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

DISEÑO Y CONSTRUCCIÓN DE UN SISTEMA PARA DETECTAR, LOCALIZAR Y CARACTERIZAR ACCIDENTES AUTOMOVILÍSTICOS.

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO

MATÍAS IGNACIO SILVA CARES

PROFESOR GUÍA:
ANÍBAL MADRID CARRASCO

MIEMBROS DE LA COMISIÓN:
SANDRA CÉSPEDES UMAÑA
ANDRÉS CABA RUTTE

SANTIAGO DE CHILE
2017

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: MATÍAS IGNACIO SILVA CARES
FECHA: 2017
PROF. GUÍA: SR. ANÍBAL MADRID CARRASCO

DISEÑO Y CONSTRUCCIÓN DE UN SISTEMA PARA DETECTAR, LOCALIZAR Y CARACTERIZAR ACCIDENTES AUTOMOVILÍSTICOS.

Los accidentes automovilísticos son un grave problema mundial sin solución. La gran cantidad de heridos y muertes como resultado de choques en carretera o ciudad ha ido en aumento año tras año debido a la gran cantidad de viajes que ocurren en todo el mundo. Las estadísticas dicen que el 25 % de la gente involucrada en accidentes podría haberse salvado con una asistencia médica inmediata, es por eso que la empresa Sosmart Labs desarrolló una aplicación para detectar choques y notificar inmediatamente la ubicación y tiempo del suceso a cercanos del pasajero.

Esta memoria se enmarca en el proyecto Sosmart Premium de la empresa, que consiste en diseñar y contruir un dispositivo electrónico que se ancle al vehículo y que realice la misma tarea de la aplicación Sosmart, además de implementar el sistema de comunicación y un proceso de post-procesamiento para caracterizar el accidente.

El trabajo consistió en el diseño e implementación de un prototipo de dispositivo electrónico a través de la elección de componentes de mercado y fácil uso, en la que se optó por Arduino como controlador, un MPU9250 como sensor de aceleraciones y un FONA 2G como módulo para comunicaciones. Luego, se desarrolló la arquitectura completa implementando una plataforma web para recibir las notificaciones y también un programa de post-procesamiento para limpiar las señales y usando un programa de simulación 3D, recrear el accidente.

El dispositivo electrónico posee un algoritmo de detección de choque autónomo que fue diseñado utilizando una base de datos de accidentes de tráfico de Estados Unidos y con ella, se desmostró que las magnitudes de aceleraciones en un choque real, poseen un valor alto que es fácil de detectar utilizando un acelerómetro de alto rango. En este caso, el MPU9250 demostró ser lo suficientemente útil para utilizar el algoritmo y discernir eficazmente de choques reales con falsos positivos. Por otro lado, el proceso notificación funcionó para los 3 canales propuestos: SMS, llamada y POST request a una plataforma web de monitoreo. Finalmente, el post-procesamiento consistió en la aplicación de filtros y zonas de histéresis para atenuar el ruido producto del sensor y con el uso de integraciones, obtener los desplazamientos y ángulos del sensor para simularlos en Unity3D.

Concluyendo, se desarrolló un sistema de detección, notificación y simulación de accidentes, en los que el sistema de detección obtuvo buenos resultados para accidentes de gran magnitud y se aislaron casos de falsos positivos, se creó la plataforma web para monitorear accidentes y en cuanto a la caracterización, el sistema desarrollado permite obtener una idea de la orientación del vehículo previo al accidente, pero no fue posible determinar la trayectoria real de este.

Agradecimientos

Ha sido un largo y duro camino, no podía dejar fuera de este trabajo a tanta gente que me brindó su apoyo y ayuda en diversos momentos. La más importante ha sido mi madre que me ayudó en todo lo que estuvo a su alcanza e incluso más para que yo me centrara y rindiera en los estudios, este título será principalmente para ella y su esfuerzo para salir adelante conmigo. También agradecer a los grandes amigos que hice durante estos años en la Universidad y agradecer tanto su riqueza intelectual como humanitaria, sin ellos todo hubiese sido mucho más difícil.

Por otro lado agradezco la amistad y apoyo en momentos difíciles a mis amigos y amigas, así como también guardo buenos recuerdos de aquellos que brindaron su apoyo y por diversos motivos ya no seguimos en contacto, pero sin duda fueron parte del camino también.

Agradecer también personalmente a Anibal Madrid y su empresa Sosmart Labs por la oportunidad de trabajar en esta memoria y aportar en los proyectos de la empresa. Sin duda ha sido un gran experiencia laboral, de mucho aprendizaje y de la que me llevo los mejores recuerdos.

Finalmente, agradecer a muchos profesores y profesionales que me enseñaron valiosas herramientas durante el camino y me hacen sentir seguro para enfrentar esta segunda parte de la vida con mucho entusiasmo.

A cada uno de ellos les mando un abrazo y de todo corazón, siento que este trabajo es reflejo de todo su apoyo y cariño a través de los años.

Tabla de Contenido

Capítulo 1

Introducción

1.1. Motivación

Los accidentes automovilísticos son un grave problema mundial sin solución. La gran cantidad de heridos y muertes como resultado de choques en carretera o ciudad ha ido en aumento año tras año debido a la gran cantidad de viajes que ocurren en todo el mundo.

Sólo en EE.UU ocurren casi 37,000 accidentes automovilísticos al año que se traducen en una muerte cada 100 millones de millas viajados, según datos del año 2015 [1]. A nivel global, las estadísticas señalan que cada día mueren más de 3000 personas, entre 15 y 44 años mayormente, por accidentes de tránsito, siendo la segunda mayor causa de muerte en el mundo de personas en este rango de edad. Además, alrededor de 20-30 millones de personas resultan heridas anualmente o quedan con secuelas de discapacidad y económicamente, los choques también tienen un costo considerable, alcanzando los casi 500 billones de dólares anuales [2].

Las principal causa de accidentes de tráfico, está en el consumo de alcohol pero la mayor causa de muerte en accidentes se debe a la ausencia de ayuda médica inmediata al pasajero o conductor del automóvil involucrado en el choque. En la mayoría de los casos es debido a que la información del choque demora mucho en llegar a los hospitales y por ende, las ambulancias demoran en prestar la ayuda necesaria. Se dice que cada minuto que pasa sin que la víctima reciba ayuda médica puede marcar una gran diferencia en su probabilidad de sobrevivencia, análisis muestran que uno de cada cuatro personas pudo haberse salvado si se hubiese prestado atención médica de emergencia a tiempo, es decir, 350,000 vidas anuales [3].

La situación en Chile es aún más preocupante considerando que posee una población y un parque vehicular mucho menor. Las estadísticas indican que para el año 2014, ocurrieron alrededor de 80000 accidentes de tránsito de los cuales resultaron fallecidas casi 1600 persones, es decir, 2 % de los accidentes fueron fatales y que facilitando asistencia médica rápidamente, alrededor de 500 hubiesen sobrevivido al accidente [4].

Se han ido desarrollando soluciones para disminuir este tiempo de respuesta en el rescate de personas involucradas en accidentes de tránsito. En la actualidad, existen varias alternativas en el mundo, algunas como una tecnología de uso público y otras a menor escala privada.

En este último punto, se encuentra la empresa Sosmart Labs que creó una aplicación para smartphones que detecta y alerta de choques automovilísticos y que además, pretende diseñar y construir un dispositivo electrónico que se conecte fácilmente a un automóvil y cumpla la misma función.

1.2. Objetivos

1.2.1. Objetivo general

Implementar un sistema para detectar, localizar y comunicar un accidente automovilístico a centrales de emergencia a través del diseño y construcción de un dispositivo electrónico embebido y el uso de un algoritmo adaptado.

1.2.2. Objetivos específicos

- Implementar un algoritmo de detección de desaceleraciones.
- Diseñar y construir un dispositivo electrónico embebido utilizando componentes de mercado y basado en la plataforma Arduino.
- Implementar el software que permita detectar el accidente, encontrar su localización e inmediatamente notificar.
- Simular el tipo de accidente con el uso de un software de modelación 3D.
- Validar el sistema mediante pruebas simuladas y reales en un automóvil.

1.3. Estructura de la memoria

El segundo capítulo busca contextualizar el proyecto y por ello se mencionan conceptos técnicos necesarios para entender las partes que siguen en el documento, la descripción de la empresa, la plataforma de trabajo a utilizar y el estado del arte de proyectos similares. El tercer capítulo habla sobre la metodología a utilizar. El cuarto detalla el diseño hecho para construir le sistema. El quinto la implementación de dicho diseño a través de código. El sexto muestra los resultados de la implementación y la validación del sistema. Finalmente en el séptimo capítulo se exponen las conclusiones de este proyecto y trabajo a futuro.

Capítulo 2

Marco teórico y estado del arte

En este capítulo se pretende contextualizar al lector en torno a las temáticas que involucran este trabajo. En la primera parte se mencionan algunos conceptos ténicos necesarios para entender gran parte de este documento en las fases de diseño e implementación. Una segunda parte, para describir la empresa solicitante del proyecto y la plataforma de trabajo que se utilizará en ella. Finalmente, el estado del arte en cuanto a soluciones para detección de accidentes y sistemas de atención de emergencias, tanto a nivel comercial como académico.

2.1. Conceptos técnicos

2.1.1. Física en un choque de cuerpos

Un choque se define como la colisión entre dos o más cuerpos en el cual el impactado está en reposo, pudiendo resultar en un choque elástico, donde hay conservación de energía cinética o inelástico, donde ocurren pérdidas de energía cinética a través de deformaciones o aumentos de temperatura de los cuerpos.

EL factor clave en un choque físico o mecánico es la aceleración o desaceleración repentina que experimenta un cuerpo. Es por esto que los choques suelen medirse con un acelerómetro, que toma unidades de metros por segundo al cuadrado pero en la práctica y por conveniencia, se miden como un múltiplo de la aceleración de gravedad **g**, que tiene un valor de $9,89665[\frac{m}{s^2}]$ a nivel de mar. Esta es la aceleración experimentada por todos los cuerpos sobre la superficie de la Tierra a nivel de mar y además, en términos más prácticos, $1g$ equivale a la variación de velocidad de $35[\frac{km}{h}]$ en cada segundo. Por lo tanto, un automóvil que viaje a $105[\frac{km}{h}]$ y frene en un segundo experimentará una aceleración de $3g$, en cambio experimentar un choque a la misma velocidad pero en $200ms$, es equivalente a casi $15g$.

Los choques pueden caracterizarse por la aceleración máxima experimentada, como también por la duración y forma del pulso de choque. En el caso de choques automovilísticos, la mayoría de los casos sólo importa el peak de aceleración/desaceleración experimentada (en

valores de g) para detectar el choque y realizar acciones rápidas para proteger al conductor, como es el caso de los air-bags.

Los air-bags son una especie de saco que se infla de forma automática en menos de 30 milisegundos al chocar el vehículo frontalmente contra un obstáculo. Este sistema de seguridad protege a los pasajeros minimizando el riesgo de sufrir fracturas de tórax y nariz causadas por el impacto contra el volante. La activación de este saco inflable está controlada por un sistema electrónico interno del vehículo que mide la desaceleración y procesa los datos para detectar cuándo se experimenta un choque o una frenada brusca que ponga en peligro a los pasajeros.

2.1.2. Sistemas de referencia: posición y rotación

Los sistemas de referencia son convenciones usadas por un observador para medir algunas magnitudes físicas de un sistema, como por ejemplo posición, traslación, rotación, aceleración, etc.

En el caso de la posición y rotación, sus valores numéricos son relativos a un sistema de referencia u origen que se considere, es por esto que se dice que el movimiento es relativo, pues siempre dependerá desde qué posición se está observando la medición.

Cualquier sistema de coordenadas para determinar posición en el espacio consiste en 3 ejes: X,Y,Z y que cubren los 3 planos del espacio. De esta forma, un punto queda determinado por este trío de coordenadas. Sin embargo, para determinar un sistema de coordenadas dentro de otro sistema de coordenadas, es necesario considerar los ángulos entre los ejes de ambos sistemas, para así determinar la posición y rotación que posee el segundo respecto al primero, estos ángulos se denominan Ángulos de Euler y son **roll**, **pitch** y **yaw** para los ejes X,Y y Z, respectivamente.

En la Figura 2.1 se muestra un sistema base xyz fijo y otro XYZ móvil que comparten origen, pero donde el segundo queda determinado a partir de los ángulos de Euler.

2.1.3. Procesamiento de señales

Una señal en el ámbito de la ingeniería es la medición de algún fenómeno físico o variable determinada. Existen señales de tipo analógica, que se presentan generalmente en la naturaleza (sonido, luz, energía, etc) y se representan matemáticamente por una función continua, como también de tipo digital, que son aquellas que sólo manejan valores discretos a través de una codificación de otra señal analógica.

Las señales son muy importantes pues almacenan información de la variable medida. Como por ejemplo, la señal del electrocardiograma que revela información de los latidos del corazón. Señales de voltaje que almacenan la amplitud y fase de la energía. Señales de audio que almacenan el sonido y es capaz de reproducirlo nuevamente, etc.

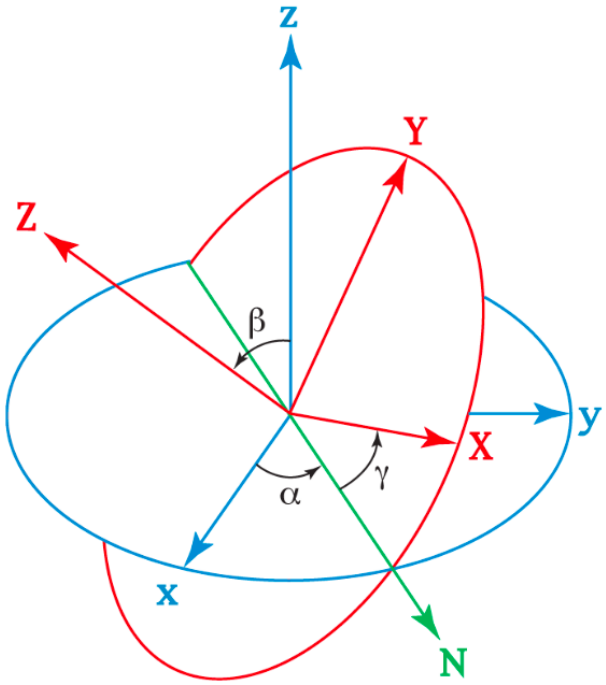


Figura 2.1: Sistema de coordenadas xyz fijo y XYZ móvil determinado por los ángulos de Euler: roll, pitch, yaw. Fuente: [https://commons.wikimedia.org/wiki/Euler_angles]

Sin embargo, muchas veces, las señales miden variables físicas con ruido debido a la imprecisión o calidad de los sensores, que no permite un buen estudio de la información. Es por esto, que existe un área de la Ingeniería eléctrica ligada al procesamiento de esta información, en la cual se estudian y desarrollan técnicas para procesar una señal y poder dismiuir el ruido asociado, extraer más características para mejorar la interpretación de la señal o modificar la información contenida de la misma.

Una de las técnicas más utilizadas para disminuir el ruido asociado de una señal es el uso de filtros en la señal, que en simples términos, permite limitar los valores de la señal y que se comporten de una forma suave. También se utilizan filtros para extraer ciertas características de las señales, como por ejemplo, detectores de bordes en señales bidimensionales o imágenes.

Los filtros pasabajos, son aquellos que intentan extraer el comportamiento ruidoso de una señal a través de la eliminación de frecuencias altas en la señal. De esta forma, una señal con valores que varíen mucho, se comportará más suave después de aplicar este tipo de filtros, como se muestra en la Figura 2.2.

Otro tipo de filtro, es el de Butterworth, que según su diseño puede actuar como pasa bajo, pasa banda o pasa alto, pero lo más importante es que definiendo una frecuencia de corte, la salida de la señal es lo más plana posible, dismuyendo las ondulaciones en la señal.

Por otro lado, existe otro tipo de ruido en las señales que se produce por errores netos del sensor o de la lectura de estos datos, los denominados **Outliers**. Los Outliers son valores particulares de la señal que en amplitud escapan notoriamente de la media anterior y posterior a este valor en la señal. Por lo que, es muy probable que no sean representativos de lo que se

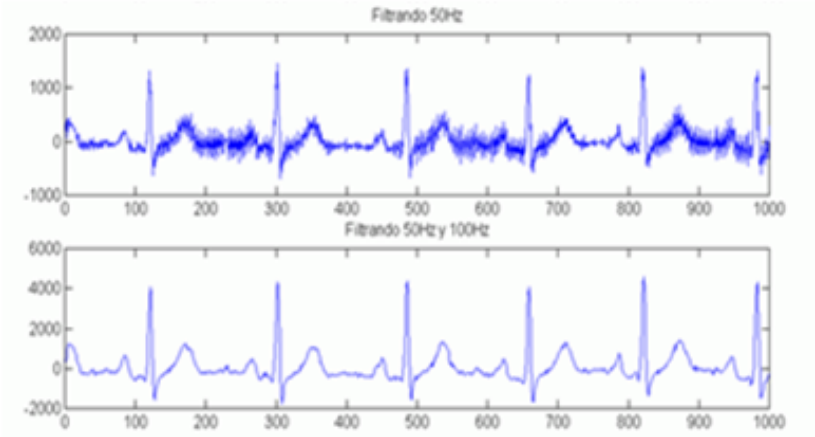


Figura 2.2: Procesamiento de una señal biomédica a través del uso de un filtro pasa bajos.

está midiendo y tomarlos en cuenta en un procesamiento puede influir en errores.

Existen varias formas de eliminarlos, desde algoritmos que utilicen la desviación estándar para filtrar y filtros pasa bajo adaptados. Como también otra opción es no considerarlos en el procesamiento para evitar falsos positivos, como por ejemplo, en un algoritmo de detección.

2.1.4. Histéresis

Esta palabra tiene muchos significados dependiendo del área en que se utilice, como física, mecánica, economía o lógica digital.

En esta última área, específicamente en el estudio de señales creadas por circuitos eléctricos, ocurre que a veces las señales toman tiempo para transicionar de un nivel a otro. Dependiendo del diseño del circuito y la variable a medir, estas señales pueden sobrepasarse, demorar en transicionar, oscilar o ser afectadas por ruido eléctrico.

Los circuitos digitales normalmente usan un umbral para decidir si una señal es alta o baja respecto a ese umbral. Si la amplitud de la señal de entrada es mayor a este umbral (llamada threshold en inglés), la señal de salida es 1. Si es menor, la señal de salida es 0. Sin embargo, cuando una señal de entrada es afectada por oscilaciones o ruido eléctrico, esta misma puede cruzar este umbral múltiples veces y la señal de salida termina mostrando varias transiciones entre altos y bajos.

Para flexibilizar estas transiciones, existe la histéresis, que es una región creada alrededor de un umbral lógico en la cual se crean 2 barreras separadas (arriba y abajo del umbral original) para cambiar de alto a bajo, y viceversa. Es decir, si la señal supera el umbral mayor, se cambia su valor a 1 pero para volver a cambiar a 0, debe ser menor al umbral menor. De esta forma, las oscilaciones de la señal cerca del umbral ya no generan que la salida cambie continuamente. En la figura 2.3 se muestra un esquema de esta situación.

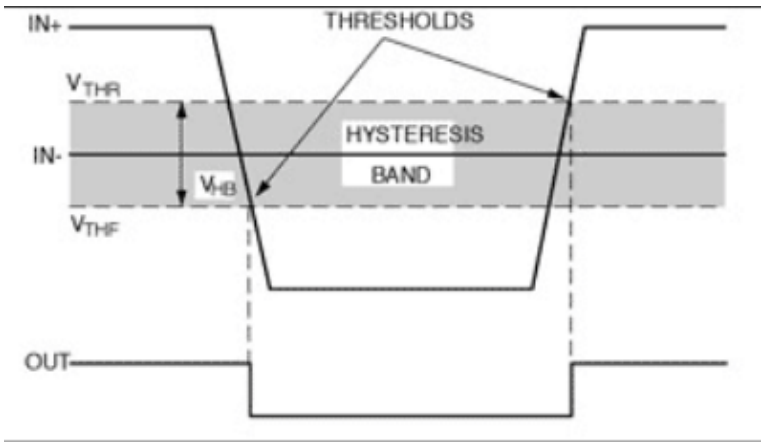


Figura 2.3: Región de histéresis en una señal digital. Fuente: [https://electronics.stackexchange.com/questions/12312/what-is-input-hysteresis]

2.2. Descripción de la empresa y plataforma de trabajo

2.2.1. Empresa Sosmart Labs

Sosmart Labs es una empresa chilena ligada al ámbito del desarrollo e innovación de productos tecnológicos. Fue fundada por el ingeniero eléctrico Anibal Madrid el año 2014 y cuenta con oficinas en Providencia.

Hace 2 años creó la aplicación SOSmart para detectar choques automovilísticos y enviar notificaciones a contactos predefinidos. De esta forma, poder disminuir los tiempos de rescate de ambulancias en este tipo de accidentes.

Gracias a SOSmart, Sosmart Labs se adjudicó diversos fondos de capital semilla (Corfo, Santiago Innova, Everis, entre otros más pequeños) y fue catalogada dentro de los 100 mejores emprendimientos innovadores a nivel mundial [5]

En la actualidad, la empresa trabaja en diversos proyectos tecnológicos siendo el de mayor impacto Momo, el reloj GPS-celular para niños [6], que fue lanzado al mercado en diciembre de 2016. Por otro lado, abrió una nueva área de VR/AR (realidad virtual y realidad aumentada) para crear productos innovadores e impactar el mercado mundial con ellos.

Aplicación SOSmart

SOSmart es una aplicación para celulares que detecta un accidente automáticamente utilizando los sensores internos del Smartphone, y de manera inmediata envía una notificación de emergencia con la ubicación a contactos de emergencia, seleccionados previamente. Así se reduce notoriamente el tiempo en que los servicios de urgencia son notificados del accidente.[7].

Algunas de sus características son:

- El servicio de detección de accidentes puede ser activado manualmente o activarse automáticamente, esto es que la aplicación detecte que el usuario está en un vehículo en movimiento. Siendo una característica de gran utilidad para aquellos que se desplazan en vehículo frecuentemente.
- Aquellos contactos que tengan la aplicación instalada, podrán ser notificados mediante una alarma ruidosa. De esta forma, podrán ser notificados y asistirte lo antes posible sin importar lo que estén haciendo en el momento.
- Posee un botón de pánico para alertar manualmente de algún tipo de emergencia y ubicación a los contactos previamente elegidos.
- El algoritmo de detección fue desarrollado utilizando datos de choques reales entregados por la *National Highway Traffic Safety Asocitation* de Estados Unidos para detectar accidentes severos. De esta manera la aplicación es capaz de aislar los casos cuando el Smartphone se cae, algunas frenadas bruscas o choques menores sin alertar innecesariamente [8]
- SOSmart también ofrece una lista de hospitales cercanos y la forma de llegar desde un determinado lugar. Todo para recibir la atención médica necesaria lo más rápido posible.

El diseño e implementación es completamente nacional y está disponible para plataformas Android e IOS de manera gratuita en todo el mundo. Por otro lado, también ofrece una plataforma web de monitoreo, que permite a todo tipo de instituciones monitorear en tiempo real emergencias activadas por sus empleados y/o afiliados. Este servicio si bien es pagado, es ampliamente usado por empresas de transporte, seguridad y hospitales.

Desde su lanzamiento en 2014 (ver figura 2.4), SOSmart ha salido en diversos medios de prensa en Chile y blogs del mundo[9], [10], [11], ganando algunos fondos de capital semilla de Corfo, Google Developers y Open Beauchef. Posee más de 30.000 descargas a nivel mundial y a la fecha lleva un registro de casi 2 millones de viajes.

Además ha establecido alianzas con empresas de asistencia para accidentes automovilísticos como Auxilia[12], SAMU Brasil [13] y está en conversaciones con empresas de seguro de autos y flota de vehículos de transporte como Cabify y Uber.

Figura 2.4: Logo de aplicación SOSmart en su lanzamiento (año 2014). Fuente: [http://www.sosmartapp.com]

Producto SOSmart Premium

Si bien la aplicación SOSmart ha tenido muy buena crítica de los usuarios por su utilidad y eficiencia (posee un rating 4.6/5 en la AppStore). Muchas empresas les pesa el hecho de

que sea necesario un celular para su uso, puesto que es un aparato más de la persona que del propio auto, lo que genera discrepancias en algunas empresas de seguro o que poseen una flota de vehículos que necesitan monitorear.

Algunos emprendimientos han intentado integrar sensores y GPS a un automóvil, pero el problema es que para ello se debe intervenir eléctricamente el automóvil por lo que se pierden los seguros contra accidentes y además resultan ser muy costosos de implementar en flotas grandes de vehículos.

