [Captura de Bluetooth en Linux].

[49] Wireshark. Programa Wireshark. https://www.wireshark.org/. [Programa analizador de protocolos].

[50] Persing Junior Cárdenas Vivanco y Ettore Apolonio de Barros. Filtro de Kalman extendido aplicado en la navegación de un AUV. https://www.redalyc.org/pdf/5055/505554819003.pdf. [Aplicación de Filtro de Kalman para determinar orientación con un IMU].

Índice de tablas

Esta página ha sido intencionalmente dejada en blanco.

Índice de figuras

Esta es la última página.
Compilado el lunes 23 agosto, 2021.
http://iie.fing.edu.uy/