El sistema ideal, sería integrar de alguna forma sencilla un dispositivo que realizara la función de SOSmart, en lo posible con sensores más precisos y que fuera más característico del automóvil mismo que del usuario.

La empresa estadounidense Splitsecnd [14] diseñó un dispositivo que se conecta a la cigarrera de los autos y pueda detectar accidentes y establecer una comunicación con una central de ayuda médica para que acudan al lugar a socorrer a los pasajeros. Sin embargo, este producto no se está comercializando en América, sólo en algunos países de Europa.

En Chile y América Latina no existen dispositivos de este estilo en el mercado por lo que Sosmart Labs pretende crear su propio dispositivo siguiendo el diseño de Splitsecnd y convencer a las empresas del rubro de seguros y atenciones médicas de instalar estos dispositivos en sus vehículos. Para detectar oportunamente accidentes como para estudiar la forma de conducir de sus conductores y segmentar de mejor forma aquellos que manejen peor y tengan mayor probabilidad de sufrir un accidente. En la Figura 2.5 se muestra parte de la imagen corporativa con la que comenzó a negociar con algunas empresas, como Clínica Las Condes y Unidad Coronaria Móvil [15]

Además, Sosmart Premium pretende funcionar como un caja negra, es decir guardar los datos de los sensores en una SD para luego ser obtenidos y reconstruir el accidente determinando la causa inmediatamente y ahorrando millones de dolares a las empresas.

Figura 2.5: Afiche corporativo de Sosmart Premium, el dispositivo para detectar accidentes en vehículos.

2.2.2. Hardware

En esta sección se describen diferentes componentes que integrados forman el dispositivo SOSmart Premium.

La parte electrónica consiste de un micro-controlador, acelerómetro, giróscopo, botón análogo, micrófono, parlante, batería y FONA 3G de comunicación con antena GSM y GPS. Por otro lado, la parte de diseño industrial para crear la carcaza que se conectará a la cigarrera.

Arduino

Existen muchos tipos de controladores en el mercado, pero pocos se enfocan en ser fáciles de usar o programar y con muy buena documentación liberada como los Arduinos.

Arduino es una compañía de hardware libre (posee librerías para ejecutar muchas cosas) y una comunidad tecnológica que diseña y crea placas computadora de desarrollo de hardware compuesta por circuitos impresos que integran un microcontrolador y además, desarrolla software a través de un entorno de desarrollo (IDE), en donde se programa cada placa. [16]

Existen muchos tipos de Arduino, siendo el más básico de todos el Arduino UNO (ver Figura 2.6), el cual posee múltiples pines de entrada y salida para señal analógica y también digital. Además de pines especiales para comunicación serial (Tx,Rx), pwm, I2c, salida de audio, alimentación USB, entre otros.

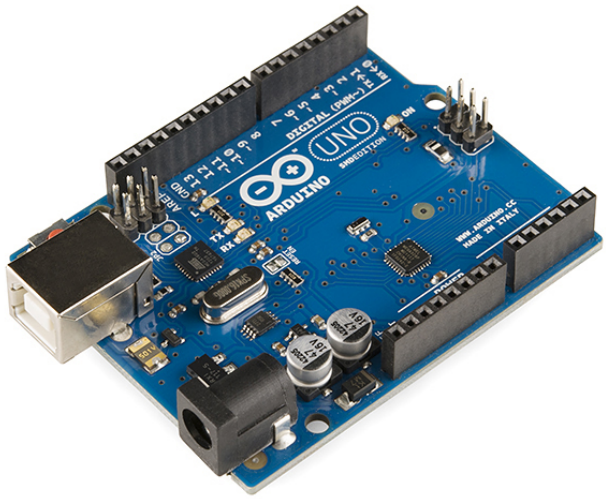


Figura 2.6: Placa Arduino UNO. Fuente: [https://en.wikipedia.org/wiki/Arduino]

Este tipo de Arduino es muy sencillo de usar y posee una capacidad suficiente para ejecutar múltiples tareas complejas, por lo que se considera uno de los controladores más recomendados para aquellos que están empezando a programar controladores. Sin embargo, su gran tamaño le impide formar parte de dispositivos comerciales por lo que sólo se usa como kit de prueba o ejercitación para estudiantes de Arduino.

El lenguaje de programación está basado en C y soporta todas las funciones del estándar C y algunas de C++. El IDE es propio de Arduino, el cual compila y carga el código en las placas ofreciendo una consola para leer y escribir datos con el programa en tiempo real. También puede integrar al código librerías propias de Arduino o externas.

Acelerómetro y giróscopo

La funcionalidad esencial del dispositivo SOSmart Premium es la detección e identificación de choques automovilísticos, para lo cual es necesario medir las variables críticas para desarrollar los algoritmos: aceleración y ángulos de inclinación [17].

Con ambas, es posible saber cuándo un automóvil sufre cambios bruscos de aceleración (detección de choque) y en qué plano del espacio se encuentra al momento del choque (identificación del tipo de choque).

Medir aceleración de un determinado cuerpo siempre necesita tener una referencia, en el caso del planeta Tierra, esta se mide respecto al nivel de la superficie donde cada cuerpo a nivel del mar está experimentando una aceleración de $9.8[\frac{m}{s^2}]$ en dirección al suelo.

En el mercado, existen componentes electrónicos que miden la aceleración que están su-

friendo en cada momento respecto al mismo origen que los demás cuerpos. Esta aceleración se mide en **g**, siendo 1**g** la medida de aceleración que experimentan los cuerpos por la gravedad de la Tierra.

Por otro lado, medir los ángulos de inclinación respecto a un sistema de coordenadas también es posible con ciertos componentes electrónicos que poseen un giróscopo interno, en el que ese sistema de coordenadas viene fijado en el componente y las lecturas de los ángulos se comparan según esos ejes de referencia, por lo que la posición del componente es relevante.

Afortunadamente, en el mercado del hardware electrónico, existen acelerómetros de tamaños muy pequeños y a bajo costo, como por ejemplo, los ADXL3 que son comunes en aplicaciones y en kit de desarrollo electrónico, en la Figura 2.7 se muestra un tipo de estos componentes y que posee 3 ejes para medir aceleración (X,Y,Z).

Figura 2.7: Acelerómetro ADXL3. Fuente: [https://www.ecured.cu/Acelerómetro]

Dependiendo del tipo de acelerómetro, estos pueden variar en su rango de medición de aceleración: $\pm 1g$, $\pm 2g$, $\pm 4g$, $\pm 8g$ o $\pm 16g$, con precisión de 8, 16 o 32bits. Así, es posible detectar movimientos muy sutiles (bajo rango y alta precisión) o aceleraciones de gran intensidad como choques, como es el caso de este trabajo (alto rango).

Memoria Micro SD

En la Figura 2.8 se visualiza un componente para conectar a un controlador y leer información de una memoria micro SD extraíble, así como también escribir sobre ella en cualquier momento. Sus distintos pines conectados al controlador permitirán a este acceder a todos los archivos que estén dentro de la tarjeta micro SD, escribir sobre ellos, también hacer transferencia de archivos y manipular la información de múltiples formas posibles.

Existen muchos tipos de memoria micro SD con diferentes capacidades de almacenamiento. Estas se usan mayormente en smartphones para extender la memoria de estos y así poder almacenar una mayor cantidad de canciones, fotos o videos. En el mercado se encuentran tarjetas que van desde los 2GB hasta los 64GB de almacenamiento.

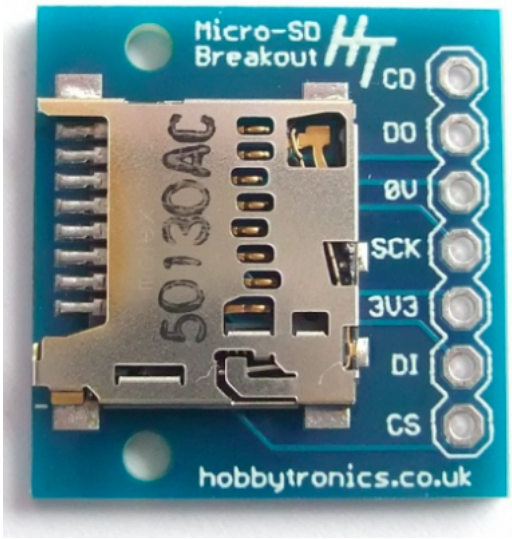


Figura 2.8: Breakout micro SD. Fuente: [http://www.hobbytronics.co.uk/microsd-breakout]

FONA 3G

El dispositivo SOSmart Premium debe tener la funcionalidad que los smartphones realizan por defecto: enviar SMS, establecer llamadas y obtener ubicación GPS. Para esto es necesario integran una SIM card activa a la red telefónica para que pueda comunicarse con los contactos predefinidos y además una antena GPS para obtener la coordenadas de posición en el momento del choque. Además, debido a que la red 2G pronto cerrará en Chile, es necesario que este módulo se conecte a la red celular a través de 3G.

Existe en el mercado internacional, un componente de la empresa Adafruit [18], una de las más importantes en cuanto a hardware electrónico del mundo, que puede realizar todas las tareas mencionadas en el párrafo anterior, el denominado FONA 3G (ver Figura 2.9)

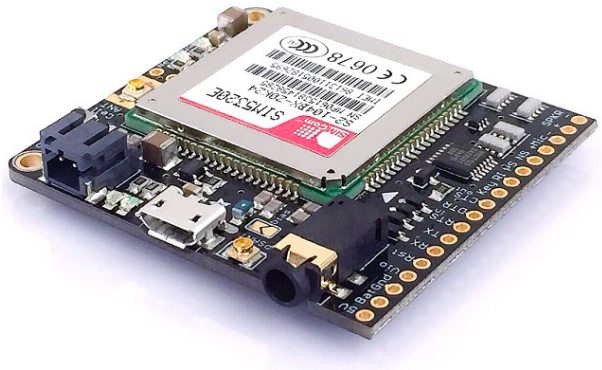


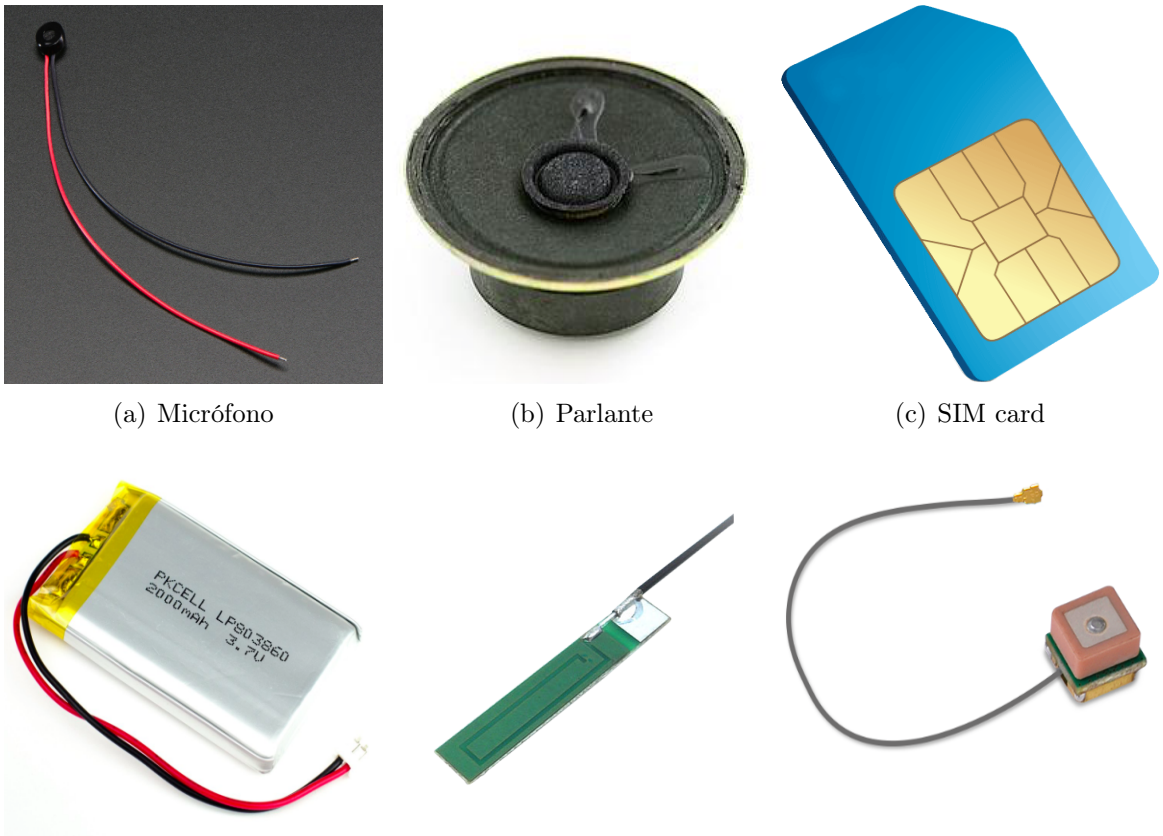
Figura 2.9: Adafruit FONA 3G Celular y GPS. Fuente: [http://www.cetronic.es/sqlcommerce/disenos/plantilla1/seccion/producto/DetalleProducto.jsp?idIdior

Esta placa tiene integrado un módulo SIM5320, que es el que permite realizar todas las funciones de un celular:

- Conectarse a antenas celulares GSM con red 2G/3G
- Realizar y recibir llamadas utilizando micrófono y parlante o auriculares
- Enviar y recibir mensajes SMS
- Posee integrado un GPS que puede ser controlado y pedido usando el mismo puerto serial

El FONA 3G además necesita una batería Lipo de 3.7V para soportar los peaks de corriente requeridos durante las llamadas y que se carga a través de un puerto micro USB que posee la placa.

En la Figura 2.10 se muestran los componentes que necesita el FONA 3G para realizar las funciones de un celular antes descritas.



(a) Micrófono (b) Parlante (c) SIM card

(d) Batería (e) Antena GSM (f) Antena GPS

Figura 2.10: Componentes del FONA 3G celular. Fuente: [https://learn.adafruit.com/adafruit-fona-3g-cellular-gps-breakout/arduino-wiring]

La programación del FONA 3G se realiza a través del Arduino, en donde Adafruit posee una librería abierta con documentación detallada e inclusive demos sobre cómo conectar el módulo con un Arduino y utilizar las funcionalidades del FONA 3G a través de código.

La comunicación entre ambas placas es serial utilizando los pines Rx/Tx, la alimentación de 5V y la batería de 3.7V conectada en todo momento. El desempeño del FONA 3G dependerá de la señal que obtenga de las antenas celulares, es decir, dependerá fuertemente de la

cobertura que tenga la SIM en el lugar.

Plug-In cigarrera

La mayoría de los vehículos actuales poseen una salida de 12V en la que se conectan Plug-in de cigarreras, es decir, un dispositivo que al conectarlo, se calienta y sirve para encender cigarros.

Otro uso que se le da a esta salida, es el de conectar un dispositivo que transforma la salida a 5V y en forma de USB, con la cual los pasajeros pueden conectar sus enchufes para recargar sus teléfonos celulares o cualquier otro dispositivo electrónico, como se muestra en la Figura 2.11.

Figura 2.11: Plug-In cigarrera con salida USB 5V. Fuente: [http://www.mcdigitalchile.cl/index.php?id$_product = 376controller = product$]

2.2.3. Software

En esta sección se describen los tres software que se utilizarán en este trabajo en las distintas etapas.

Arduino

Para programar el Arduino y conectar los diversos módulos externos, la misma plataforma ofrece un IDE de programación basado en el lenguaje C++, con las librerías necesarias para ejecutar los algoritmos y también conectar los diversos componentes. También provee una consola para mostrar datos en tiempo real de ejecución y un sistema de debbuging, para encontrar errores de ejecución.

El softare que provee Arduino es útil para todos los tipos de controladores Arduino, por lo que después de configurar el entorno para el tipo de Arduino, el código fuente funciona de igual forma para cualquier hardware.

Además, a través de su página web [16], se creó una comunidad vasta de usuarios de Arduino en el que se discuten innumerables temas y resuelven problemas de programación, haciendo de Arduino una herramienta fuerte pero también amigable tanto en hardware como también en software.

Matlab

Uno de los software más populares a nivel mundial para tareas de ingeniería es Matlab, debido a su gran capacidad de cómputo matricial, el manejo de estructuras de datos complejas, la gran variedad de librerías para ejecutar modelos de control e inteligencia artificial, procesamiento de señales e imágenes y también porque posee un simulador muy potente [19].

Si bien es un software bastante poderoso, tiene la limitación que no es compatible con sistemas embebidos de bajo procesamiento, debido a que es un software muy pesado en instalación y ejecución. En general se utiliza para tareas de post-procesamiento o análisis de datos guardados y no en tiempo real.

La licencia de este software no es gratuita y es costosa para personas, no así tanto para instituciones académicas o empresas de ingeniería. Para este trabajo, se utilizará Matlab versión 2013 que es la que tiene contratada la Universidad de Chile.

Unity3D

Este software permite la modelación y animación de objetos en un entorno, además de mantener muchos hilos de ejecución al mismo tiempo. Está diseñado para el desarrollo de videojuegos, utilizando 2 lenguajes de programación: Csharp y Javascript[20].

Las mayores ventajas que posee es que tiene una licencia gratuita para desarrollo y aprendizaje, una comunidad de soporte grande pero principalmente, una compatibilidad con muchas otras herramientas de desarrollo, pudiendo crear aplicaciones con casi todos los sistemas operativos de smartphones (Android, Iphone y Blackberry) y también con bases de datos como Parse.

En los últimos años, desarrolló una librería llamada Vuforia [21] que permite crear aplicaciones en un entorno de realidad aumentada o también de realidad virtual (AR/VR) y que en la actualidad han estado explotando el área con aplicaciones de todo tipo: libros interactivos, juegos de mesa en 3D, aplicaciones publicitarias, etc.

Unity tiene la facilidad de modelar y aplicar física a cuerpos en un ambiente gravitacional, por lo que usando este software es posible crear una simulación de accidente vehicular teniendo los datos de posición y rotación antes y durante el accidente.

2.3. Estado del arte

Con la explosión de los smartphones y vehículos con sistemas integrados más inteligentes, aparecieron diversas soluciones en el mundo para disminuir el tiempo de respuesta de emergencias, desde productos y servicios comerciales como también publicaciones en el mundo académico.

2.3.1. Soluciones comerciales

Sistema e-call

En Europa, como iniciativa de la Comisión Europea, nació la idea de implementar un sistema de emergencia el año 2009 que tuviese como objetivo proporcionar ayuda rápida a los automovilistas implicados en un accidente de tráfico en cualquier parte de la Unión Europea y así poder salvar la mayor cantidad de vidas posibles involucradas en choques automovilísticos.

Este sistema, llamado e-call, está acoplado al sistema electrónico interno de un vehículo y es capaz de conectarse con el punto de atención de llamadas de emergencia más cercano, al momento de detectar automáticamente un potencial accidente de tránsito o en respuesta a una activación manual de emergencia por parte del ocupante del vehículo[22].

En la Figura 2.12 se muestra el esquema general del sistema. Al momento de detectar un choque, se establece una transacción e-call con la red móvil que reconoce e-call como una llamada de emergencia 112 (número de emergencia único en Europa). Primero, se envían un conjunto mínimo de datos al operador de la central como la ubicación (coordenadas GPS), tipo, dirección y sentido del vehículo, tiempo del accidente y estimar el número de pasajeros según la cantidad de cinturones abrochados. Luego, se intenta establecer la llamada de audio con el conductor para determinar si es necesario movilizar los servicios de emergencia y en base a estos datos, el operador toma la decisión más adecuada para socorrer en el menor tiempo posible a los ocupantes del vehículo.

En caso de una llamada e-call manual por parte del conductor, se establece inmediatamente la llamada de audiocon el operador y este acudirá oportunamente a las necesidades que presente el conductor en dicho momento.

Para que el despliegue del sistema eCall haya sido efectivo se ha necesitado el compromiso de los Estados miembros de la UE, así como de todas las partes interesadas: fabricantes de automóviles, de equipos telemáticos, operadores de telefonía móvil, proveedores de servicios, protección civil, Centros 112 o PSAP (Public Safety Answering Point), ya que estos deben garantizar la implantación del eCall llevando acabo los siguientes pasos:

- Promover activamente la utilización del número de urgencia único europeo 112 y adoptar las medidas necesarias para acelerar la introducción de los datos de localización.
- Firmar el Memorándum de Acuerdo de eCall en todos los países europeos.

Figura 2.12: Esquema del sistema e-call implementado en Europa. Fuente: [https://ec.europa.eu/transport/themes/its/road/action$_p lan/ecall_e n$]

- Modernizar los centros de emergencias que reciben las llamadas 112 y las efectuadas por el sistema eCall, para diferenciar ambas llamadas.
- Mejorar sus protocolos de actuación en caso de urgencia, velar por la buena formación del personal involucrado y garantizar la atención lingüística necesaria.

El e-call lleva implantado varios años en algunos países como Suecia, Finlandia, Reino Unido, Holanda o Eslovenia y algunas ciudades de España. Obteniendo positivas conclusiones usando el sistema: por ejemplo en Finlandia, en su primer año de implementación se redujo un 10 % el número de muertes por accidentes de tránsito [23].

Sin embargo, la implementación del sistema a nivel europeo, ha sufrido un gran retraso y gasto económico, debido a la gran dificultad de poner de acuerdo al mismo tiempo a todas las partes involucradas mencionadas anteriormente. La UE fijó como plazo el año 2018, fecha para la cual todas las empresas fabricantes de automóviles comiencen a integrar en su vehículos el sistema e-call.

Se estima que e-call tiene el potencial de salvar 2.500 vidas al año en Europa cuando esté introducido en todos los vehículos, así como reducir la gravedad de las secuelas en los heridos por accidentes de tráfico en un 10-15 % de los casos y reducir el tiempo de respuesta de los servicios de emergencia en un 50 % en zonas rurales y en un 40 % en las urbanas.

Onstar

Otro sistema de atención de emergencias de vehículos es el servicio de On Star, compañía subsidiaria de General Motors (multinacional estadounidense que diseña, fabrica, publicita y distribuye vehículos y piezas de estos, además de vender servicios financieros) fundada en 1996 en colaboración con Electronic Data Systems (desarrollo de sistemas, manejo de información y soporte técnico de tecnología) y Hughes Electronics Corporation (tecnología satelital y electrónica automotora).

OnStar, es un sistema que se integra a la electrónica del vehículo y provee servicios de comunicación en seguridad (accidentes y robos), realizar llamadas con manos libres, asesoría de navegación y diagnósticos remotos de conducción, entre otros. En la Figura 2.13 se muestra el espejo retrovisor de un vehículo marca Chevrolet Cruze que integró la tecnología OnStar y que el usuario puede utilizar a través de esos comandos en el espejo.

Figura 2.13: Tecnología OnStar integrada en un vehículo Chevrolet. Fuente: [http://www.chevroletuniversidad.com/OnStar.php]

OnStar es un servicio disponible en varios países del mundo, incluyendo México, Estados Unidos, Brasil, Argentina, China y buena parte de Europa. Teniendo más de 6 millones de usuarios en el año 2011 [24], aunque si la persona no se subcribe a los planes, los servicios estarán desactivados. Estos planes varían según precio y características[25]:

- Básico ($0): Gratis los primeros 5 años, incluye acceso remoto, diagnóstico avanzado e inteligente de conducción para proveer consejos de conducción.
- Protección ($19.99/m): Incluye un sistema que detecta y notifica colisiones del vehículo (a través de sensores puestos en los air-bags). Además de servicios de emergencia y asistencia en carretera.
- Seguridad ($24.99/m): Posee adicionalmente al plan anteior de Protección, un servicio de asistencia en caso de robo del vehículo.
- Guía ($34.99/m): Posee adicionalmente al plan anteior de Seguridad, un servicio de navegación en tiempo real, comunicación con operadores de OnStar en todo momento y algunos minutos gratis para realizar llamadas durante la conducción.

Automatic Pro

Automatic es un dispositivo que se instala en el puerto OBD-II de un automóvil para acceder a los datos del vehículos y conectarse con una nube, para después haciendo uso de una aplicación, obtener estadísticas de mantención, conducción y seguridad del vehículo. En la Figura 2.14 se muestra el dispositivo.

Las principales características que posee es que puede obtener diagnósticos técnicos de

mantención del automóvil (combustible, estado de batería, aceite, motor, etc), sincronización con una nube a través de 3G, monitoreo del vehículo en estado de conducción y estacionamiento y una serie de servicios para mejorar la calidad de conducción de la persona (conexión a Google Street para sugerir lugares a visitar y rutas óptimas) [26].

Además, posee un sistema de detección de accidentes que notifica al teléfono y también a operadores del servicio que llamarán al usuario del teléfono, en caso de existir alguna emergencia, los operadores notifican a familiares y también activa los servicios de emergencia para socorrer al conductor. Sin embargo, si el dispositivo resulta dañado o desconectado del puerto después del choque, o no hay señal GPS o celular, las notificaciones no se pueden enviar.

El dispositivo es compatible con la mayoría de los vehículos fabricados después del año 1996, aquellos que utilizan combustible, diesel o híbridos (no eléctricos). En cuanto a la aplicación, es gratuita y tiene soporte para sistemas operativos iOS 10+ y Android 5.0+. Finalmente, dependiendo de la cantidad de servicios que el dispositivo pueda proveer, el precio de adquisición varía entre $80 y $130 dólares.

Figura 2.14: Adaptador Automatic para puerto OBD-II de vehículos. Fuente: [https://www.automatic.com/pro/]

2.3.2. Publicaciones académicas

Sensor bidimensional para predicción de choques automovilísticos.

El enfoque que tratan de dar para detectar choques en [27], no es el uso de acelerómetros, sino de un sensor magnetoresistivo y un sonar que alerten de una colisión inminente.

El sensor magnetoresistivo es usado para calcular el campo magnético de otro vehículo cercano para estimar su posición relativa, velocidad y orientación. Esto no es un cálculo

trivial debido a que la variación del campo magnético se modela con una función analítica de parámetros desconocidos según el tipo y modelo de vehículo, por lo que se requiere uso de un sonar y un estimador adaptativo (EKF). En la Figura 2.15 se muestra un esquema de la posición y los parámetros a ser estimados durante una colisión.

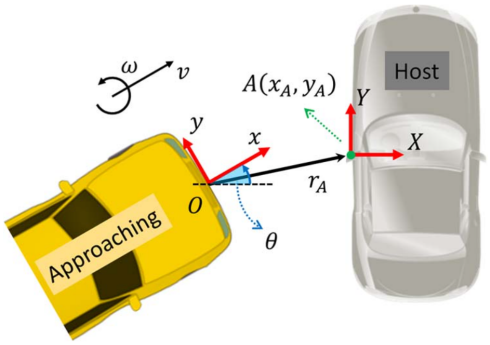


Figura 2.15: Estimación bidimensional de la posición y los parámetros a ser estimados [27].

Los resultados fueron validados en pruebas de laboratorio y confirman la viabilidad del sistema. Sin embargo, en la práctica requiere un hardware muy potente (muchos cálculos) y costoso, por lo que utilizar este sistema en cada vehículo resulta imposible, además que este paper no considera choques con objetos que no son vehículos o volcamientos.

Aplicación para smartphone

Con las mejoras tecnológicas de la electrónica en los últimos años, se generó una explosión en la fabricación de telefónos que cada vez eran más inteligentes y polifuncionales. Realizar llamadas y mandar mensajes era algo muy básico y el enfoque estaba en que los celulares fuese capaces de realizar cada vez más tareas a través del uso de aplicaciones.

Una de las mejoras significativas, fue la inclusión de un acelerómetro y giróscopo en la electrónica del teléfono que permitió crear aplicaciones para sensar aceleración e inclinación del teléfono. De esta manera, fue posible crear aplicaciones para detectar choques automovilísticos utilizando los propios sensores del smartphone y aprovechando su comunicación GSM y GPS, para mandar notificaciones alertando del accidente.

Dentro de las mayores ventajas de utilizar una aplicación de celular en comparación a los sistemas convencionales intrínsecos del vehículo, es que son independientes del vehículo y aún así pueden proveer información valiosa de la conducción de este como también de accidentes, además de videos y audios para facilitar los servicios de emergencia.

Se mencionó anteriormente que la característica principal para detectar un choque es la aceleración experimentada durante este. En un sistema intrínseco del vehículo como OnStar o E-call, estos sensores están puestos en la parte delante del vehículo y son los mismos que utiliza el air-bag para activarse. Sin embargo, si bien en la posición de los pasajeros se experimenta una aceleración mucho menor, sigue siendo lo suficientemente grande (sobre 4g) para poder detectarla con los acelerómetros de los celulares y además para diferenciarse de otro tipo de impactos más pequeños, como una caída de celular o un freno brusco [28].

En [29] estipulan que choques a menor velocidad que 24 $\frac{km}{h}$ no superan los 4G de aceleración y podrían no ser detectados. Sin embargo, se propone la utilización del micrófono del celular para detectar cuando una saturación de este podría significar un choque y también imágenes y videos a través de la cámara que podrían ayudar a la detección de un incidente. Claro está que el celular en este caso no debe estar expuesto a música fuerte o ruidos de muchos decibeles dentro del vehículo, lo que imposibilita al usuario a hacer uso del teléfono mientras conduce.

También se propone utilizar la velocidad que registra el GPS para discernir entre una caída o movimiento fuerte de celular (ver Figura 2.16 en donde hay momentos que la aceleración medida supera el umbral 4g) y un choque leve (aceleraciones cercanas), para eso se procesa la variación de velocidad en el tiempo y bajo ciertos parámetros es posible diferenciar cada caso, haciendo más robusto el sistema de detección.

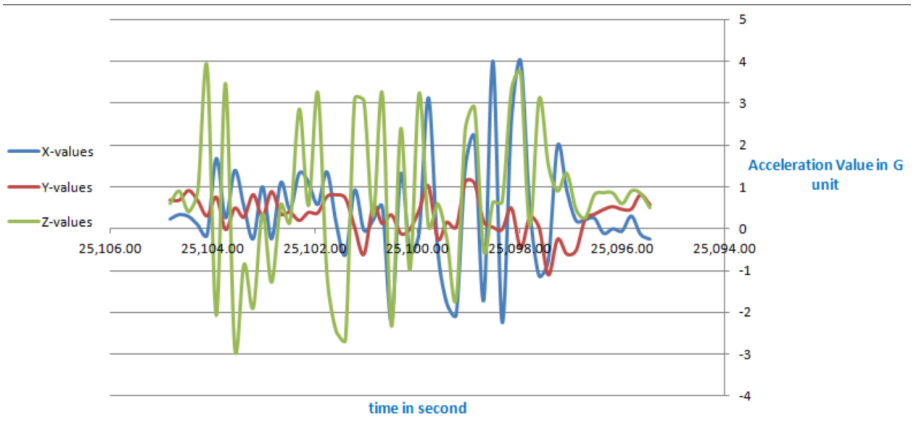


Figura 2.16: Valores de aceleración durante un movimiento fuerte de un smartphones. Caso para la utilización del GPS para desechar esta detección [30].

En [30] proponen una técnica matemática (Dynamic Time Warping) para además determinar la magnitud del accidente y disminuir los falsos positivos, a través de la superposición de vectores de aceleración.

Una vez detectado el choque, la aplicación debe enviar las coordenadas e información básica a un servidor o nube, a través de peticiones HTTP o URL y almacenar la información en base de datos para posteriormente ser leídas a través de una plataforma web de monitoreo. Además es posible incluir en el sistema, testigos del accidente que suban a través de la aplicación imágenes y videos para mandar al centro de monitoreo y que estos puedan tomar medidas más pertinentes respecto al caso.

Por otro lado, enviar mensajes SMS de emergencia a contactos cercanos. El primer enfoque, alude a tener una central operadora pendiente de todos los accidentes que se vayan detectando y que estas se comuniquen oportunamente con los conductores para prestar la atención de emergencia necesaria (e-call, OnStar, Automatic), el segundo básicamente a que sean los propios usuarios los encargados de comunicarse con una central de emergencia para alertar del accidente (Sosmart).

Cláramente, las ventajas del primero van por el lado de la rapidez de acción y que una

operadora pueda estar pendiente en todo momento de monitorear posibles accidentes, detalle no menor considerando lo importante que es cada minuto que pasa después de un choque automovilístico para pretender salvar la vida del conductor y pasajeros. Sin embargo, esta coordinación también requiere un costo mayor y por tanto estas soluciones tienden a ser mucho más caras para los usuarios. A diferencia de Sosmart que posee el segundo enfoque y es de carácter gratuito para los usuarios.

Sistema inteligente de notificación de accidentes vehiculares

Investigadores españoles estipulan otro paradigma en los sistemas de ayuda inmediata en accidentes vehiculares, en el queaedmás de centrarse en la detección también lo hacen en la forma de notificar el accidente a los centros de emergencia, de manera que la información llegue rápidamente y lo más completa posible. [31].

Para ello, pretenden que una vez detectado un accidente a través de un hardware instalado en el vehículo, la información llegue a las centrales médicas a través de una red vehicular (comunicación inalámbrica V2V y V2I) y que utilizando data mining e inferencia, determinar la gravedad de este según las variables del accidente (velocidad, tipo de vehículo, impacto y estado de los airbags) al ser comparados con datos almacenados en una gran base de datos.

Por tanto, no sólo la detección aporta información, sino que en la misma red, llega esta información, se estima la gravedad del accidente usando algoritmos de machine learning con una base de datos entrenada y se notifica a las unidades de emergencia para que prioricen y optimicen la ayuda.

En la Figura 2.17 se muestra la arquitectura del sistema propuesto.

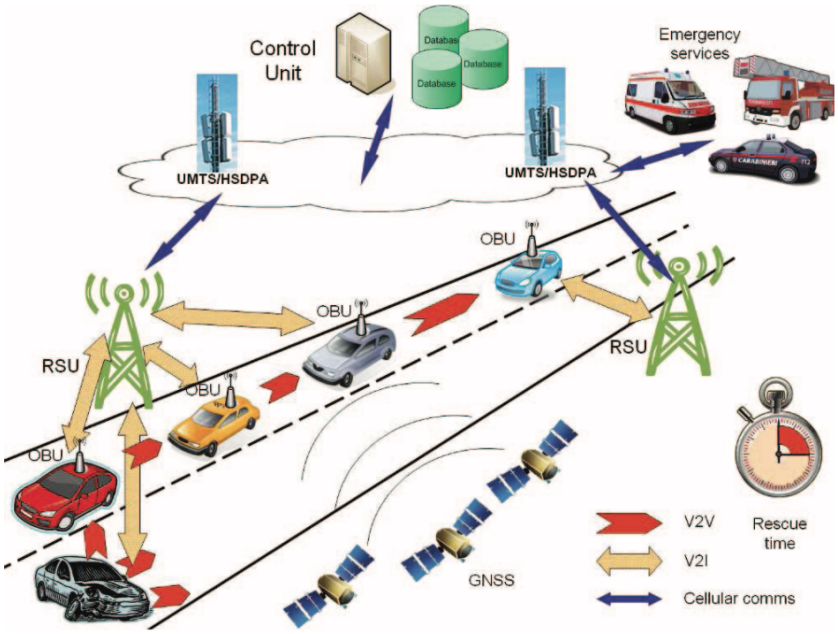


Figura 2.17: Arquitectura del sistema propuesto para la notificación automática de accidentes vehiculares y asistencia utilizando redes vehiculares [31].

Capítulo 3

Metodología

Para desarrollar este proyecto, se siguieron una serie de pasos que se describen en orden a continuación:

- Estudio de características en la acelaración durante un accidente: este primer paso fue buscar una base de datos de accidentes automovilísticos y obtener las principales características para que, en base a esto, construir o adaptar un algoritmo de detección de choques, haciendo uso de un sensor de aceleración.
- Selección de componentes electrónicos: este segundo paso, consisitó en seleccionar los componentes electrónicos que forman parte del diseño del dispositivo que se conecta al automóvil y realiza la tarea de detectar, localizar y comunicar el accidente automovilístico. Este diseño se basó en el uso de componentes accesibles en el mercado, de bajo costo y la utilización de una plataforma robusta y de fácil uso, como es Arduino.
- Implementación del hardware y software: en este paso se realizó la conexión entre estos componentes para dejar funcional el dispositivo. Para esto se utilizaron librerías liberadas y bien documentadas que permitieron la integración de cada módulo con el controlador Arduino central, es decir, se integró el módulo de comunicación, el de almacenamiento de información y el acelerómetro. Finalmente, se implenentó el algoritmo de detección diseñado anteriormente.
- Construcción de prototipo y pruebas: Se trabajó en conjunto de un diseñador industrial para diseñar la placa externa que dio término a la construcción del prototipo. Una vez funcional, se realizaron pruebas simuladas de accidentes con aceleraciones de menor escala y también dentro de un automóvil, para obtener un desempeño cuantitativo del algoritmo.
- Post-procesamiento de señales y simulación: el último paso, fue la ejecución de un post-procesamiento de los datos almacenados que determinaron la gravedad del accidente y que finalmente, haciendo uso de un software de modelación 3D, se realizaron simulaciones de los instantes previos y durante del choque.

Capítulo 4

Diseño

En esta sección se explica detalladamente el diseño general del sistema de detección, localización y caracterización de un accidente, así como los agentes involucrados. Además de la elección e integración de los componentes que forman el dispositivo electrónico.

4.1. Arquitectura del sistema

Como se ha dicho en secciones anteriores, la idea de detectar rápidamente un choque es que después se pueda prestar la ayuda médica lo más prontamente posible para disminuir el riesgo de secuelas o muerte de los pasajeros.

Puesto que la finalidad de este proyecto no varía con respecto a las soluciones presentes en el mercado, es que la arquitectura del sistema a construir es muy similar a la de dichas solucionas. Este esquema se muestra en la Figura 4.1.

En el cuadro central de la Figura 4.1 se tiene el choque del automóvil, el cual estará usando en todo momento el dispositivo electrónico y que detectará el instante cuando el choque se produzca. Luego, el dispositivo alertará vía llamada telefónica y SMS (el SMS es necesario para enviar las coordenadas del lugar del accidente en caso que no se pueda realizar la llamada por daño del dispositivo o que el pasajero esté inhabilitado de contestar) utilizando la red telefónica nacional, a contactos familiares del ocupante (que puedan avisar a un centro médico cercano del accidente) o directamente a una central de ayuda médica, ambas opciones predefinidas dentro del dispositivo. Así, se pueda enviar rápidamente la ambulancia con las medidas necesarias para el rescate de las personas al interior del vehículo.

Adicionalmente a estas 2 formas de notificación, también se utiliza una notificación web (Post Request) directo a una base de datos determinada, para que sea desplegada a través de una página web de Internet. Este sistema es el más útil en centro de emergencia en el que operadores están monitoreando continuamente servicios de emergencia, ya que en una misma página web se muestra un mapa con la ubicación de los accidentes que van siendo detectados en tiempo real.

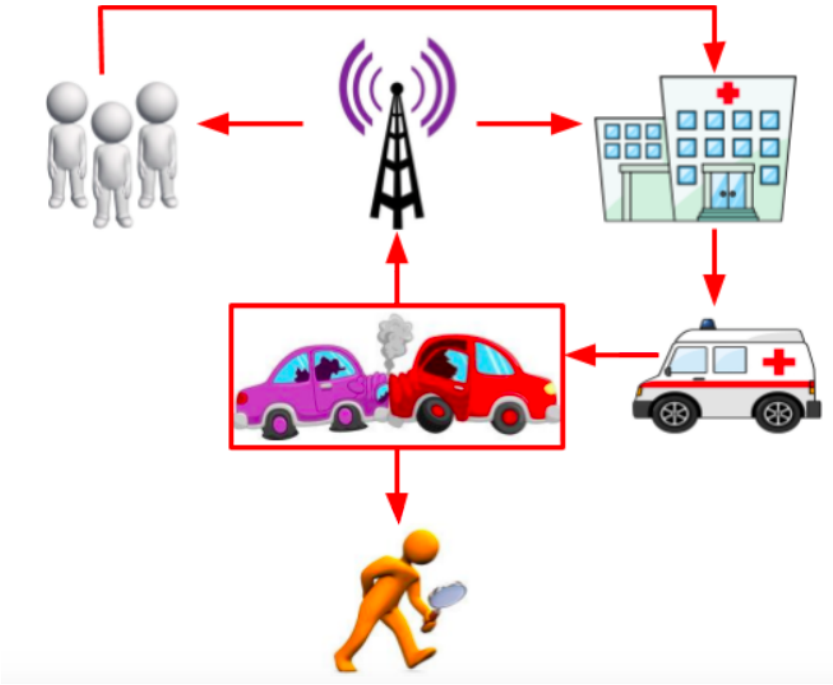


Figura 4.1: Esquema del sistema completo de detección, localización y caracterización de accidentes automovilísticos.

Otro agente importante en este sistema, es el de la investigación del accidente, que muchas veces involucra peritaje costoso y procesos judiciales largos hasta aclarar las causas del accidente. Es por esto, que el sistema a construir también consta de una fase de post-procesamiento en el que utilizando los datos guardados en una memoria extraíble dentro del dispositivo electrónico, se modela la simulación de los instantes previos y durante el accidente utilizando un software de modelamiento.

4.2. Algoritmo de detección de choques

Como se mencionó en la sección 2.1.1, en un choque de cuerpos la variable más representativa es la magnitud en la desaceleración experimentada y que revela la gravedad del este choque. Es por esto, que el componente esencial que el dispositivo electrónico debe integrar es un acelerómetro electrónico, cuya lectura de aceleraciones en los 3 ejes espaciales se debe hacer usando un controlador y a su vez, este controlador ejecutar un algoritmo continuamente para detectar cuándo ocurre un choque.

Según la base de datos utilizada de aceleraciones experimentadas durante un choque (*National Highway Traffic Safety Asocitation*)[32], poco importa la aceleración particular en alguno de los ejes, sino que es más representativo el módulo de las 3, dada por la ecuación 4.1 y que en choques de gran magnitud, este valor puede superar los 20g con facilidad en un rango de 0.1-0.5[ms] [33].

$$A = \|A_x + A_y + A_z\| \tag{4.1}$$

Los acelerómetros de los smartphones, poseen un buen rate (100Hz) pero no superan los 3g de medición, siendo un rango suficiente para aplicaciones deportivas, caminatas o de movimientos suaves. Pero no suficiente para discernir entre un choque (>20g) o agitar el celular o una caída del mismo (4g), debido a que ante ambos eventos, el acelerómetro se satura y mide lo mismo [30]. Debido a esto, SOSmart y otras aplicaciones que intentan detectar choques utilizan además del acelerómetro, el GPS para determinar la velocidad y de esta forma aislar estos casos en que ocurren desaceleraciones altas pero no debido a un choque pues la velocidad indica que el aparato no va dentro de un vehículo en gran movimiento[29].

Los acelerómetros electrónicos de mercado tienen una gran ventaja respecto a los embebidos en los smartphones, y es que son capaces de medir en un rango mucho más amplio alcanzando 16g algunos de ellos. Sin embargo, este aumento de rango tiene la desventaja que la precisión disminuye (debido a que en su procesador, los bits deben representar un rango más amplio), incluyendo más ruido en las mediciones y la presencia de *outliers*.

Es por esto, que el algoritmo de detección a desarrollar se basa en el desarrollado en la empresa para la aplicación SOSmart [7] pero con una leve adaptación debido al mayor rango de medición del acelerómetro a utilizar. A continuación se resumen estas características principales:

- Se basa sólo en el módulo de la aceleración, aunque se debe considerar 1g adicional producto de la aceleración de gravedad presente en las mediciones.
- Debido a su amplio rango de medición, no es necesario utilizar la información del GPS. Sólo definir apropiadamente los umbrales para la detección del accidente.
- Debido a su mayor imprecision en la medición, se deben aplicar histéresis y ventanas de tiempo para atenuar el ruido y los outliers presentes en los datos. De esta forma, el algoritmo descartará un gran nùmero de falsos positivos.
- El controlador a utilizar no posee una potencia computacional elevada, por lo que aplicar cálculos muy pesados podría bajar la frecuencia de adquisición, que en definitiva es lo más importante.

Se descartó el uso de algoritmos de clasificación debido al costo computacional de estos principalmente, entre ellos: series de tiempo, DTW, SVM, redes neuronales. El uso de un algoritmo simple, con regiones de histéresis y umbrales parametrizados correctamente, es suficiente para la detección de desaceleraciones a nivel de choque.

Los parámetros a definir son:

- Umbral de entrada a histéresis
- Umbral de salida de histéresis
- Umbral de deteccion
- Número de puntos mínimos para entrar/salir de histéresis
- Número de detecciones consecutivas mínimas para activar detección.
- Tamaño ventana de búsqueda
- Tamaño ventana de detección

El pseudo código del algoritmo se muestra a continuación:

```
Data: A_x, A_y, A_z
Result: FueDetectado
while Existe Data do
    A = modulo(A_x,A_y ,A_z);
    Ventana-Busqueda.agregarNuevo(A);
    Ventana-Deteccion.agregarNuevo(A);
    if Esta fuera de region de histeresis then
        if Ventana-Busqueda posee más de un minimo de puntos mayores a Umbral de
         entrada then
            Entrar a region de histeresis;
        else
    else
        if Ventana-Busqueda posee menos de un minimo de puntos mayores a Umbral
         de salida then
            Salir de region de histeresis;
        else
    end
    if Esta dentro de region de histeresis then
        if Ventana-Detección.promedio mayor a Umbral de detección then
            Aumentar contador de detecciones consecutivas;
        else
            Contador de detecciones consecutivas = 0;
        end
    else
        Contador de detecciones consecutivas = 0;
    end
    if Contador de detecciones consecutivas es suficiente then
        FueDetectado;
    else
    end
    Ventana-Busqueda.borrarViejo;
    Ventana-Deteccion.borrarViejo;
end
```

El algoritmo consiste en obtener continuamente los datos de aceleración desde el acelerómetro y aplicar módulo para obtener la magnitud del vector final de aceleracíon. Este valor se almacena en 2 ventanas, una denominadan de búsqueda y otra de detección, siendo la primera de mayor tamaño que la segunda.

La ventana de búsqueda, es aquella donde se determina entrar o salir de la zora de histéresis de detección. Esta decisión se toma según la cantidad de puntos dentro de la ventana que superen el umbral de entrada o salida a la zona de histéresis.

Al momento de entrar a la zona de histéresis, se toma la ventana de detección que es más pequeña y es donde se determina si ocurre una detección o no. Esto se realiza utilizando la media de los valores de esta ventana y evaluar si esta media supera un umbral de detección.

En caso de ser positivo, se aumenta un contador de detecciones consecutivas y en caso de ser negativa, este contador se resetea.

Finalmente, se evalúa si este contador de detecciones consecutivas es lo suficientemente alto para decidir si fue un choque como tal y alertar al controlador para que ejecute la tarea de notificaciones a través del FONA. De esta forma, el algoritmo asegura que no se trató de un simple Outlier y por ende, es muy probable que la detección sí corresponda a un choque.

4.3. Diseño del dispositivo

El punto central del sistema a implementar y de este trabajo, es la construcción del prototipo de dispositivo electrónico que detectará el accidente y enviará las notificaciones. Las características que debe tener este prototipo se describen a continuación:

- Conexión a través de la cigarrera del automóvil, que posee una salida de 12V y se encuentra en posición horizontal en la mayoría de los modelos de automóvil.
- Detección de choques en tiempo real
- Establecer llamadas a contactos predefinidos
- Envío de SMS con las coordenadas del choque a contactos predefinidos
- Envío de Post Request con las coordenadas del choque a una base de datos predefinida.
- Botón de pánico con el cual establecer llamadas de emergencia a un contacto predefinido
- Almacenamiento de datos para realizar post-procesamiento de accidente.

Para que el dispositivo cumpla con todos estos puntos, es necesario utilizar diversos compoenentes electrónicos e integrarlos de manera inteligente. Afortunadamente, en el mercado existe una variedad de opciones que permiten lograrlo. Sin embargo, algunos escapan del presupuesto de la empresa y son muy difíciles de usar o integrar, por lo que la elección se reduce a aquellos disponibles el mercado chileno, que sean de uso estándar en prototipado y además, que sean aprobados por la empresa para ser adquiridos.

Para el módulo de comunicación, la empresa dispone de un FONA 2G, que posee similares características y librerías para utilizarlo que el FONA 3G descrito en la sección 2.2.2, salvo por el uso de la red celular 2G. En algunos países, esta red está cerca a quedar obsoleta y por ende no sería una opción válida utilizarla para comunicación. Pero en Chile aún quedan algunos años para esto y además, dado que se trata de la construcción de un prototipo, el uso del módulo 3G no amerita obligación por lo que el FONA 2G de Adafruit, será el módulo de comunicación a utilizar.

Junto con este FONA 2G, también se necesita de una batería lipo de 3.3V, un parlante de 20[Ω], un micrófono, una antena GSM, una antena GPS y una mini SIM card. Todos estos elementos forman parte del kit de venta del Fona 2G [34] y fueron adquiridos por la empresa, por lo que serán utilizados para este proyecto. Salvo la sim card que en este caso se utilizará una Virgin Mobile prepago que se dice tiene muy buena cobertura a nivel nacional y además funciona con la red 2G.

Lo siguiente, es definir qué controlador usar para conectar el módulo de comunicación y los demás componentes. En la sección 2.2.2 se describió la facilidad que tiene la plataforma Arduino para ejecutar una gran cantidad de tareas y que sería una muy buena alternativa si tuviera un menor tamaño.

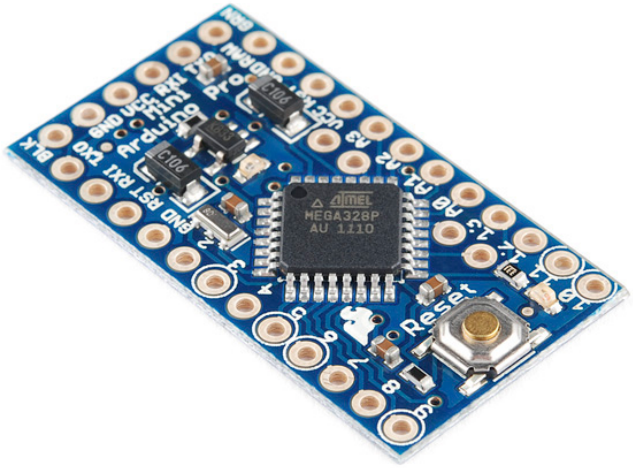


Figura 4.2: Placa Arduino Pro-Mini 5V. Fuente: [https://learn.sparkfun.com/]

Afortunadamente, Arduino desarrolló una placa especialmente pequeña para formar parte de sistemas electrónicos integrados, esta es Arduino Pro-Mini (ver Figura 4.2), que en cuanto a procesador es casi tan potente como el de Arduino UNO aunque tiene menor cantidad de entradas/salidas, además de prescindir de entrada USB para programar la placa directamente. Debido a esto, la programación del Arduino Pro-Mini puede realizar usando los 5 pines del costado menor y usando un adaptador USB serial o también usando un Arduino UNO como puente de programación [35].

Posee entradas digitales de comunicación serial (Tx,Rx) necesarias para comunicarse con el FONA 3G, entradas análogas para conectar sensores (como por ejemplo, el acelerómetro/giróscopo), digitales para conectar el botón y entradas de voltaje y tierra, para la alimentación del controlador (5V).

En cuanto al acelerómetro, uno de los más completos y con mayor documentación es el MPU9250 [36], que posee 3 ejes para medir aceleración (X,Y,Z), 3 ejes para medir grados de inclinación (Raw, Pitch, Yaw) y además 3 ejes para medir intensidad magnética, es decir, el mismo componente es un acelerómetro, giróscopo y magnetómetro (Ver en Figura 4.3).

Una característica importante de este componente, es que también posee un alto rango de medición de aceleración: $\pm 2g$, $\pm 4g$, $\pm 8g$ y $\pm 16g$ con precisión de 16 bits, es decir, la precisión suficiente para separar grandes desaceleraciones como en un choque, de otras más pequeñas como caídas o frenos bruscos a baja velocidad. Además, posee un rango de medición de ángulos de inclinación: ± 250, ± 500, ± 1000 y ± 2000 grados por segundo, con precisión de 16 bits en la lectura también, por lo que es posible hacer una lectura en tiempo real de los ángulos en cada momento.

Para ir guardando estos datos de aceleración y ángulos de inclinación, es necesario integrar un componente de almacenamiento de datos. Para este proyecto, se elige trabajar con un

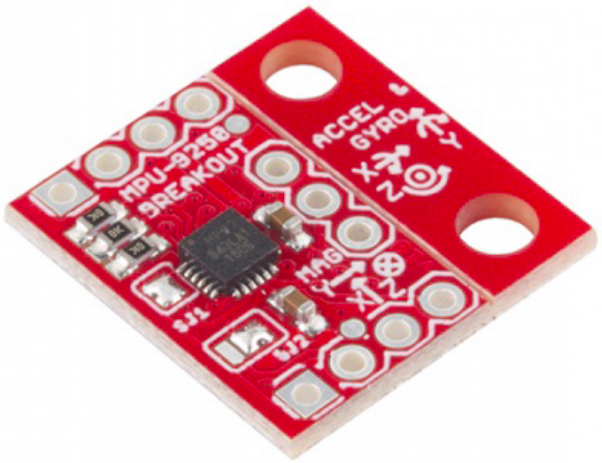


Figura 4.3: MPU9250 con acelerómetro, giróscopo y magnetómetro. Fuente: [https://learn.sparkfun.com/]

SparkFun microSD Transflash Breakout (Figura 4.4) debido a que se encuentra en el mercado a muy bajo costo y también posee librerías nativas para trabajar en Arduino, lo que lo hace muy robusto y fácil de usar [37].

En este caso, los datos a guardar son datos de bajo tamaño (números de aceleración y giróscopo), por lo que una tarjeta de 2GB sería suficiente para almacenar datos hasta 30 minutos, lo que es más que suficiente para determinar el tipo de choque que no deben ser más que un par de segundos.

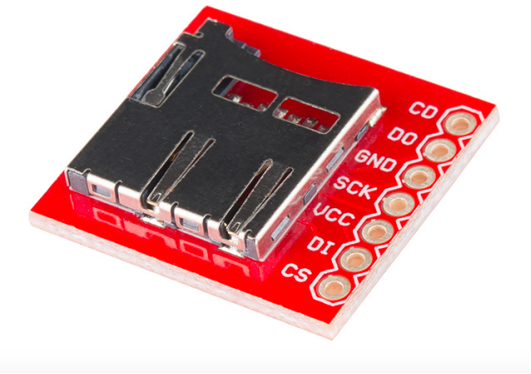


Figura 4.4: SparkFun microSD Transflash Breakout. Fuente: [https://www.sparkfun.com/]

Como se dijo al comienzo de esta sección del documento, el dispositivo pretende simular la aplicación SOSmart pero sin utilizar el celular. Sin embargo, intervenir eléctricamente el automóvil tampoco es negocio para las empresas por lo que una buena solución es usar la cigarrera del automóvil como fuente de energía.

A estas alturas, todos los vehículos cuentan con un plug-in de 12V de salida en el que la mayoría de los usuarios conecta cigarreras o puertos USB para cargar las baterías de sus

celulares o cualquier otro dispositivo electrónico, como se describió en la sección 2.2.2. La idea es que el dispositivo se alimente de esta fuente de 12V para funcionar. Pero dado que es de gran uso para los usuarios que requieran cargar celulares, sería muy útil que el dispositivo cuente con salidas USB de 5V para que el dispositivo no implique un estorbo al usuario. Sin embargo, esta característica no está incluída en los objetivos de este trabajo.

Para no implementar desde cero el circuito interno de estos plug-in que realizan la transformación de voltaje, se reciclará el circuito de un plug-in común de cualquier tienda de dispositivos electrónicos. En este caso se usará el que se muestra en la Figura 4.5.

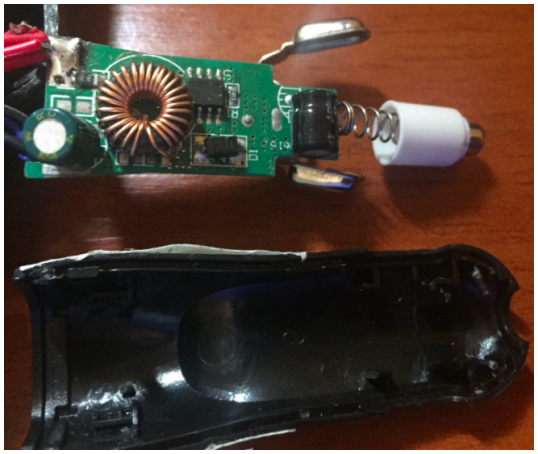


Figura 4.5: Circuito electrónico de un plug-in de cigarrera. Entrada de 12V y salida de 5V.

Además, en estricto rigor, sería necesario realizar en paralelo otra transformación de voltaje para cargar la batería de emergencia de 5V que alimentará el circuito cuando el dispositivo se desconecte de la cigarrera. Este caso, es para que en caso de choque y de dañarse la entrada de cigarrera, el dispositivo siga funcionando para enviar las notificaciones o realizar las llamadas necesarias para socorrer a los pasajeros. Sin embargo, esto también escapa de los objetivos centrales de la memoria, por lo que no se realizará.

Hasta aquí se han mencionado todos los componentes electrónicos específicos que se utilizarán para armar el dispositivo, cuya integración debe usar el menor espacio posible para que el resultado final sea un prototipo usable para un conductor. Es por esto, que una parte fundamental del dispositivo es su parte visible y la interacción que el usuario tendrá con este. En la siguiente lista mencionan las características, que se complementan con las descritas al principio de la sección:

- Posee un botón de pánico.
- Micrófono en el sector izquierdo (cerca al conductor) para que este pueda hablar en la llamada.
- Parlante en la parte superior para que se escuche en altavoz la llamada.
- Espacio para puertos USB en la parte frontal para que el usuario pueda cargar aparatos electrónicos
- Logo de la empresa Sosmart Labs al costado.
- Plug-In de entrada a la cigarrera del automovil con anclaje fijo, para que el dispositivo no se mueva.

Al ser un diseño de producto único, este tipo de carcasas no se encuentran en el mercado y la única opción es modelarla y fabricarla. Este tipo de trabajo escapan de las habilidades que un ingeniero eléctrico debe poseer por lo que se necesitará la colaboración de un diseñador industrial que diseñe, modele digitalmente y mande a fabricar a impresoras 3D una carcasa que pueda alojar los componentes electrónicos y tenga las características antes mencionadas. En la Figura 4.6 se muestra un diseño preliminar de la carcasa hecha por el diseñador industrial contratado por la empresa.

Hay que tener en cuenta, que por dentro esta placa plástica debe contener todos los componentes electrónicos que se usarán para hacer funcional al dispositivo, en especial el acelerómetro/giróscopo, para que quede fijo y así los datos tomados no contengan errores sistemáticos.

Si bien la calidad de las carcasas producidas por impresoras 3D no es apta para productos comercializables, es suficiente para un prototipo como el que se está desarrollando en este trabajo.

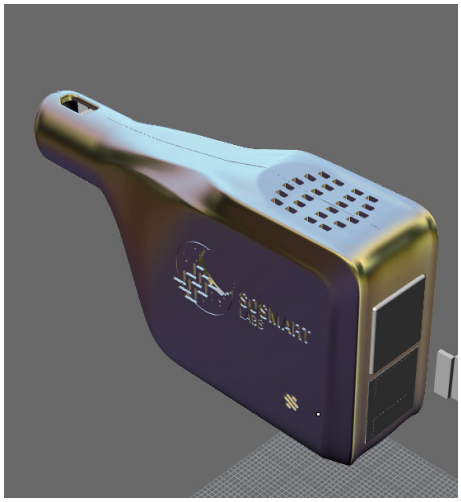


Figura 4.6: Modelo digitalizado de carcasa del prototipo.

Capítulo 5

Implementación

En esta sección se describe detalladamente todo el proceso de implementación del sistema, comenzando con la implementación del algoritmo en lenguaje C para Arduino, la integración completa del dispositivo y finalmente el post-procesamiento de los datos para determinar cómo fue el choque.

5.1. Algoritmo de detección de choque

Como se describió en la sección 4.2, el algoritmo de detección tiene ciertos parámetros que definir para su funcionamiento. La configuración de estos depende del acelerómetro mismo (ruido en la lectura), la frecuencia de adquisición, la magnitud y duración de las aceleraciones medidas durante choques reales.

A partir de los datos de la base de datos de choques automovilísticos de la NHTSA, se observó en diferentes curvas de aceleración, que los accidentes duran entre 0.03 y 0.05 [s] y aquellos que resultan más graves, ocurren a velocidades mayores a 40-50 [km/h]. En la Figura 5.1 se muestra una gráfica de un choque a 56[km/h] donde se ve la magnitud de la aceleración.

$$A_{cc} = \frac{V_i - V_f}{\Delta t} = \frac{12{,}5 - 0}{0{,}04} = 312{,}5[\frac{m}{s^2}] = 31{,}88g \tag{5.1}$$

Usando el promedio de ese rango y asumiento que la velocidad baja a 0 [km/h] en ese lapso de tiempo, se calcula que las aceleraciones [g] que experimentan los choques superan los 30g (ver ecuación 5.1 en donde la velocidad está en $\frac{m}{s}$). Un valor mucho mayor al del máximo medido por el acelerómetro a utilizar (16g). Sin embargo, este valor puede variar en la medición de un acelerómetro según la frecuencia de adquisición, oscilando entre 10g y 16g fácilmente en los momentos previos al choque, por lo que también es un factor a considerar al momento de configurar los umbrales.

La frecuencia de adquisición del dispositivo, es de 90Hz, es decir, los valores de aceleración

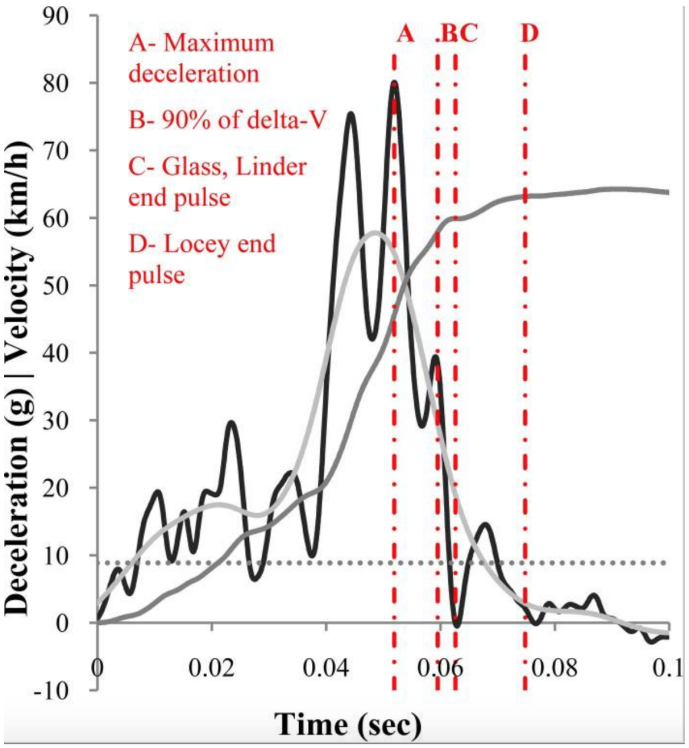


Figura 5.1: Choque frontal de automóvil a 56 [km/h] [30]

medidos son cada 0.0111[s]. Dado eso y la duración de los accidentes, el tamaño de las ventanas de búsqueda y detección se definen considerando este valor para evitar falsos positivos por errores de lectura. La ventana de búsqueda para entrar a la histéresis puede tener un ancho entre 6 y 12 (abarca 0,067 - 0,12 segundos) y la de detección, entre 3 y 6 (abarca 0,033 - 0,066 segundos), de esta forma el algoritmo asegura que cualquier peak de aceleración corto en duración, no será considerado como detección.

Dicho lo anterior, los parámetros quedaron configurados de la siguiente manera:

- Tamaño ventana de búsqueda: **7**
- Tamaño ventana de detección: **4**
- Umbral de entrada a histéresis: **12g**
- Umbral de salida de histéresis: **10g**
- Umbral de deteccion: **12g**
- Número de puntos mínimos para entrar/salir de histéresis: **5**
- Número de detecciones consecutivas mínimas para activar detección: **4**

En la Figura 5.2 se muestran los datos del módulo de aceleración en **g** multiplicados por 1000 de un test de choque a 48 [km/h] y de 0.1[s] de duración [33]. El algoritmo se aplicó a esta curva y obtuvo una detección de aceleración debido a que se cumplieron las condiciones: entrada a histéresis y detección más de 4 veces seguidas.

El código de de dicho gráfico está en MATLAB y se usó para estudiar el comportamiento del algoritmo en diversas curvas de datos.Naturalmente, este código no es compatible al de

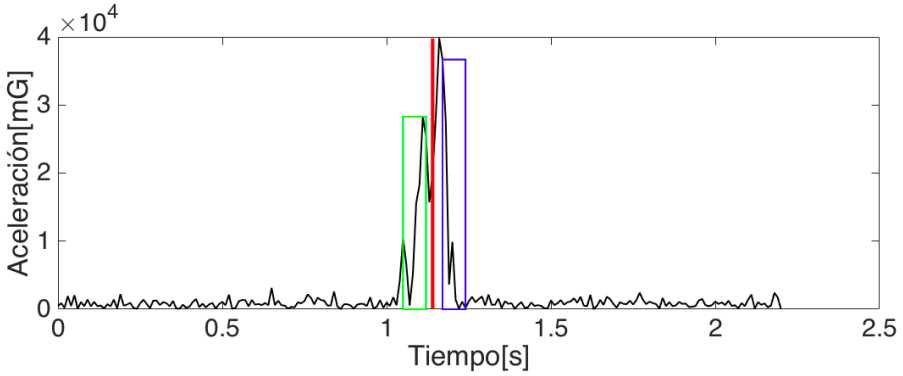


Figura 5.2: Choque detectado usando algoritmo y código MATLAB

Arduino y se programó uno para la placa también y que está en Anexo 8.3.2

5.2. Implementación del dispositivo

5.2.1. Software Arduino Pro Mini

Programar en Arduino es una tarea sencilla incluso para los que están comenzando, debido a que en su plataforma web está el programa de instalación (la interfaz de programación y compilación), guías tutoriales e incluso programas de ejemplo para compilarle a la placa desde el computador a través de un puerto USB.

Dado que en este caso, el Arduino a utilizar es el Pro Mini, no se tiene un puerto USB para compilar directamente a través del computador los programas a la placa. Es por esto, que fue necesario utilizar como puente un Arduino UNO para programar el Arduino Pro Mini[38]. Los pasos son los siguientes:

- Extraer cuidadosamente el Atmel IC del Arduino UNO (micro controlador). Esto para que al compilar un programa con el IDE, no se ejecute en la placa grande sino en el Arduino Pro Mini.
- Realizar la conexión entre ambas placas usando 5 pines:
  - Salida 5V de UNO hacia VCC de Pro Mini (Alimentación)
  - GND de UNO hacia GND de Pro Mini (Tierra)
  - RX de UNO hacia RXI de Pro Mini (Comunicación)
  - TX de UNO hacia TX0 de Pro Mini (Comunicación)
  - RESET de UNO hacia GRN de Pro Mini
- Conectar por USB el computador al Arduino UNO y cargar a través del IDE un programa de prueba para comprobar la compilación exitosa del Arduino Pro Mini. Previamente se debe configurar el IDE para programar en Arduino Pro Mini 3.3V 16MHz.

Esto no presentó mayores problemas debido a que ambas placas utilizan el mismo entorno y librerías de programación, por lo que seguir las instrucciones de conexión fue suficiente.

Como observación importante, está el hecho de que dependiendo del tipo de Arduino Pro Mini (3.3V o 5V) se debe conectar hacia la salida respectiva del Arduino UNO. En este caso, fue utilizada la de 3.3V. Y por otro lado, es necesario dejar a disposición estos pines para realizar futuras cargas de software cuando el sistema esté embebido completamente, por lo que se soldaron a la placa conectores hembra.

Otra opción era utilizar un cable TTL convertidor serial pero dado que en la empresa ya estaba a disposición un Arduino UNO, fue posible utilizarlo como puente de programación (además que realizar esta operación no deja inutilizable la placa).

5.2.2. Integración del FONA 2G

El primer componente a integrar es el módulo de comunicación, que como se describió en la sección de diseño, se utilizó un FONA 2G con sus accesorios correspondientes.

Afortunadamente, el sitio web de Adafruit [39] posee toda la documentación con las características del hardware y las instrucciones necesarias para conectar el módulo con sus componentes. De esta forma, el primer paso es conectar las antenas GPS y GSM al módulo, las cuales se ensamblan a presión en los pines GPS Ant y GSM Ant, respectivamente. También se conecta la batería de litio 3.7V en el conector JST, que es estrictamente necesaria para la utilización del módulo, aún cuando esté alimentado con una fuente externa (debido a que es la batería la que entrega los peaks de corriente necesarios en las llamadas).

Luego, con soldadura se conectan el parlante y el micrófono, en los pines SPKR y Mic, respecivamente. Y finalmente, se debe conectar el módulo con el controlador Arduino Pro Mini, el cual se conecta de la siguiente forma:

- Vio conecta 3.3V del Arduino (alimentación del FONA 2G)
- GND conecta a GND del Arduino (Tierra)
- Key conecta a GND del Arduino (Siempre encendido)
- RX conecta a pin digital 2 (Comunicación)
- TX conecta a pin digital 3 (Comunicación)
- RST conecta a pin digital 4

La Figura 5.3 muestra la conectividad del FONA 2G con el Arduino Pro Mini y los componentes.

Adicionalmente, el FONA 2G necesita una mini Sim para poder acceder a la red celular, en la Figura 5.4 se muestra el bloque posterior del FONA con la mini SIM card Virgin puesta en el slot.

Para poder utilizar el FONA 2G, Adafruit posee documentación que describe la conexión directamente al computador o también a través de la programación con Arduino. Como este es el caso, se utilizó la librería disponible para configurar y también probar todas las funciones del módulo [40].

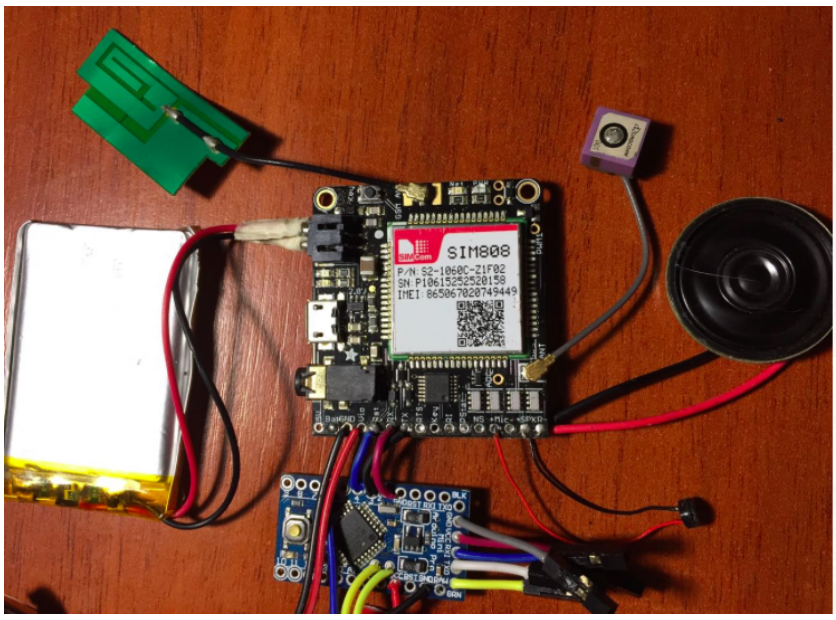


Figura 5.3: Conexión de componentes con el módulo FONA 2G: Arduino Pro Mini, batería, parlante, micrófono, antena celular y GPS

Figura 5.4: Vista trasera FONA 2G con una mini SIM card Virgin Mobile en el slot.

El FONA 2G funciona con un protocolo de comunicación serial en el cual Arduino debe escribir sobre el módulo comandos de texto en un formato **AT + COMMAND** para ejecutar una acción. Dado que esto es un poco tedioso, la librería disponible contempla un archivo .h y .cpp para traspasar todos estos comandos a funciones más legibles y fáciles de utilizar. Con esto, se generó el código en Arduino (Anexo 8.1) que a continuación resume en qué consiste:

1. Incluir librerías: Para poder comunciarse con el FONA desde el Arduino y utilizar las funciones del módulo, es necesario incluirlas una vez al comienzo del script.
2. Definir variables: Acá se fijan los pines que se utilizan para comunicarse con el FONA

(2,3 y 4). Además, se inicializan las variables globales del script, dentro de las cuales, la pricipal es la del objecto que representa al FONA y que llama a las funciones de la librería.

3. Iniciar programa: Al momento de iniciar la ejecución del programa en la placa Arduino, se intenta configurar el Baud Rate de comunicación serial con el Fona, que en este caso es 4800. Si esta comunicación no es exitosa, el programa queda en un loop intentándolo continuamente.
4. Configurar FONA: Una vez detectado el FONA, hay que configurar ciertas funcionalidades. Estas son, fijar la salida de audio por parlante, fijar los volúmenes al máximo del parlante y del micrófono, activar el GPS, fijar el número telefónico al que se le enviará el SMS y se le establecerá la llamada y finalmente, fijar la URL al que se le enviará el Post Request. Esta última es **http://soymomosupport.herokuapp.com/memoria** y es una página web de la empresa y adaptada para este trabajo, que en la sección 5.3 se explica su uso.
5. Loop: En esta parte, se escribe el algoritmo central que ejecuta continuamente el dispositivo mientras está encendido. En esta parte ocurre lo siguiente:
   - Lectura GPS: Aquí se hace la lectura de GPS obteniendo las coordenadas de localización. En caso de que la señal GPS no sea buena, el FONA 2G puede encender el GPRS para obtener una ubicación estimada utilizando triangulación con torres celulares. Si aún así no se pueden obtener coordenadas, el algoritmo almacena un contador para saber si los datos de GPS son muy antiguos.
   - Envío de SMS: Se forma un mensaje de texto según el tipo de emergencia, es decir, si fue detectado un choque o si se oprimió el botón de pánico del dispositivo (descripción de implementación más adelante). Este mensaje tiene asociado las últimas coordenadas legibles del GPS y que, en caso de ser muy antiguas, lo dice a través del mensaje.
   - Envío de POST a URL: Se forma un mensaje en formato JSON para adjuntarlo al POST HTTP. Este JSON posee sólamente 3 atributos: **latitude**, **longitude** y **type**, que representan las coordenadas GPS y el tipo de emergencia realizado.
   - LLamada telefónica: Finalmente, se intenta la comunicación telefónica con el número configurado en el punto 4. Para este prototipo, se utilizó el número telefónico del alumno.

5.2.3. Integración del acelerómetro

El segundo componente a integrar en el sistema es el MPU9250, el cual posee un sistema de coordenadas definido y mide aceleraciones (incluyendo la gravitacional) respecto a ese sistema. Además, también posee un giróscopo interno que mide los ángulos de Euler, que serán muy útiles para restar la componente de la gravedad en la medición de aceleraciones (ver Sección 5.4.1).

Estas lecturas de datos las debe realizar el Arduino Pro mini, cuya conexión se realiza usando el protocolo I^2C y con alimentación de 3.3V como se muestra en la Figura 5.5

Una característica esencial de este componente, es que puede configurar a través de código

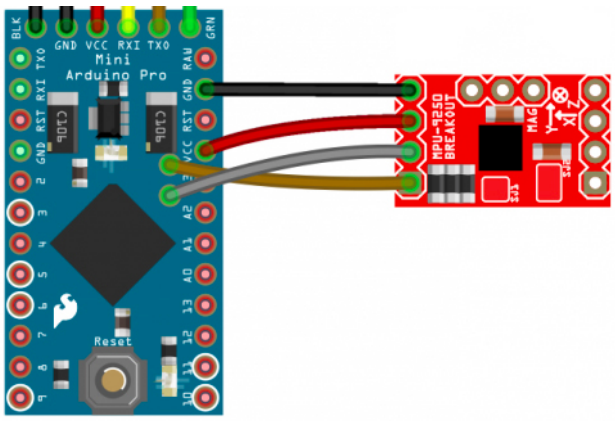


Figura 5.5: Montaje de conexión entre Arduino Pro-Mini y MPU-2950.Fuente: [https://www.sparkfun.com/]

diferentes rangos de medición de aceleración y velocidad angular. Para este trabajo, se utiliza la más alta debido a que se están midiendo aceleraciones altas y se pretende detectar desaceleraciones rápidas como ocurre durante un choque, es decir, un rango $\pm 16g$ que permitirá discernir de mejor forma entre detectar choques o frenos bruscos dado que es difícil saturar el acelerómetro hasta ese rango. Por otro lado, se configura el componente a medir velocidades ángulares hasta $\pm 250DPS$ (grados por segundo) dado que en este trabajo el auto no debería experimentar giros tan rápidos y de esta forma tendrá mayor precisión para medir los ángulos de Euler que el componente esté experimentando.

La librería de Kris Winer [41] para Arduino implementa la configuración de rangos del sensor y la lectura de datos en tiempo real, así como la transformación usando quaterniones para obtener los ángulos de Euler respecto a los ejes definidos en la placa, es decir, todo lo necesario para que el controlador Arduino obtenga los datos en forma continua siguiendo el sistema de referencia fijado en el componente.

Como se mencionó en la sección 2.1.2, la detección de choques sólo toma en cuenta la magnitud de la aceleración como característica importante para desarrollar el algoritmo. Sin embargo, la obtención de los ángulos de Euler y saber cómo evoluciona la aceleración en cada eje, permitirá desarrollar la simulación del choque a posteriori, por lo que es importante establecer este sistema de referencia.

Afortunadamente, la elección de cómo fijar el acelerómetro dentro del dispositivo no es relevante, pues los datos se leen respecto al sistema de referencia propio del MPU9250 y sólo se debe elegir el sistema que parezca más cómodo de trabajar. Para optimizar espacio, se elige ubicarlo verticalmente, quedando el eje X apuntando hacia adelante, el eje Y hacia arriba y el eje Z hacia la derecha dentro del automóvil, como se muestra en la Figura 5.6 siempre y cuando la entrada de la cigarrera sea horizontal.

La adaptación del código de Kris Winer para configurar el componente y leer los datos en tiempo real a una frecuencia a elección, se encuentra en Anexo 8.3.1.

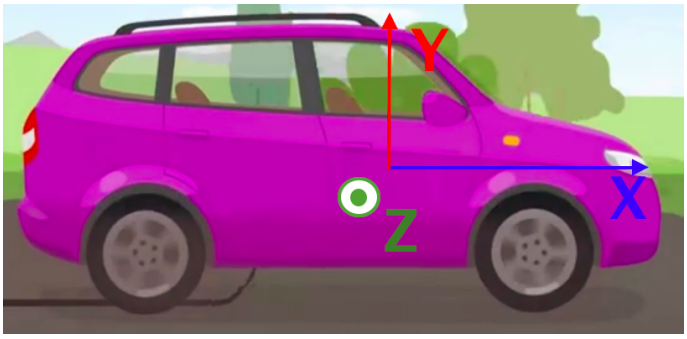


Figura 5.6: Sistema de referencia para aceleración.

5.2.4. Integración de la tarjeta SD

La conexión entre el Arduino Pro-Mini y la tarjeta Micro SD Transfer Breakout se realizó siguiendo las instrucciones de la guía de la empresa fabricante Sparkfun [42], alimentado con 3.3V el componente y utilizando los pines de conexión 8,11,12 y 13. La Figura 5.7 muestra le esquema de conexión.

La programación utiliza las librerías nativas de Arduino **SD** y **SPI** para leer y escribir datos de un archivo de la tarjeta micro SD. En Anexo 8.4 se encuentra un código de prueba adaptado desde la guía del componente, para escribir datos frame a frame en un archivo de extensión **.txt**.

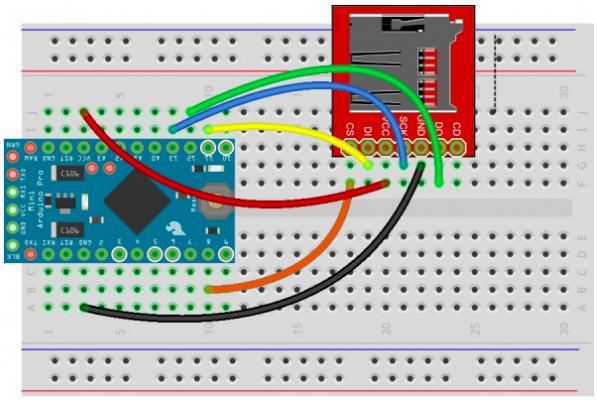


Figura 5.7: Conexión Arduino-Pro Mini 3.3V con Micro SD Transfer Breakout. Fuente: [https://www.sparkfun.com/]

Si bien el código de Anexo 8.4 funciona perfectamente, generándose el archivo **.txt** con la información en el interior. La integración de este con el código de Anexo 8.3 (lectura MPU2950) no funcionó correctamente, debido a interferencias en la comunicación entre ambos componentes con el Arduino, que producían una interrupción de la escritura de datos.

La unica solución encontrada fue bajar la tasa de lectura del sensor para disminuir esta interferencia. Sin embargo, esta disminución resulta ser demasiado baja (3-5Hz) para este trabajo donde se requiere una frecuencia de lectura mayor a 15Hz para desarrollar una buena simulación de los momentos previos al choque y aún mayor (sobre 20Hz) para implementar

un algoritmo de detección de buen desempeño (debido a que los accidentes son de muy corta duración).

Es por esto, que este componente quedó descartado del circuito que forma el prototipo y las pruebas reales a realizar para la validación del sistema se dividirán en 2 partes: la detección se realizará utilizando el dispositivo y la simulación del accidente, recolectando datos de viajes utilizando una versión kit del dispositivo sin el FONA y conectado al computador, este esquema se muestra en la Figura 5.8.

Figura 5.8: Esquema de la versión kit del dispositivo para leer datos del MPU2950 con Arduino UNO y conexión al computador.

5.2.5. Alimentación del dispositivo

La integración de un Plug-in de cigarrera común y corriente es sencillo, pues sólo se deben conectar los componentes al pin del voltaje de salida y la Tierra. Sin embargo, estos Plug-in poseen una salida de 5V, que es más de lo que permite el acelerómetro, el Arduino Pro-Mini a utilizar y la tarjeta SD.

Es por esto, que es necesario bajar esta voltaje a 3.3V para poder alimentar todo el circuito sin riesgo de que los componentes se dañen por exceso de voltaje. Según [43], la mejor solución es utilizar un regulador de voltaje LM7833 que conecta la salida 5V y la transforma en 3.3V sin riesgo de elevación de temperatura debido a que la conversión es baja. Además, este componente es de bajo costo y pequeño, por lo que embeberlo en el sistema no implica mayor problema ni riesgo. En la Figura 5.9 se muestra este componente, donde el pin izquierdo corresponde a la entrada 5V, el del medio a la Tierra y el de la derecha como la parte metálica superior, corresponden a la salida de 3.3V donde se deben conectar

los componentes a alimentar.

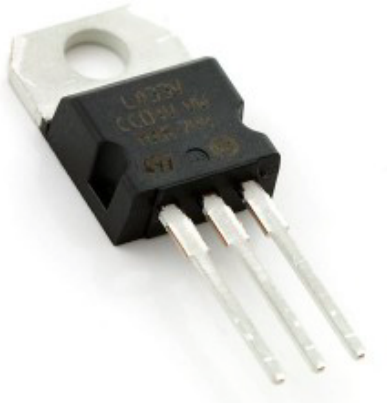


Figura 5.9: Regulador de voltaje a 3.3V LM7833.

5.2.6. Prototipo embebido

Para diseñar la carcasa junto al diseñador industrial, se tomaron las medidas de cada uno de los componentes a utilizar para fabricar las placas con los encajes y espacio necesarios para que todo el circuito quede embebido correctamente sin peligro de daños.

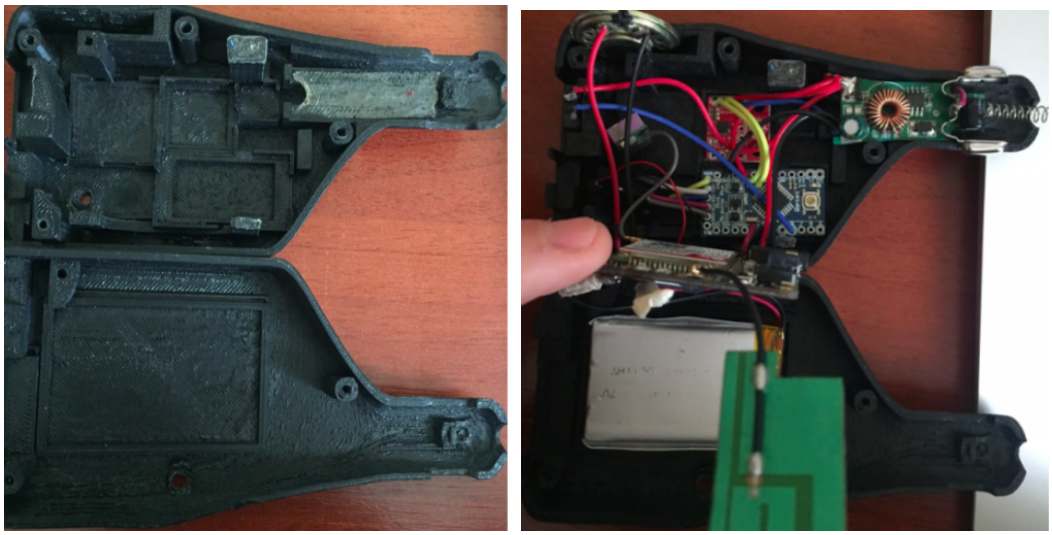
Si bien fue una tarea remunerada del profesional, el modelar tridimensionalmente la carcasa, cotizar el material y la empresa donde realizar la impresión. El diseño interior fue una tarea en conjunto en el que se complementaron las ideas de optimizar espacio como las de factibilidad técnica.

Como observación, se tiene que al momento de diseñar esta carcasa, no se tenía en consideración la utilización de un regulador de voltaje y sí de una tarjeta SD. Sin embargo, debido a los problemas presentados durante la implementación, el prototipo final sí posee un regulador de voltaje pero no utiliza tarjeta SD, por lo que el espacio de este último se utilizó como encaje del primero sin mayor problema.

De esta forma, el resultado final de la carcasa se muestra en la Figura 5.10, donde se tiene la vista interior sin componentes y con componentes. A continuación, se describen los detalles que se tuvieron en consideración:

- Placa de 2 partes para una mayor comodidad al momento de la integración y visualización de conexiones.
- Encajes para el plug-in de alimentación, placa Arduino Pro-Mini, MPU9250, Micro SD, micrófono y antena GPS en la primera capa. Luego el módulo FONA y su antena GSM,en la segunda y finalmente, la batería en la tercera capa.
- Espacio entre capas de 1 cm para evitar contacto y también una mejor disipación de calor.
- Uso de pegamento para un anclaje firme de las placas de acelerómetro en el eje de coordenadas descrito en la Figura 5.6.

- Uso de soportes para fijar el módulo FONA y así separar la primera de la segunda capa sin posibilidad de contacto.
- Encaje de batería en la otra parte de la placa para mayor comodidad al abrir el prototipo.
- Espacio suficiente en la parte frontal para anclar un botón digital con un soporte por detrás de una trozo plástico y que tenga sensibilidad al oprimir el botón.
- Agujero para anclar micrófono y no quede obstruído hacia el exterior para un mejor alcance de la voz.
- Agujeros en la parte superior para que el parlante no quede obstruído y así evitar atenuar el volumen en la llamada.
- Espacio en la parte frontal suficiente para anclar puertos USB en un trabajo futuro.
- Diseño de para encaje de 4 tornillos a presión, para poder sellar las placas.



(a) Sin componentes (b) Con componentes

Figura 5.10: Vista interior de la carcasa del prototipo.

Una vez embebido, el resultado del dispositivo se muestra en la Figura 5.11 y sus medidas finales son:

- Alto: 6,2cm
- Ancho: 3,3cm
- Largo zona de encaje a plug-in: 3,7cm
- Largo total: 12,8cm

5.3. Sistema de atención de emergencia

Como se mostró en la Figura 4.1, la arquitectura del sistema contempla 3 agentes posterior a la detección del choque, estos son los contactos del conductor, la central de emergencia y el peritaje del accidente.

(a) Frontal (b) General

Figura 5.11: Vistas del dispositivo electrónico terminado.

Para el primer agente, se tienen los SMS alertando de cuándo de ocurre un accidente y que estos recurran rápidamente a ayuda médica para el accidentado. Si bien este medio también sirve para una central médica, este canal no es escalable debido a que es personalizado, por lo que existe otro medio más eficaz para manejar información en cas de muchos accidentes en poco tiempo, la Internet.

Es por esto, que en el diseño del sistema de este trabajo, se estipuló el envío de notificaciones POST Request a una base de datos y que las oficinas de atención de emergencias vean a través de una plataforma web, todas las emergencias que van llegando en tiempo real y poder actuar rápidamente acorde a los tiempos, gravedad y ubicaciones de cada uno.

5.3.1. Base de datos: Back4App

La empresa SOSmart Labs aloja muchos de sus proyectos en Back4app [44], una base de datos confiable y con una vasta y documentada API que integra muchos lenguajes de programación. Se decidió utilizar esta base de datos por 2 razones principales:

1. La empresa lo pidió como requisito debido a la centralización de toda su información en un solo proveedor de almacenamiento de información.
2. Posee una API fácil de usar para conectar a la plataforma de monitoreo de accidentes (ver Sección 5.3.2)

Desafortunadamente, Arduino y el FONA no poseen librerías para acceder directamente a la información de esta base de datos, ni tampoco escribir sobre ella. Pues se deben anclar las llaves de acceso de la aplicación y esto no es posible en un POST Request. Es por esto, que se decidió utilizar un servidor web adicional, al que le llegarán estos POST Request y este se encargará de guardar la información en Back4App.

5.3.2. Server web: Recibir POST Request

Como se adelantó en la sección 5.2.2, esta página web a la que llegarán los POST Request es **http://soymomosupport.herokuapp.com/memoria**, que es una página web creada por el alumno para otros fines de la empresa y que en este caso, se adaptó una parte de su código para integrar al sistema.

Esta página web fue desarrollada en Ruby on Rails [45] y posee librerías que permiten conectarse a la API de Back4App fácilmente y poder leer y escribir en la base de datos. El código de la adaptación está en Anexo 8.4.

Una forma de probar el funcionamiento de este server web, sin modificar la base de datos, es enviar un POST Request manualmente desde una terminal unix con el siguiente código.

```
curl -v H "Accept: application/json" -H "Content-type: application/json" -X
  POST -d ' {"latitude":"-33.42", "longitude":"-70.4122", "type":"
  emergencyButton"}' http://soymomosupport.herokuapp.com/memoria
```

5.3.3. Dashboard de monitoreo

La plataforma web de monitoreo de accidentes también se desarrolló en Ruby on Rails y fue creada por el alumno para monitorear accidentes detectados por la aplicación Sosmart de la empresa. Dado que esta plataforma también se conecta con la base de datos de Back4App, la utilización de este dashboard para el proyecto es directo. La Figura 5.12 muestra la interfaz para el usuario y que tiene muchas funcionalidades (sistema de ingreso privado, validar fichas médicas y enviar emails) que permiten distinguir choques o emergencias por botones de pánico, para así agilizar los protocolos de rescate en accidentes. El código de esta plataforma se reserva pues esta plataforma escapa del enfoque principal de esta memoria.

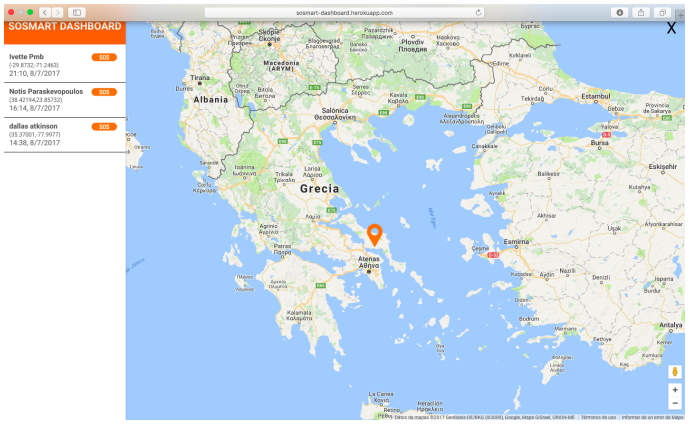


Figura 5.12: Dashboard para monitoreo de accidentes. **http://sosmart-dashboard.herokuapp.com/**

Así, la arquitectura final del sistema de atención de emergencias se muestra en la Figura 5.13.

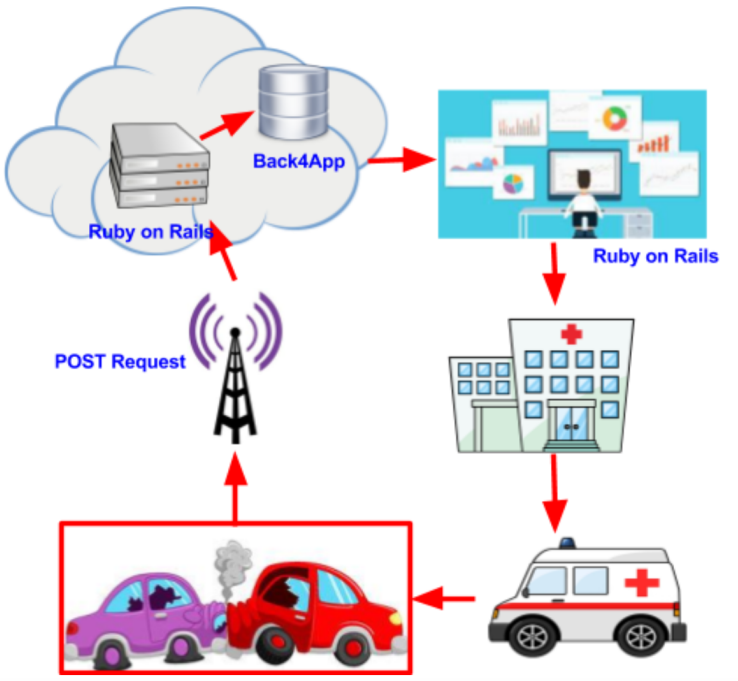


Figura 5.13: Arquitectura sistema de emergencia de accidentes.

5.4. Post-procesamiento

Los datos obtenidos por el MPU9250 que lee el Arduino son válidos para el algoritmo de detección de choques pues este sólo necesita el módulo de la aceleración. Sin embargo, para simular el choque, es necesario procesarlos debido a que tienen mucho ruido asociado y principalmente, porque esos datos incluyen la aceleración por gravedad, por lo que el componente está continuamente leyendo una aceleración incluso cuando está en reposo.

5.4.1. Procesamiento de datos de acelerómetro y giróscopo

Para realizar este procesamiento y limpiar los datos obtenidos desde el MPU9250, se utilizó el programa computacional MATLAB (ver código completo en Anexo 8.5) y se tomaron algunas técnicas de procesamiento realizado en [46]. A continuación, se describen los pasos que componen este procesamiento y la razón de su aplicación.

Agregar histéresis para ángulos límite

El MPU9250 mide los ángulos de Euler en un rango entre -180 y 180, para cubrir todo el plano según la dirección de inclinación del eje. Por lo tanto, en la práctica y debido al ruido, ocurre que en esos límites, el sensor puede pasar continuamente de valores positivos a valores negativos y viceversa, a pesar que el cambio de ángulo es leve, imposibilitando el uso de filtros pasabajos que usan la media como medida de atenuación de ruido. Este fenómeno

se aprecia en la Figura 5.14 con color azul.

Para combatir este fenómeno, se utilizó un pequeño algoritmo que aplica una histéresis en la zona, es decir, para ángulos superiores a 175º y menores a -175º, se utilizará el valor que indica el sensor pero con el mismo signo del ángulo previo a la entrada en la zona. De esta forma, no ocurrirán cambios de signo continuamente en esa zona y aunque se pierden 5 grados de precisión, no deberían afectar en rotaciones significativas.

La salida de la histéresis ocurre cuando los ángulos son menores a 170º o mayores a -170º, caso en el que la lectura de los datos comienza a ser normal. En la Figura 5.14 con color rojo, se muestra el resultado despúes de aplicar este procesamiento inicial.

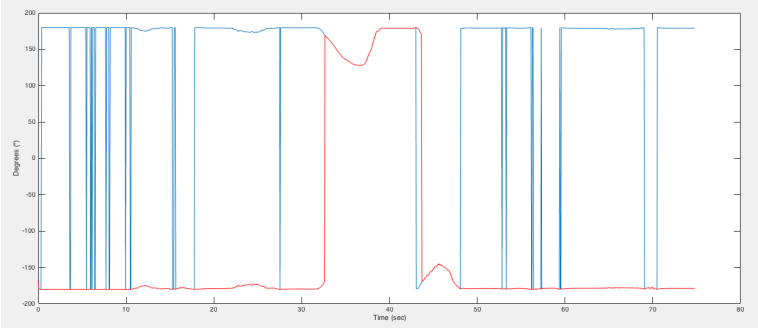


Figura 5.14: Lectura de ángulo roll con acelerómetro en reposo y eje Z apuntando hacia abajo. En azul, la señal directa y en rojo, después de la aplicación de histéresis.

Calcular offset DC inicial de las señales

La lectura de los datos desde el MPU9250 vienen con una calibración interna configurada por software, según la posición y orientación que tenga el componentes cuando se inicie su lectura. A pesar de esto, la lectura inicial también posee un ruido que se denomina DC, debido a que es una constante que afecta las mediciones.

La forma de calcularlo para cada eje de medición, es obtener un conjunto de datos del componente en reposo y fijado con un sistema de referencia conocido. De esta forma, el offset de cada señal vendrá dado por la media de ese conjunto de datos.

En la Figura 5.15, se muestra la orientación en la que se dejó el componente para medir datos, con Z apuntando hacia abajo para restar fácilmente la aceleración dada por la gravedad y obtener los offset de cada componente de aceleración. Por otro lado, el offset de los ángulos viene dado según la posición inicial en que se encuentre el dispositivo, por lo que no es un valor fijo y para calcularlo se toman los primeros 200 datos de ángulos y la media de este conjunto, se toma como origen. Finalmente, el offset de cada aceleración es:

- Ax: $-0{,}59$
- Ay: $0{,}6932$
- Az: $-3{,}9155$

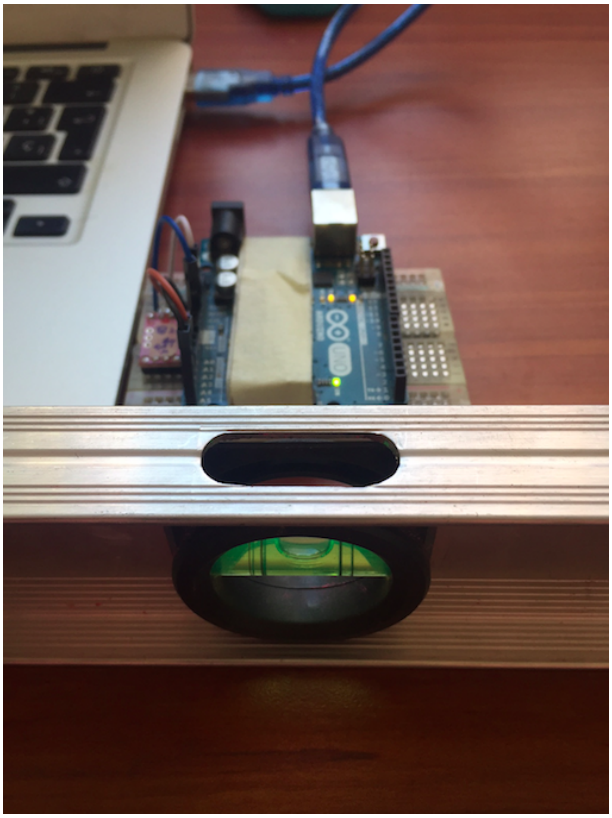


Figura 5.15: Obtención de offset DC de aceleraciones y ángulos.

Eliminar offset DC y evitar overflow de ángulos

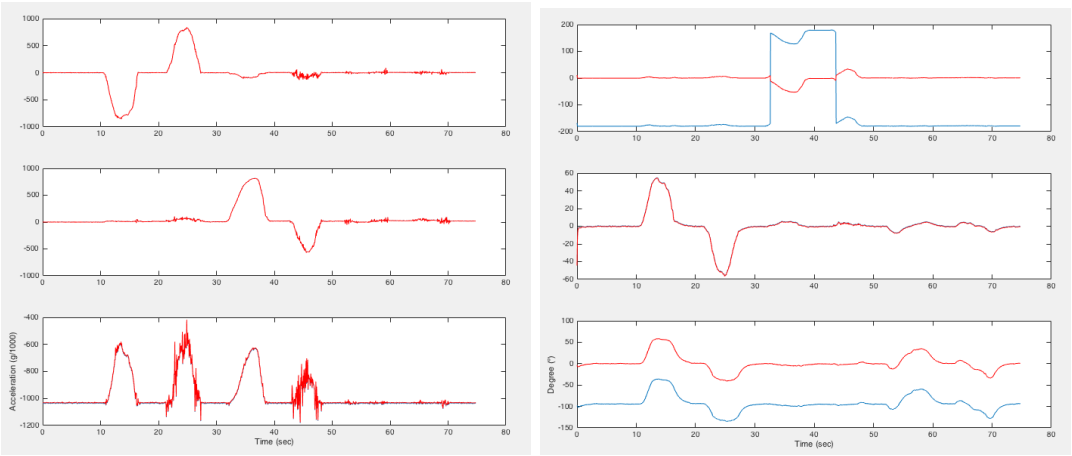
Para eliminar el offset, simplemente se debe tomar los valores obtenidos anteriormente y restárselos a cada lectura nueva del MPU9250. De esta forma, se obtienen datos más limpios y centrados como los que muestra la Figura 5.16, en el que tanto aceleración como ángulos están centrados en 0 debido a que el comoponente estaba en reposo en un comienzo (salvo el eje Z que aún tiene la componente de aceleración por gravedad).

Notar que para la aceleración, el offset no es muy significativo a simple vista, pero después de realizar las integraciones en los pasos que siguen, si se observa un cambio importante.

Además, debido a esta resta, puede ocurrir que algunos ángulos se salgan del rango original y se muestren valores por sobre 180º, es por esto que se aplica un algoritmo que en caso de ser mayor a 180º, se le reste 360º para que quede en el rango pero indicando la misma posición. La Figura 5.16b en rojo, muestra el resultado después de eliminar el offset y el overflow.

Filtro pasa bajos: atenuación de ruido

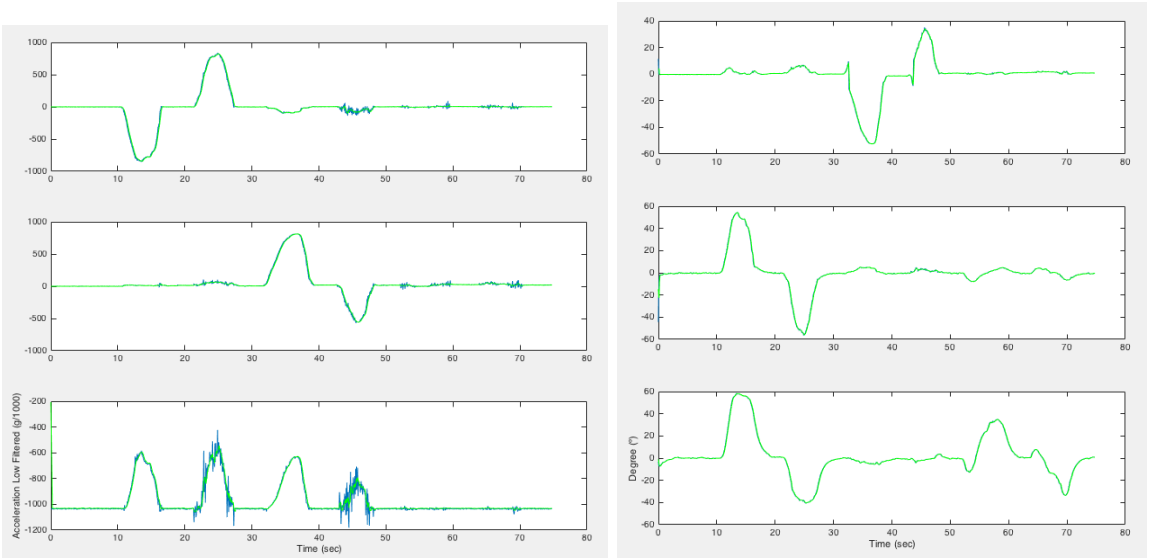
El siguiente paso es usar un filtro pasa bajos para suavizar la curva de la señal. Este proceso funciona con una pequeña ventana de largo parametrizable que va recorriendo la señal y tomando las medias para recontruir otra señal a partir de esas medias. En este caso,



(a) Aceleración en sus 3 ejes (b) Ángulos en sus 3 ejes

Figura 5.16: Señales con offset (azul) y sin offset ni overflow (rojo)

se usó una ventana de largo 3 para los ángulos y de largo 5 para las aceleraciones. La Figura 5.17 muestra en azul las señales antes del proceso y en rojo, el resultado después del filtro.



(a) Aceleración en sus 3 ejes (b) Ángulos en sus 3 ejes

Figura 5.17: Señales sin filtro pasabajo (azul) y con filtro pasabajo (rojo)

Eliminar componente por gravedad

El siguiente paso del procesamiento, es eliminar de las aceleraciones en los 3 ejes, la componente dotada por la gravedad (1g), debido a que para simular el accidente se necesita el desplazamiento que sufre el automóvil frame a frame y si la integración que obtiene este desplazamiento toma en cuenta la gravedad, los resultados arrojarán que el auto está en caída libre constante.

El cálculo de esta componente para cada eje viene dado por la ecuación 5.2. Finalmente, la aceleración final será la resta entre la original y esta componente de gravedad en cada eje y dado que hasta el momento las unidades están en $\frac{g}{1000}$, se debe dividir por 1000 y multiplicar por 9.8 (1g) para obtener la aceleración en $\frac{m}{s^2}$

$$Ax_g = sin(pitch * \frac{\pi}{180}) \tag{5.2}$$

$$Ay_g = sin(roll * \frac{\pi}{180})$$

$$Az_g = cos(roll * \frac{\pi}{180}) + cos(pitch * \frac{\pi}{180}) - 1$$

La Figura 5.18 muestra el resutado de las componentes de aceleración sin considerar la gravedad, luego de aplicar nuevamente un filtro pasabajo para reducir ruido.

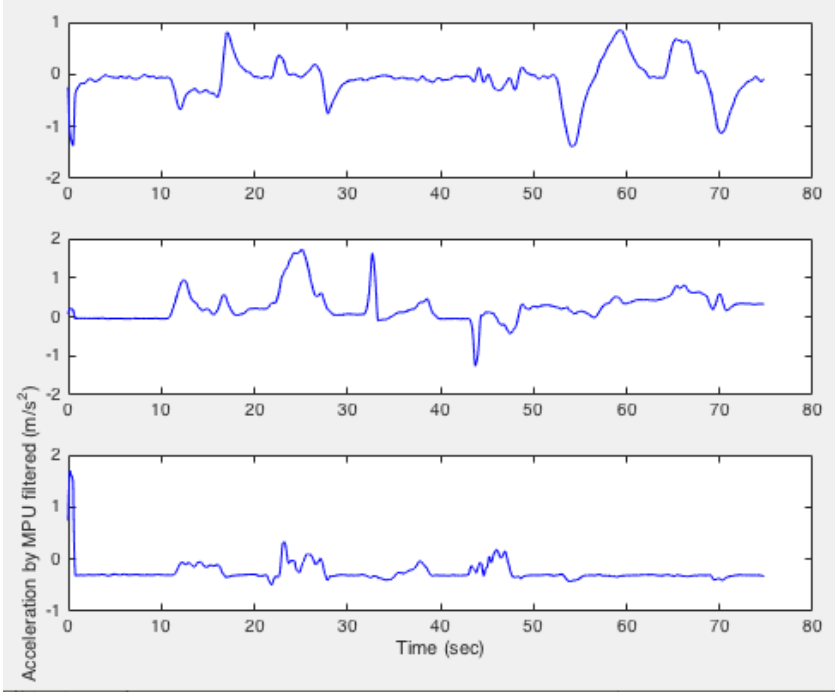


Figura 5.18: Aceleración sin componente de gravedad

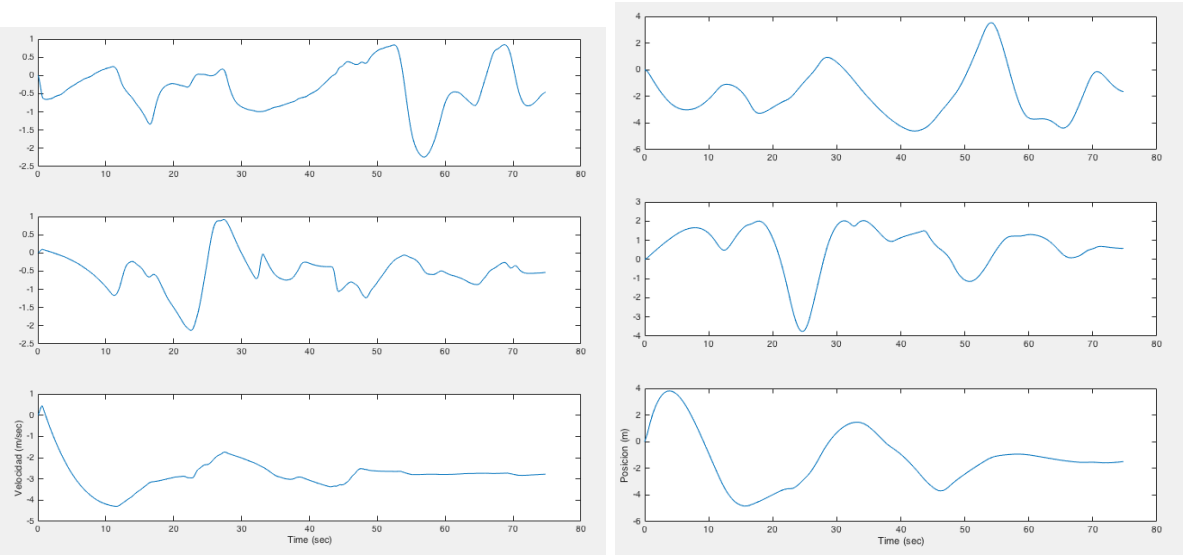
Filtro pasa-alto

La integración de la aceleración se realiza usando la integración trapezoidal y para esto, se deben evitar las componentes DC de la señal para que no se transformen en pendientes ni después, al cabo de la segunda integración, en exponenciales. Por lo que para ello, se aplica un filtro pasa alto entre ambas integraciones.

El filtro pasa alto a utilizar es el Butterworth con una frecuencia de corte de 30 Hz y orden 6.

Integración

Después de aplicar el filtro pasa alto, se realiza la integración trapezoidal sobre cada eje de aceleración, de esta forma se obtiene la velocidad experimentada a través del tiempo (Ver Figura 5.19a). Luego, se vuelve a aplicar el filtro pasa alto antes de una nueva integración, esta vez para obtener el desplazamiento experimentado por el componente a través del tiempo (Ver Figura 5.19b)



(a) Velocidad en sus 3 ejes (b) Desplazamiento en sus 3 ejes

Figura 5.19: Señales de velocidad y desplazamiento después de la primera y segunda integración de la aceleración, respectivamente.

Teniendo el desplazamiento en cada momento y los ángulos, es posible obtener una simulación frame a frame de cómo se movió el objeto. Finalmente, estos datos se guardan en otro archivo **.txt**.

5.4.2. Simulación del choque

Esta parte se realizó en el software de modelación 3D Unity. El cual permite manipular objetos y animaciones de múltiples formas. Pero en este caso, sólo basta con ejecutar un script que traslade el objeto en la dirección dada por los desplazamientos obtenidos del procesamiento y también las rotaciones experimentadas frame a frame.

El primer paso, es colocar un objeto auto en el espacio y con un sistema de referencia por defecto, En la Figura 5.20 se muestra un Asset de automóvil descargado de la nube Unity y el sistema de referencia definido.

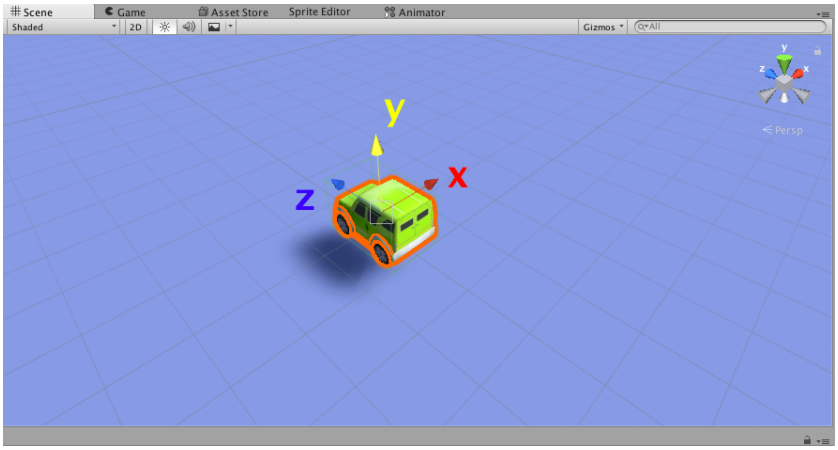


Figura 5.20: Aceleración sin componente de gravedad

De esta forma, el código que ejecuta (Ver Anexo 8.6) el programa lee línea por línea a una frecuencia configurada (la misma que el de la lectura) las coordenadas de desplazamiento y rotación para aplicarlas a los ejes respectivos del espacio de Unity (XYZ ->YZX y roll,pitch,yaw ->pitch,-yaw,roll) y de esta forma mover y rotar el objeto en el tiempo.

Capítulo 6

Resultados y Análisis

En esta sección se muestran las diferentes pruebas realizadas para evaluar el funcionamiento del sistema: detección, notificación y post-procesamiento.

6.1. Detección

Para evaluar la detección, se usaron 3 tipos de datos:

1. Pruebas de choques a baja escala
2. Pruebas de manejo real.
3. Curvas reales de choques de una base de datos

Dado que no se pudo integrar la tarjeta SD para almacenar datos, las pruebas 1 y 2 se realizaron montando el acelerómetro con un Arduino UNO y monitorear a través de la consola de Arduino las aceleraciones medidas. La Figura 5.8 muestra el esquema utilizado.

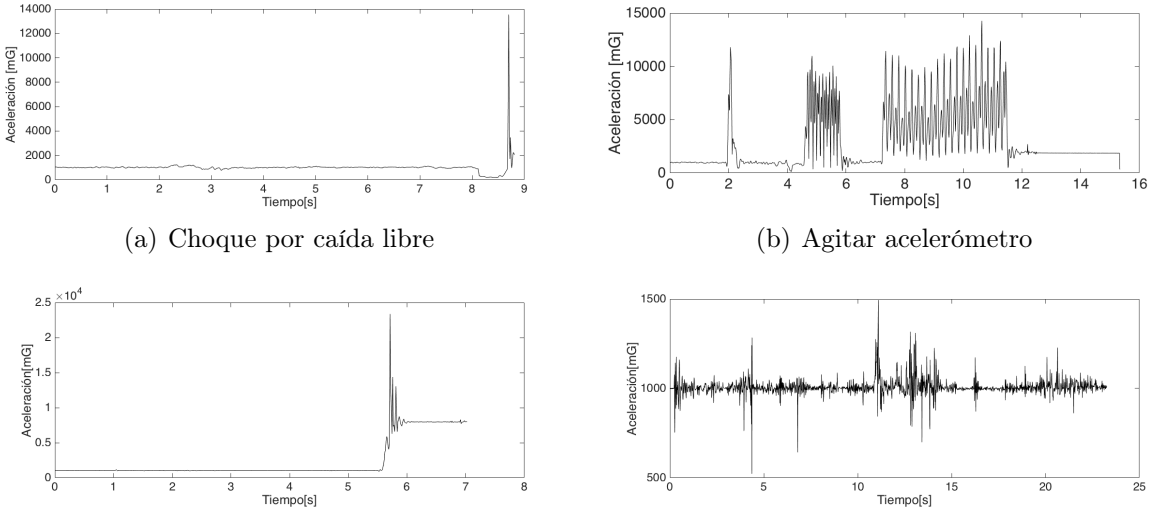
6.1.1. Pruebas de choques a baja escala

Se realizaron diferentes movimientos del acelerómetro (versión kit) mostrando los datos en consola y posterior evaluación del algoritmo en MATLAB. Se analizaron 15 pruebas divididas en:

- Caída libre contra el piso a diferentes alturas.
- Giros y frenos del acelerómetro
- Agitar el acelerómetro rápidamente
- Choques contra paredes y obstáculos usando una patineta como vehículo.

Con la configuración descrita en la sección 5.1, los resultados arrojaron que ninguno de los casos medidos, resultó en detección de choque. La Figura 6.1 muestra los gráficos resultantes

de la lectura de datos de alguna de las pruebas.



(a) Choque por caída libre

(b) Agitar acelerómetro

(c) Choque frontal

(d) Giros y frenos

Figura 6.1: Pruebas de choque a baja escala.

Los resultados fueron esperados ya que las pruebas realizadas no son comparables a choques reales y como se aprecia en los gráficos, el acelerómetro alcanza valores altos en su medición pero son peaks con muy baja duración por lo que el algoritmo los considera como simples *outliers* en los datos.

El caso más crítico es el de la Figura 6.1b en el que se miden aceleraciones muy altas seguidas de muy bajas por milisegundos (agitar fuerte el acelerómetro), pero dada la forma del algoritmo (uso de histéresis y detecciones consecutivas), tampoco hubo detección de choque. Aún cuando este último caso es difícil que ocurra con el dispositivo real debido a que al estar conectado al vehículo, este tipo de movimientos no deberían ocurrir.

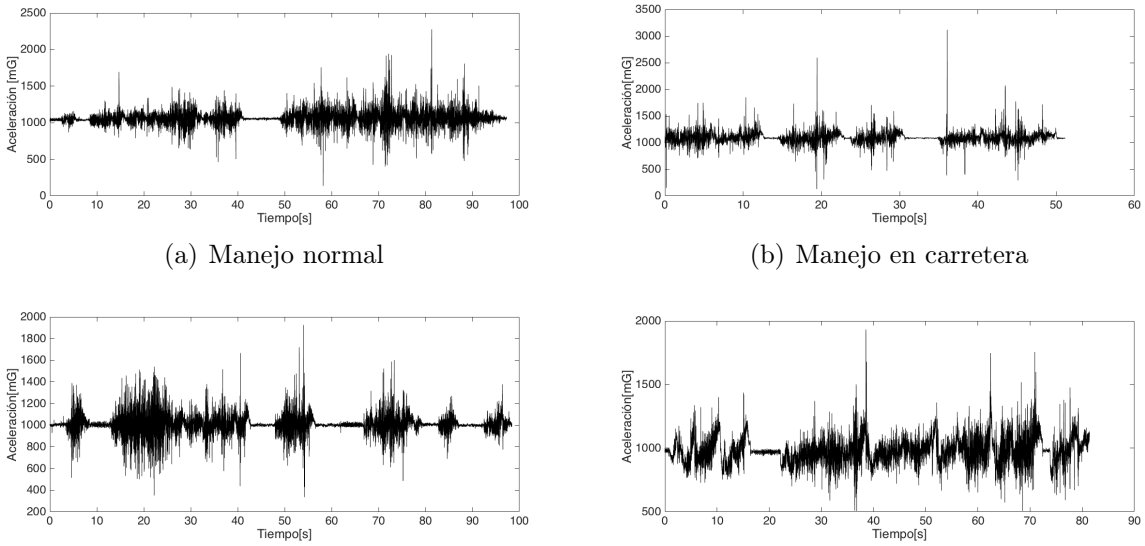
6.1.2. Pruebas de manejo real

Se realizaron diferentes viajes en automóvil utilizando el dispositivo conectado y en ningún caso se detectó algún choque. Naturalmente, estas pruebas se hicieron en calles y carreteras de la ciudad de Santiago y no se experimentó ningún accidente por razones obvias).

Sin embargo, para poder hacer un estudio más detallado de los datos, se utilizó el mismo kit del acelerómetro de las pruebas anteriores y se fijó al tablero del automóvil para realizar rutas por la ciudad mientras se guardaban los datos en un computador.

Después de correr el algoritmo sobre los datos, tampoco se obtuvo detección de choque y algunos de los gráficos se muestran en la Figura 6.2.

Los resultados de esta prueba fueron los esperados, debido a que al no haber un choque real, las aceleraciones experimentadas por el dispositivo oscilan entre los 1-2g y con peaks que no superan los 3g. Por lo que el algoritmo no detecta choques en ninguno de los casos.



(a) Manejo normal

(b) Manejo en carretera

(c) Manejo con muchos virajes

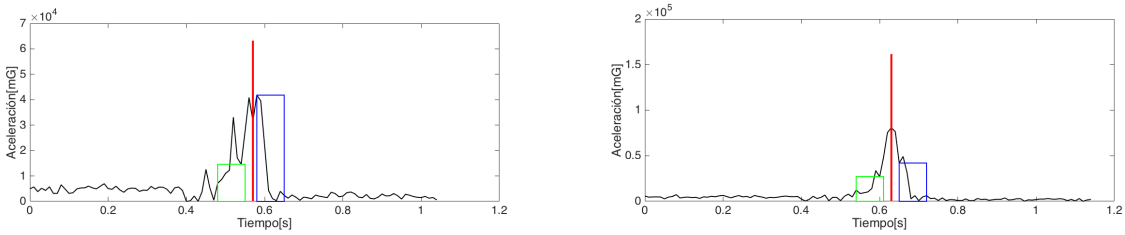
(d) Manejo con otra posición de acelerómetro

Figura 6.2: Pruebas en vehículo por calles de Santiago.

6.1.3. Curvas reales de choques de una base de datos

Para esta prueba, la empresa facilitó unos datos obtenidos de la NHTSA en el que se muestran curvas de aceleración en cada eje del acelerómetro durante un choque fuerte (superior a 50 [km/h]).

Alrededor de 20 curvas fueron probadas por el algoritmo y en todas se pudo detectar el accidente, debido a que en promedio, la aceleración medida durante el choque superaba fácilmente los 50g en cada una de ellas. En la Figura 6.3 se observan dos de estas pruebas.



(a) Choque frontal a 50km/h

(b) Choque frontal a 60km/h

Figura 6.3: Pruebas de choque de la NHTSA.

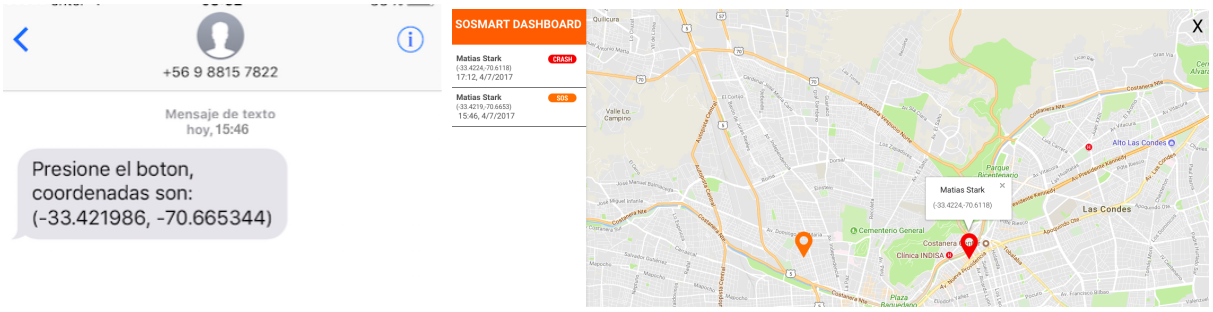
Por tanto, el desempeño del algoritmo para choques fuertes es muy bueno, debido a que de las 15 curvas probadas, en todas se detectó el momento del choque. Sin embargo, no es posible afirmar un desempeño perfecto de este algoritmo, debido a que las pruebas realizadas fueron en choques sobre los 50[km/h] y con un acelerómetro de mayor calidad y rango que el utilizado en este trabajo. Además, el desempeño del algoritmo no pudo ser evaluado en choques de menor intensidad (no se encontraron datos) por lo que quedó como trabajo futuro definir el desempeño en esos casos.

6.2. Notificación

Debido a que no se pudo recrear un choque con el dispositivo funcionando dentro de un automóvil, se tuvo que reconfigurar los parámetros del algoritmo para que la detección fuese con una desaceleración mucho menor (3g) para que así, el dispositivo mandara la notificación SMS, POST Request y posteriormente estableciera la llamada.

Por otro lado, presionando el botón de pánico del dispositivo una vez anclado al vehículo, se envió el mensaje que muestra la Figura 6.4a y posteriormente se ejecutó la llamada telefónica.

Los POST request de ambas pruebas se enviaron de forma exitosa mostrándose en el dashboard pocos momentos después, ver en Figura 6.4b. Las direcciones corresponden a los lugares de prueba.



(a) Mensaje recibido después de presionar el botón de pánico

(b) Interfaz del dashboard con los 2 eventos notificados

Figura 6.4: Pruebas de choque de la NHTSA.

Los resutados como muestran las figuras, son los esperados para lo que se pretendía que hiciera el sistema. Sin embargo, hubo problemas en el envío de coordenadas GPS en la prueba del botón de pánico en lugares cerrados (estacionamientos y calles encerradas entre edificios), debido a que no se obtenían las coordenadas GPS en ese lugar y por tanto el mensaje se enviaba sin esa información. Esto se debió probablemente al uso de red 2G del FONA, que posee una cobertura mucho menor a la del 3G pensado inicialmente.

Se espera que este problema desaparezca con el uso de un FONA 3G y además, la inclusión de un buffer con coordenadas y *timeStamp* para enviar la última localización GPS con su hora y así estimar qué tan cerca pudo ocurrir el accidente real.

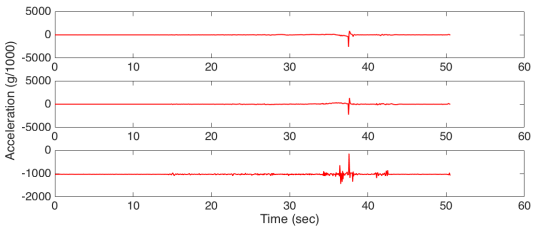
6.3. Post-procesamiento

En esta parte se usaron los datos de las mismas pruebas descritas en la sección 6.1.1 y 6.1.2. Los datos de la sección 6.1.3 no se pueden usar debido a que no tienen los ángulos de Euler en la medición.

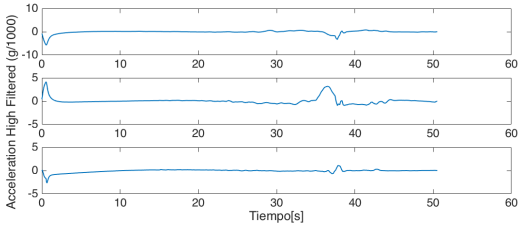
6.3.1. Filtro de señales

Las pruebas tomadas con el acelerómetro en versión kit fueron las de giros y choques contra obstáculos (las de caída libre o agitar acelerómetro, carecen de sentido para esta parte).

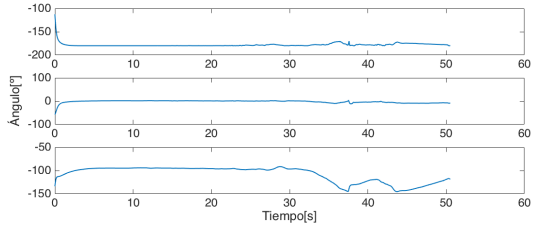
El procesamiento, como se dijo en la sección de Implementación, permite que las señales de aceleración y ángulos atenúen el ruido y eliminar los offset (además de la componente de gravedad en la aceleración). En la Figura 6.5 se muestra el resultado de la aplicación de todos los filtros para la aceleración y los ángulos en una de las pruebas realizadas.



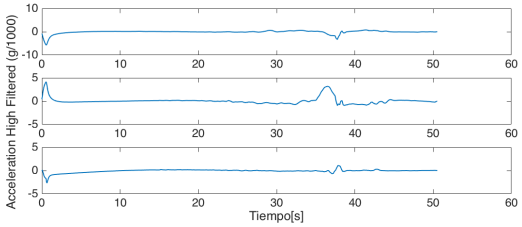
(a) Aceleraciones leídas por MPU9250



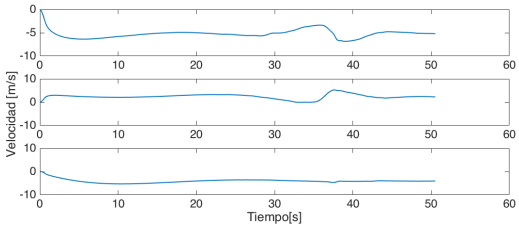
(b) Aceleraciones filtradas



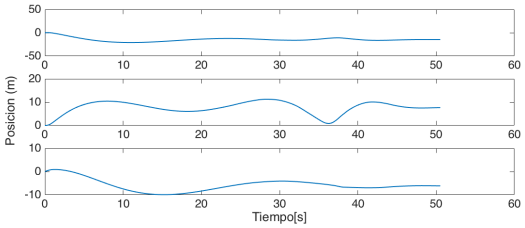
(c) Ángulos leídos por MPU9250



(d) Ángulos después del filtro



(e) Velocidad del auto en el tiempo



(f) Posición del auto en el tiempo

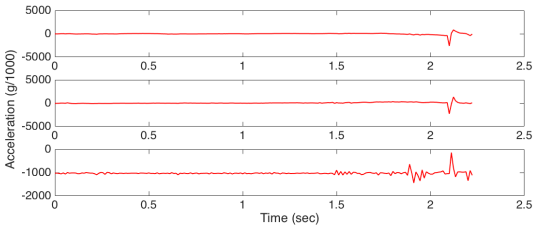
Figura 6.5: Prueba de choque a baja escala. Choque después de un viraje hacia la derecha en 50 segundos.

Este resutado muesta que la aplicación del filtro ayuda a centrar y atenuar las señales, pero aún así el ruido se arrastra y las integraciones poseen resultados que se reproducen mal en el simulador, debido al ruido en la señal de velocidad que se traduce en cambios de posición muy alejados de la realidad.

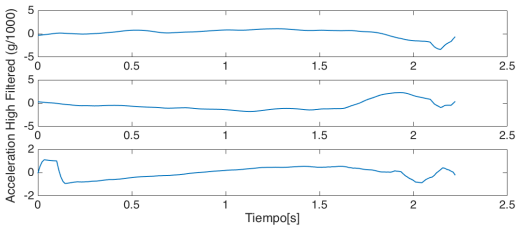
En las pruebas de manejo de vehículo, el resultado de la simulación fue aún peor debido a la duración de esas señales. Dado esto, es que para poder realizar la simulación del accidente de mejor forma posible, es necesario acortar estas curvas a no más de 2 o 3 segundos previos a la detección del choque y asumir que la velocidad al momento posterior de la detección del choque es de 0 $\frac{km}{h}$ para así poder encontrar una velocidad inicial restando este valor en toda

la curva para finalmente, obtener la nueva posición en el tiempo después de la integración.

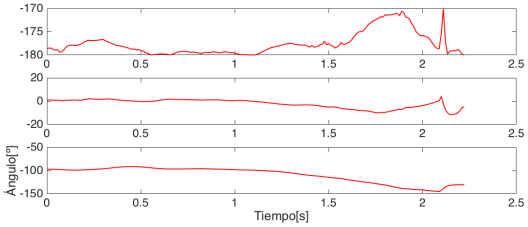
La Figura 6.6 muestra los nuevos resultados de la misma prueba anterior de la Figura 6.5.



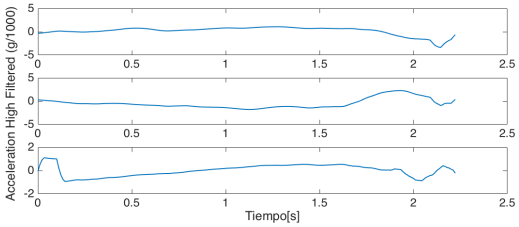
(a) Aceleraciones leídas por MPU9250



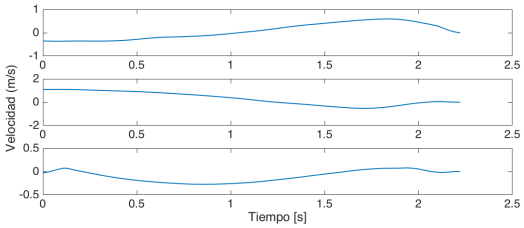
(b) Aceleraciones filtradas



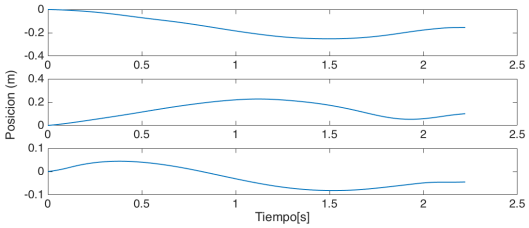
(c) Ángulos leídos por MPU9250



(d) Ángulos después del filtro



(e) Velocidad del auto en el tiempo



(f) Posición del auto en el tiempo

Figura 6.6: Prueba de choque a baja escala considerando sólo 2 segundos previos al choque forzadamente detectado.

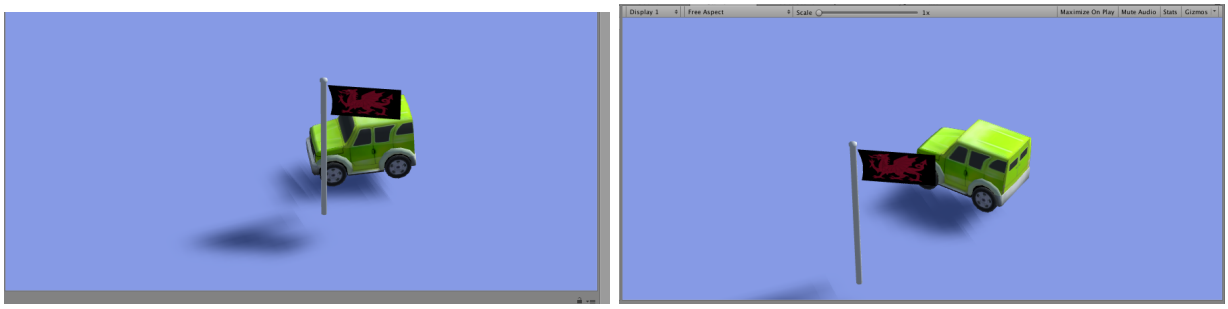
De esta forma, mejoró notoriamente la obtención del desplazamiento del objeto en el instante que importa, es decir, el momento previo al accidente. Con estas nuevas curvas de desplazamiento y ángulo, es posible simular el accidente en Unity3d.

6.3.2. Simulación en Unity3D

Con esta adaptación del algoritmo original para obtener sólamente el desplazamiento en un corto período de tiempo, se obtuvo una simulación más cercana a lo que ocurrió en la realidad. Sin embargo, pequeños ruidos de aceleración hacen variar el comportamiento del objeto en el simulador, por lo que la posición final no es exacta.

Es por esto, que esta simulación cumple con el objetivo principal de obtener una visión general de cómo se comportó el automóvil previo al accidente, pero no es posible obtener con certeza la distancia recorrida, debido a que el ruido en los ejes del acelerómetro es suficiente como para desviar esta trayectoria.

En la Figura 6.7 se muestran las 2 etapas de la simulación: cuando comienza y cómo termina el automóvil durante el movimiento (choque al virar hacia la derecha). La bandera indica el punto inicial del auto en la simulación.



(a) Etapa inicial de la simulación

(b) Etapa final: choque

Figura 6.7: Etapas de simulación en Unity3D del choque a baja escala con viraje hacia la derecha.

Como se observa, el automóvil quedó en la orientación real a la de la prueba. Sin embargo, ocurrió un leve desplazamiento hacia atrás debido al ruido de la aceleración en ese eje y por ende, la distancia final desplazada no corresponde a la ocurrida en la realidad.

Este es un resultado positivo en la medida de poder obtener una pequeña pista de cómo fue evolucionando la orientación y velocidad del vehículo previo al accidente. Pero no es un resultado óptimo para un peritaje detallado del choque en el que se quiera determinar cuántos metros avanzó el vehículo o cuánto se desplazó posterior la choque, pues el ruido del acelerómetro incide en estos desplazamientos incorrectos.

Se realizó la misma implementación con las demás pruebas obteniendo resultados similares e incluso menos precisos en desplazamiento en aquellos casos en que las aceleraciones aumentaban. Lo cual reafirma lo dicho en el párrafo anterior de que no es un resultado óptimo para evaluar trayectorias completas.

Capítulo 7

Conclusiones y trabajo futuro

En este trabajo se presenta un sistema de detección, notificación y simulación, bajo ciertas condiciones, de accidentes automovilísticos, a través del diseño e implementación de un dispositivo electrónico que va conectado al vehículo y posee un algoritmo autónomo para detectar choques. Todo en el marco del proyecto Sosmart Premium de la empresa SosmartLabs, creadora de la aplicación Sosmart y que pretende, a través de este proyecto, extender el servicio de detección de accidentes a un hardware pra automóviles.

El diseño del dispositivo consistió en un Arduino Pro-Mini, un acelerómetro/giróscopo MPU9250 de 16g de rango de medición y un FONA 2G como módulo de comunicación para enviar las notificaciones en sus 3 formatos: SMS, llamadas y POST Request. La elección de estos componentes permitió armar el prototipado de forma rápida y robusta debido a la existencia de librerías documenadas para ello. El MPU9250 se configuró para medir en un rango de 16g debido a que las aceleraciones en los accidentes vehiculares supera con facilidad los 40g y por tanto, con un rango mayor, el algoritmo es capaz de discernir de mejor forma cuándo se trata de un choque o un falso positivo. Sin embargo, este componente al ser tratado con este rango también aumenta su ruido en la medición por lo que un sistema de filtros e histéresis fue necesario para atenuar este ruido y así aumentar el desempeño del algoritmo.

Las pruebas para validar el algoritmo consistieron en 3 tipos: una base de datos con choques reales, pruebas de choques a baja escala usando una patineta y pruebas de manejo real por las calles de la ciudad de Santiago. Los resultados arrojaron que el algoritmo sólo detectó choque en aquellos datos que sí representaban choques mayores a 50[km/h], desechando las pruebas a baja escala y las de manejo real en automóvil, por lo que se puede decir que el algoritmo cumple con lo solicitado.

El sistema de notificación también funciona cuando el FONA 2G es capaz de enviar una localización GPS al momento de la detección. De esta forma, los choques son guardados en una base de datos de Back4App y pueden ser monitoreados en tiempo real a través de una plataforma web. El mayor problema de este proceso fue que el FONA al utilizar la red 2G, en lugares cerrados o cerca de edificios altos, no posee la suficiente señal para enviar los datos a Internet ni obtener las coordenadas GPS, por lo que este sistema de notificación sólo es válido en carreteras o calles abiertas.

El post-procesamiento se tuvo que realizar de aparte del uso del dispositivo, debido a que no fue posible integrar un sistema de almacenamiento de información sin reducir la frecuencia de adquisición de los datos. Debido a que la frecuencia no debe disminuir de los 80Hz par que el algoritmo funcione correctamente, es que se optó por desechar este almacenamiento y la obtención de los datos se hizo con una versión kit del prototipo en el que se usaba el computador para leer y guardar los datos para su posterior procesamiento.

EL post-procesamiento de los datos consistió en el uso de histéresis y filtros para atenuar lo máximo posible el ruido de las señales de aceleración y más importante, extraer la componente por gravedad de los datos. De esta forma, a través de integración se pudo obtener la curva de desplazamiento del vehículo. Este proceso, no resultó como se esperaba debido a que el ruido aún seguía siendo lo suficiente como para arrastrar el error en curvas de larga duración y por tanto, la simulación no se aceracaba a la realidad. Es por esto, que se decidió tomar sólamente el momento previo al choque y asumir como velocidad final 0 [km/h] en el momento posterior a este.

Con esta nueva implementación, la simulación obtuvo mejores resultados y se pudo observar cómo se movía el automóvil en el momento previo al choque y después de este. Sin embargo, los pequeños ruidos existentes no permitieron que la simulación pudiera obtener de manera precisa la trayectoria del vehículo, dado que en algunas pruebas, el vehículo si bien se orientaba según la realidad, no se trasladaba la distancia recorrida en realidad.

De esta forma, se concluye que este trabajo diseñó e implementó un prototipo de dispositivo electrónico capaz de detectar choques de alto riesgo con el uso de un algoritmo de buen desempeño y puede comunicarse inmediatamente después por medio de los 3 canales: SMS, Post Request y llamada. En el caso de Internet, se creó una plataforma web de monitoreo para completar la arquitectura del sistema. Finalmente, esos choques son capaces de caracterizarse parcialmente usando un simulador en el que se observan los movimientos del automóvil previo y posterior al choque, aunque las distancias recorridas no son representativas del todo en la realidad.

Debido a los problemas descritos, se consideran las siguientes mejoras como trabajo futuro para complementar este proyecto y avanzar en la línea de un producto de calidad mundial.

El uso del acelerómetro con un rango de 16g fue un acierto, pero imposibilitó la escritura de los datos en una tarjeta SD para almacenar la información que posteriormente será analizada para caracterizar el choque. Es por esto, que se sugiere una nueva estrategia de comunicación para leer los datos del MPU9250 y almacenarlos en una tarjeta externa. Para lograr esto, se puede cambiar el Arduino Pro-Mini por uno de mayor procesamiento (la serie de los MEGA) y puertos de comunicación serial en el que las comunciaciones no se interrumpan sin tener que bajar la frecuencia de adquisición (mínimo 80Hz).

Por otro lado, el uso del FONA 2G presentó problemas de señal en ciertos lugares, esto debería verse disminuído con el uso del FONA 3G, que funciona con las mismas librerías por lo que no sería problema reintegrar el componente en el dispositivo, salvo que el 3G es de un tamaño levemente mayor.

El dispositivo, no cuenta con salidas USB, que son muy importante para la usabilidad del

producto. Pues si bien, no influye en los objetivos de este trabajo, si será un factor importante a la hora de vender el producto comercialmente, por lo que se sugiere instalar puertos USB y conectarlos directamente a la alimentación 5V.

El prototipo tampoco posee una alimentación independiente que permita mantener encendido el dispositivo un lapso de tiempo, posterior a que se desenchufe de la cigarrera. Esto es importante en la práctica debido a que un choque podría dañar la alimentación y desactivar el dispositivo antes de poder enviar la notificación o sin poder establecer la llamada. Por lo que es necesario el uso de una batería externa (adicional a la ya utilizada para el FONA) que se cargue cuando el dispositivo esté conectado y alimente el sistema cuando no.

El algoritmo de detección tuvo muy buen desempeño en choques reales estudiados de una base de datos y fue robusto a falsos positivos derivados de situaciones de choque a baja escala. Sin embargo, no se pudo probar en choques de mediana intensidad debido a que no se encontraron estos datos, por lo que no es posible definir el umbral de desempeño en el que el algoritmo detecta choques. Sólo se validó para accidentes de alto riesgo que si bien son los más importantes de detectar, se espera de un producto de calidad, definir el mínimo de velocidad de choque en el que el algoritmo actúa.

Para finalizar, en el tema de la caracterización, el resultado de la simulación no es tan robusto como se esperaba en un principio, debido al ruido del acelerómetro. Sin embargo, otra alternativa aunque un poco más costosa, es el uso de algoritmos de clasificación más sofisticados que obtengan un grado de gravedad del accidente y que también podrían ser notificados. O al igual que en este proyecto, ser parte de un post-procesamiento para rescatar información útil del accidente. Entre estas técnicas, se encuentra DTW, SVM y redes neuronales, que son capaces de tomar las curvas de aceleración y luego de un proceso de entrenamiento, determinar el tipo de accidente que ocurrió, sin la necesidad de un simulador.

Bibliografía

[1] IIHS-HLDI. Fatality facts statistics. http://www.iihs.org/iihs/topics/t/general-statistics/fatalityfacts/state-by-state-overview. Accessed: 2017-05-21.

[2] ASIRT. Road crash statistics. http://asirt.org/initiatives/informing-road-users/road-safety-facts/road-crash-statistics. Accessed: 2017-05-22.

[3] W. Evanco. The impact of rapid incident detection on freeway accident fatalities. *Mitretek Systems, Inc*, 1996.

[4] CONASET. Información estadística descriptiva de la realidad vial de chile. https://www.conaset.cl/programa/observatorio-datos-estadistica/biblioteca-observatorio/estadisticas-generales/. Accessed: 2017-05-21.

[5] Venture Beat. Noticia de los 100 mejores start-ups. https://venturebeat.com/2016/05/01/100-noteworthy-young-startups. Accessed: 2017-05-03.

[6] Momo. Reloj gps-celular para niños. https://soymomo.com. Accessed: 2017-05-03.

[7] Sosmart Labs. Sosmart app. http://www.sosmartapp.com. Accessed: 2017-05-03.

[8] NHTSA. National highway traffic safety asocitation. https://www.nhtsa.gov. Accessed: 2017-05-03.

[9] Fayerwayer. Blog sobre sosmart app. https://www.fayerwayer.com/2015/11/sosmart-la-app-que-podria-salvar-tu-vida-en-un-accidente-de-transito/. Accessed: 2017-05-03.

[10] Emol. Noticia de sosmart app. http://www.emol.com/noticias/Economia/2015/11/05/757831/SOSmart.html. Accessed: 2017-05-04.

[11] Premier Choice Insurance. Five cool apps to have in the event of a car accident. http://premierchoiceaz.com/five-cool-apps-car-accident/. Accessed: 2017-05-04.

[12] Auxilia. Página oficial. http://www.auxilia.cl. Accessed: 2017-05-05.

[13] SAMU Brasil. Official page. https://www.samu.es/samu-internacional-brasil/.

Accessed: 2017-05-21.

[14] Splitsecnd. Official landing page. https://www.splitsecnd.com. Accessed: 2017-05-05.

[15] Unidad Coronaria Móvil de Chile. Página oficial. http://www.ucmchile.cl. Accessed: 2017-05-05.

[16] Wikipedia. Arduino. https://es.wikipedia.org/wiki/Arduino. Accessed: 2017-05-13.

[17] El androide libre. Funcionamiento acelerómetro/giróscopo. https://www.elandroidelibre.com/2016/07/giroscopio-movil-android.html. Accessed: 2017-05-16.

[18] Adafruit. Official page. https://www.adafruit.com. Accessed: 2017-05-20.

[19] Matlab. Official page. https://www.mathworks.com/products/matlab.html. Accessed: 2017-06-14.

[20] Unity3D. Official page. https://unity3d.com. Accessed: 2017-06-25.

[21] Vuforia. Official page. https://www.vuforia.com. Accessed: 2017-06-25.

[22] European Comission. E-call. https://ec.europa.eu/digital-single-market/en/ecall-time-saved-lives-saved. Accessed: 2017-05-20.

[23] CEA seguridad vial. Blog sobre e-call. https://www.seguridad-vial.net/vehiculo/seguridad-pasiva/120-que-es-el-ecall. Accessed: 2017-05-21.

[24] GM. Onstar reverses decision to change terms and conditions. http://media.gm.com/media/us/en/gm/news.detail.html/content/Pages/news/us/en/2011/Sep/0927_onstar.html. Accessed: 2017-06-12.

[25] On Star. Official page. https://www.onstar.com. Accessed: 2017-06-12.

[26] Automatic. Official page. https://www.automatic.com. Accessed: 2017-06-12.

[27] Rajesh Rajamani Saber Taghvaeeyan. Two-dimensional sensor system for automotive crash prediction. Vol. 15:N°1, February, 2014.

[28] Brian Dougherty Adam Albright Chris Thompson, Jules White and Douglas C. Schmidt. Using smartphones to detect car accidents and provide situational awareness to emergency responders. *Institute for Software Integrated Systems*.

[29] Zainab S. Alwan Hamid M. Ali. Car accident detection and notification system using smartphone. *IJCSMC*, Vol. 4:620–635, April, 2015.

[30] Ruba Abu-Salma Humaid Al-Ali May Al-Merri Fadi Aloul, Imran Zualkernan. ibump: Smartphone application to detect car accidents. August, 2014.

[31] Francisco J. Martinez Manuel Fogue, Piedad Garrido. A system for automatic notification and severity estimation of automotive accidents. 2013.

[32] NHTSA. National highway traffic safety asocitation database. https://www-nrd.nhtsa.dot.gov/database/VSR/veh/LatestTestInfo.aspx. Accessed: 2017-05-04.

[33] Dainius Dalmotas Alan German. Crash pulse fata from event data record in rigid barrier tests. *Jean-Louis Comeau Transport Canada*, 11-0395.

[34] Adafruit. Fona 2g. https://www.adafruit.com/product/2542. Accessed: 2017-06-27.

[35] Arduino. Arduino pro-mini. https://store.arduino.cc/usa/arduino-pro-mini. Accessed: 2017-05-14.

[36] SparkFun. Mpu-9250. https://learn.sparkfun.com/tutorials/mpu-9250-hookup-guide?_ga=1.69945686.718755419.1484683726. Accessed: 2017-05-15.

[37] SparkFun. Micro sd breakout. https://www.sparkfun.com/products/544. Accessed: 2017-05-18.

[38] Instructables. Uploading sketch to arduino pro mini using arduino uno. http://www.instructables.com/id/Uploading-sketch-to-Arduino-Pro-Mini-using-Arduino/. Accessed: 2017-06-14.

[39] Adafruit. Fona 2g conexión. https://learn.adafruit.com/adafruit-fona-808-cellular-plus-gps-breakout/overview. Accessed: 2017-06-14.

[40] Adafruit. Fona 2g librería. https://learn.adafruit.com/adafruit-fona-808-cellular-plus-gps-breakout/arduino-test. Accessed: 2017-06-14.

[41] Git Hub: Kris Winer. Arduino sketches for mpu9250 9dof with ahrs sensor fusion. https://github.com/kriswiner/MPU-9250. Accessed: 2017-05-18.

[42] SparkFun. Micro sd wiring and software. https://learn.sparkfun.com/tutorials/microsd-shield-and-sd-breakout-hookup-guide. Accessed: 2017-06-14.

[43] Educachip. 5 formas de pasar de 5v a 3.3v. http://www.educachip.com/pasar-de-5v-a-3-3v/#Usando-un-regulador-de-voltaje-variable-LM317. Accessed: 2017-06-29.

[44] Back4app. Official page. https://www.back4app.com. Accessed: 2017-06-28.

[45] Ruby on Rails. Official page. http://rubyonrails.org. Accessed: 2017-06-28.

[46] Luc van Wietmarschen Charlotte Treffers. Position and orientation determination of a

probe with use of the imu mpu and a atmega 8 microcontroller. June 15,2016.

Capítulo 8

Anexo

8.1. Código en Arduino para utilizar FONA 2G

```
1
#include <Adafruit_FONA.h>
3 #include <SoftwareSerial.h>
#define FONA_RX 2
5 #define FONA_TX 3
#define FONA_RST 4
7
//Fona variables
9 SoftwareSerial fonaSS = SoftwareSerial(FONA_TX, FONA_RX);
SoftwareSerial *fonaSerial = &fonaSS;
11 Adafruit_FONA fona = Adafruit_FONA(FONA_RST);
char number[12]
13 char messageSMS[150]
char url[80]
15 char dataJSON[140]
float latitude, longitude, speed_kph, heading, speed_mph, altitude;
17 int gpsCounter=0

19 void setup() {

21   //Iniciar comunicaci n con el FONA
   while (!Serial);
23   Serial.begin(115200);
   Serial.println(F("Iniciando....(Puede tomar 3 segundos)"));
25   fonaSerial->begin(4800);
   if (! fona.begin(*fonaSerial)) {
27     Serial.println(F("No se pudo encontrar FONA"));
     while (1);
29   }
   Serial.println(F("FONA is OK, listo para configurar..."));
31   settingFona();
}
33
//Configurar fona
35 void settingFona(){
```

```
37    //FIjar numero de telefono para llamada emergencia
      number = "962381578"
39    //Fijar url para hacer POST
      url = "soymomosupport.herokuapp.com/memoria"
41
      //Configurar salida de audio por parlante y microfono al maximo
43    if (! fona.setAudio(FONA_EXTAUDIO)) {
        Serial.println(F("No se configur  salida de audio por parlante"));
45    } else {
        Serial.println(F("OK! Configurado parlante"));
47    }
      fona.setMicVolume(FONA_EXTAUDIO, 10);
49
      //Configurar volumen del parlante al maximo
51    if (! (fona.setVolume(100) && fona.setRingVolume(100)) ) {
        Serial.println(F("No se configur  volumen del parlante"));
53    } else {
        Serial.println(F("OK! Configurado volumen del parlante al m ximo"));
55    }

57    //Encender GPS
      if (! (fona.enableGPS(true)) ) {
59      Serial.println(F("No se activ  GPS"));
      } else {
61      Serial.println(F("OK! Activado el GPS"));
      }
63  }

65  //Funcion que se ejecuta en cada frame
    void loop() {
67
      //Obtener coordenadas GPS y velocidad
69    boolean gps_success = fona.getGPS(&latitude, &longitude, &speed_kph, &
       heading, &altitude);
      if (gps_success) {
71      gpsCounter=0;
        Serial.print("GPS latitude:");
73      Serial.println(latitude);
        Serial.print("GPS longitude:");
75      Serial.println(longitude);
        Serial.print("GPS velocidad KPH:");
77      Serial.println(speed_kph);
      }
79    else {
        Serial.println("No se obtuvo localizaci n GPS");
81      gpsCounter++;
      }
83
      // Veriricar conexion a torres celular, asi encender GPRS y obtener
       ubicacion. No funciona en el Fona 3G
85    if (fona.getNetworkStatus() == 1) {
        boolean gsmloc_success = fona.getGSMLoc(&latitude, &longitude);
87      if (gsmloc_success) {
          gpsCounter=0
89        Serial.print("GSMLoc latitude:");
```

```
      Serial.println(latitude);
      Serial.print("GSMLoc longitude:");
      Serial.println(longitude);
    } else {
      gpsCounter++;
      Serial.println("GSM location failed...");
      Serial.println(F("Disabling GPRS"));
      fona.enableGPRS(false);
      Serial.println(F("Enabling GPRS"));
      if (!fona.enableGPRS(true)) {
        Serial.println(F("Failed to turn GPRS on"));
      }
    }
  }

  //Si ocurre la deteccion o se presiona el boton de panico
  if (DETECTION || PANIC_BUTTON) {
    //Construir mensaje a enviar por SMS y tambien el JSON del POST request
    char latitud[50], longitud[50], velocidad_choque[50];
    sprintf (latitud, "%f", latitude);
    sprintf (longitud, "%f", longitude);
    sprintf (velocidad_choque, "%f", speed_kph);
    if (gpsCounter > 300){
      if (DETECTION){
        message = "La data no es reciente. Tuve un accidente, coordenadas son:
    " +"("+latitud+","+longitud +")";
        dataJSON = "{\"latitude\":\" " + latitud+ " \",\"longitude\":\" " +
   latitud+ " \",\"type\":\"carCrash\"}";
      }
      else {
        message = "La data no es reciente. Presion  el boton, coordenadas son
    : " +"("+latitud+","+longitud +")";
        dataJSON = "{\"latitude\":\" " + latitud+ " \",\"longitude\":\" " +
   latitud+ " \",\"type\":\"emergencyButton\"}";
      }
    }
    else{
      if (DETECTION){
        message = "Tuve un accidente, coordenadas son: " +"("+latitud+","+
   longitud +")";
        dataJSON = "{\"latitude\":\" " + latitud+ " \",\"longitude\":\" " +
   latitud+ " \",\"type\":\"carCrash\"}";
      }
      else {
        message = "Presione el boton, coordenadas son: " +"("+latitud+","+
   longitud +")";
        dataJSON = "{\"latitude\":\" " + latitud+ " \",\"longitude\":\" " +
   latitud+ " \",\"type\":\"emergencyButton\"}";
      }
    }

    // Enviar SMS
    if (!fona.sendSMS(number, message)) {
      Serial.println(F("Fallido envio de mensaje"));
    } else {
      Serial.println(F("Enviado!"));
```

```
    }
    // Enviar POST Request
    uint16_t statuscode;
    int16_t length;
    if (!fona.HTTP_POST_start(url, F("text/plain"), (uint8_t *) dataJSON,
    strlen(dataJSON), &statuscode, (uint16_t *)&length)) {
      Serial.println("Fallo envio de POST!");
      break;
    }
    while (length > 0) {
      while (fona.available()) {
        char c = fona.read();
        #if defined(__AVR_ATmega328P__) || defined(__AVR_ATmega168__)
          loop_until_bit_is_set(UCSR0A, UDRE0); /* Wait until data register
    empty. */
          UDR0 = c;
        #else
          Serial.write(c);
        #endif
        length--;
        if (! length) break;
      }
    }
    fona.HTTP_POST_end();

    // Realizar llamada
    delay(1000);
    if (!fona.callPhone(number)) {
      Serial.println(F("Fallida la llamada"));
    } else {
      Serial.println(F("Llamando!"));
    }
    delay(15000);
  }
}
```

8.2. Código en Ruby On Rails de una sección de la plataforma Web que recibe el POST Request

```
  def memoria

    latitude = params['latitude']
    longitude = params['longitude']
    type = params['emergency']

    if type == "carCrash"
      carCrash = MyParse.object("CarCrash")
      carCrash["location"] = Parse::GeoPoint.new({"latitude" => latitude.to_f,
    "longitude" => longitude.to_f })
      carCrash.save
```

```
    elsif type == "emergencyButton"
      emergencyButton = MyParse.object("EmergencyButton")
      emergencyButton["location"] = Parse::GeoPoint.new({"latitude" =>
    latitude.to_f, "longitude" => longitude.to_f })
      emergencyButton.save
    else

    end

    redirect_to welcome_index_url
  end
```

8.3. Código Arduino-MPU9250

8.3.1. Código adaptado en Arduino para lectura de datos de MPU9250

```
#include <SoftwareSerial.h>
#include "quaternionFilters.h"
#include "MPU9250.h"

#define AHRS true         // Set to false for basic data read
#define SerialDebug true  // Set to true to get Serial output for debugging

MPU9250 myIMU;
bool accel_enabled = false;

void setup() {

  //Configurar comunicacion serial
  Wire.begin();
  Serial.begin(38400);

  //Configurar inicio de MPU9250: Calibraciones internas
  byte c = myIMU.readByte(MPU9250_ADDRESS, WHO_AM_I_MPU9250);
  Serial.print("MPU9250 "); Serial.print("I AM "); Serial.print(c, HEX);
  Serial.print(" I should be "); Serial.println(0x71, HEX);

  if (c == 0x71) // WHO_AM_I should always be 0x68
  {
    Serial.println("MPU9250 is online...");

    // Start by performing self test and reporting values
    myIMU.MPU9250SelfTest(myIMU.SelfTest);

    // Calibrate gyro and accelerometers, load biases in bias registers
    myIMU.calibrateMPU9250(myIMU.gyroBias, myIMU.accelBias);
    myIMU.initMPU9250();
    // Initialize device for active mode read of acclerometer, gyroscope, and
    // temperature
    Serial.println("MPU9250 initialized for active data mode....");
```

```
36      // Read the WHO_AM_I register of the magnetometer, this is a good test of
        // communication
38      byte d = myIMU.readByte(AK8963_ADDRESS, WHO_AM_I_AK8963);
        Serial.print("AK8963 "); Serial.print("I AM "); Serial.print(d, HEX);
40      Serial.print(" I should be "); Serial.println(0x48, HEX);

42      // Get magnetometer calibration from AK8963 ROM
        myIMU.initAK8963(myIMU.magCalibration);
44      // Initialize device for active mode read of magnetometer
        Serial.println("AK8963 initialized for active data mode....");
46      accel_enabled= true;
    } // if (c == 0x71)
48  else
    {
50      Serial.print("Could not connect to MPU9250: 0x");
        Serial.println(c, HEX);
52      accel_enabled= false;
    }
54  }

56  void loop() {
    if (!accel_enabled){
58      //Si MPU9250 no inicia, salir.
        return;
60  }

62  //Si esta listo para leer un nuevo dato
    if (myIMU.readByte(MPU9250_ADDRESS, INT_STATUS) & 0x01)
64  {
      myIMU.readAccelData(myIMU.accelCount);  // Read the x/y/z adc values
66    myIMU.getAres();

68    // Now we'll calculate the accleration value into actual mg's
      // This depends on scale being set
70    myIMU.ax = (float)myIMU.accelCount[0]*myIMU.aRes; // - accelBias[0];
      myIMU.ay = (float)myIMU.accelCount[1]*myIMU.aRes; // - accelBias[1];
72    myIMU.az = (float)myIMU.accelCount[2]*myIMU.aRes; // - accelBias[2];

74    myIMU.readGyroData(myIMU.gyroCount);  // Read the x/y/z adc values
      myIMU.getGres();
76
      // Calculate the gyro value into actual degrees per second
78    // This depends on scale being set
      myIMU.gx = (float)myIMU.gyroCount[0]*myIMU.gRes;
80    myIMU.gy = (float)myIMU.gyroCount[1]*myIMU.gRes;
      myIMU.gz = (float)myIMU.gyroCount[2]*myIMU.gRes;
82
      myIMU.readMagData(myIMU.magCount);  // Read the x/y/z adc values
84    myIMU.getMres();
      // User environmental x-axis correction in milliGauss, should be
86    // automatically calculated
      myIMU.magbias[0] = +470.;
88    // User environmental x-axis correction in milliGauss TODO axis??
      myIMU.magbias[1] = +120.;
90    // User environmental x-axis correction in milliGauss
```

```
    myIMU.magbias[2] = +125.;
92
    // Calculate the magnetometer values in milliGauss
94  // Include factory calibration per data sheet and user environmental
    // corrections
96  // Get actual magnetometer value, this depends on scale being set
    myIMU.mx = (float)myIMU.magCount[0]*myIMU.mRes*myIMU.magCalibration[0] -
98              myIMU.magbias[0];
    myIMU.my = (float)myIMU.magCount[1]*myIMU.mRes*myIMU.magCalibration[1] -
100             myIMU.magbias[1];
    myIMU.mz = (float)myIMU.magCount[2]*myIMU.mRes*myIMU.magCalibration[2] -
102             myIMU.magbias[2];
  } // if (readByte(MPU9250_ADDRESS, INT_STATUS) & 0x01)
104
  // Must be called before updating quaternions!
106 myIMU.updateTime();

108 //Obetener quateriones
  MahonyQuaternionUpdate(myIMU.ax, myIMU.ay, myIMU.az, myIMU.gx*DEG_TO_RAD,
110                       myIMU.gy*DEG_TO_RAD, myIMU.gz*DEG_TO_RAD, myIMU.my,
                          myIMU.mx, myIMU.mz, myIMU.deltat);
112
  myIMU.delt_t = millis() - myIMU.count;
114
  //Configurar velocidad de lectura -> 90Hz
116 if (myIMU.delt_t > 0)
  {
118   //Obtener  ngulos  en grados a partir de Quaterniones
    myIMU.yaw   = atan2(2.0f * (*(getQ()+1) * *(getQ()+2) + *getQ() *
120                 *(getQ()+3)), *getQ() * *getQ() + *(getQ()+1) * *(getQ()+1)
                  - *(getQ()+2) * *(getQ()+2) - *(getQ()+3) * *(getQ()+3));
122   myIMU.pitch = -asin(2.0f * (*(getQ()+1) * *(getQ()+3) - *getQ() *
                  *(getQ()+2)));
124   myIMU.roll  = atan2(2.0f * (*getQ() * *(getQ()+1) + *(getQ()+2) *
                  *(getQ()+3)), *getQ() * *getQ() - *(getQ()+1) * *(getQ()+1)
126               - *(getQ()+2) * *(getQ()+2) + *(getQ()+3) * *(getQ()+3));
    myIMU.pitch *= RAD_TO_DEG;
128 myIMU.yaw   *= RAD_TO_DEG;
    myIMU.yaw   -= 8.5;
130 myIMU.roll  *= RAD_TO_DEG;

132   //Print en consola en formato Ax,Ay,Az,roll,pitch,yaw
    Serial.print((int)1000*myIMU.ax);
134 Serial.print(","); Serial.print((int)1000*myIMU.ay);
    Serial.print(","); Serial.print((int)1000*myIMU.az);
136 Serial.print(",");
    Serial.print(myIMU.roll, 2);
138 Serial.print(",");
    Serial.print(myIMU.pitch, 2);
140 Serial.print(",");
    Serial.println(myIMU.yaw, 2);
142
    myIMU.count = millis();
144 }
}
```

8.3.2. Código en Arduino para el algoritmo de detección

```
#include <SoftwareSerial.h>
#include "quaternionFilters.h"
#include "MPU9250.h"
#define AHRS true         // Set to false for basic data read
#define SerialDebug true  // Set to true to get Serial output for

//Parameters
#define WINDOW_SEARCH_LENGTH 7
#define WINDOW_DETECTION_LENGTH 3
#define ACCEL_THRESHOLD 12000
#define INPUT_THRESHOLD 12000
#define OUTPUT_THRESHOLD 10000
#define WINDOW_SEARCH_POINTS 4
#define DETECTION_POINTS 5

//Acelerometer variables
MPU9250 myIMU;
int window_search[WINDOW_SEARCH_LENGTH] = {0,0,0,0,0,0,0};
int window_detection[WINDOW_DETECTION_LENGTH] = {0,0,0};
int window_search_index=0;
int window_detection_index=0;
boolean insideHisteresis=false;
int detection_counter=0;
int acceleration_module=0;
boolean accel_enabled= true;

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  Wire.begin();
  byte c = myIMU.readByte(MPU9250_ADDRESS, WHO_AM_I_MPU9250);
  Serial.print("MPU9250 "); Serial.print("I AM "); Serial.print(c, HEX);
  Serial.print(" I should be "); Serial.println(0x71, HEX);
  if (c == 0x71) // WHO_AM_I should always be 0x68
  {
    myIMU.calibrateMPU9250(myIMU.gyroBias, myIMU.accelBias);
    myIMU.initMPU9250();
    byte d = myIMU.readByte(AK8963_ADDRESS, WHO_AM_I_AK8963);
    myIMU.initAK8963(myIMU.magCalibration);
  }
  else
  {
    accel_enabled= false;
  }
}

void loop(){
  if (!accel_enabled){
    return;
```

```
  }
  if (myIMU.readByte(MPU9250_ADDRESS, INT_STATUS) & 0x01)
  {
    myIMU.readAccelData(myIMU.accelCount);  // Read the x/y/z adc values
    myIMU.getAres();
    myIMU.ax = (float)myIMU.accelCount[0]*myIMU.aRes; // - accelBias[0];
    myIMU.ay = (float)myIMU.accelCount[1]*myIMU.aRes; // - accelBias[1];
    myIMU.az = (float)myIMU.accelCount[2]*myIMU.aRes; // - accelBias[2];
    myIMU.readGyroData(myIMU.gyroCount);  // Read the x/y/z adc values
    myIMU.getGres();
    myIMU.gx = (float)myIMU.gyroCount[0]*myIMU.gRes;
    myIMU.gy = (float)myIMU.gyroCount[1]*myIMU.gRes;
    myIMU.gz = (float)myIMU.gyroCount[2]*myIMU.gRes;
    myIMU.readMagData(myIMU.magCount);  // Read the x/y/z adc values
    myIMU.getMres();
    myIMU.magbias[0] = +470.;
    myIMU.magbias[1] = +120.;
    myIMU.magbias[2] = +125.;
    myIMU.mx = (float)myIMU.magCount[0]*myIMU.mRes*myIMU.magCalibration[0] -
   myIMU.magbias[0];
    myIMU.my = (float)myIMU.magCount[1]*myIMU.mRes*myIMU.magCalibration[1] -
   myIMU.magbias[1];
    myIMU.mz = (float)myIMU.magCount[2]*myIMU.mRes*myIMU.magCalibration[2] -
   myIMU.magbias[2];
  }

  myIMU.updateTime();
  MahonyQuaternionUpdate(myIMU.ax, myIMU.ay, myIMU.az, myIMU.gx*DEG_TO_RAD,
                         myIMU.gy*DEG_TO_RAD, myIMU.gz*DEG_TO_RAD, myIMU.my,
                         myIMU.mx, myIMU.mz, myIMU.deltat);
  myIMU.delt_t = millis() - myIMU.count;

  if (myIMU.delt_t > 00)
  {
    //Calcular modulo
    acceleration_module = (int) sqrt(1000000*myIMU.ax*myIMU.ax + 1000000*myIMU
   .ay*myIMU.ay + 1000000*myIMU.az*myIMU.az);

    //Guardar nuevos valores
    window_search[window_search_index] = acceleration_module;
    window_detection[window_detection_index] = acceleration_module;

    //Ingreso o salida de Histeresis
    if (!insideHisteresis){
      int counter=0;
      for (int c=0; c<WINDOW_SEARCH_LENGTH;c++){
        if (window_search[c]>INPUT_THRESHOLD){
          counter++;
        }
      }
      if (counter>=WINDOW_SEARCH_POINTS){
        insideHisteresis=true;
      }
    }
    else{
      int counter=0;
```

```
      for (int c=0; c<WINDOW_SEARCH_LENGTH;c++){
        if (window_search[c]>OUTPUT_THRESHOLD){
          counter++;
        }
      }
      if (counter<WINDOW_SEARCH_POINTS){
        insideHisteresis=false;
      }
    }

    //Evaluar deteccion dentro de histeresis
    if (insideHisteresis){
      int result=0;
      for (int c=0; c<WINDOW_DETECTION_LENGTH;c++){
        if (window_detection[c]>OUTPUT_THRESHOLD){
          result+=window_detection[c];
        }
      }
      result = result/WINDOW_DETECTION_LENGTH;

      if (result>ACCEL_THRESHOLD){
        detection_counter++;
      }
      else{
        detection_counter=0;
      }
    }
    else{
      detection_counter=0;
    }

    //Evaluar si hay deteccion o no. Encender LED del Arduino.
    if (detection_counter > DETECTION_POINTS){
      digitalWrite(LED_BUILTIN, HIGH);
    }
    else{
      digitalWrite(LED_BUILTIN, LOW);
    }

    //Actualizar indice de ventanas
    if (window_search_index == WINDOW_SEARCH_LENGTH){
      window_search_index=0;
    }
    else{
      window_search_index++;
    }
    if (window_detection_index == WINDOW_DETECTION_LENGTH){
      window_detection_index=0;
    }
    else{
      window_detection_index++;
    }

    if(SerialDebug)
    {
      Serial.println(acceleration_module);
```

```
    }
158     myIMU.count = millis();
  }
160 }
```

8.4. Código test en Arduino para escritura en micro SD

```
#include <SPI.h>
2 #include <SD.h>

4 const int chipSelect = 8;

6 void setup()
{
8   // Open serial communications and wait for port to open:
  Serial.begin(38400);
10  while (!Serial) {
    ; // wait for serial port to connect. Needed for Leonardo only
12  }

14  Serial.print("Iniciando SD...");
  pinMode(10, OUTPUT);
16  pinMode(chipSelect, OUTPUT);

18  if (!SD.begin(chipSelect)) {
    Serial.println("Tarjeta fallada o no detectada.");
20    return;
  }
22  Serial.println("Tarjeta reconocida.");
}
24
void loop()
26 {
  //Formar string (ac  se lee el sensor y se forma el string con los datos)
28  String dataString = "Ax,Ay,Az,roll,pitch,yaw";

30  //Abrir archivo para escribir (Crear si no existe).
  File dataFile = SD.open("sd_test.txt", FILE_WRITE);
32
  //Si el archivo est  disponible, se escribe.
34  if (dataFile) {
    dataFile.println(dataString);
36    dataFile.close();
  }
38  //Sino, imprimir error en consola.
  else {
40    Serial.println("Error en apertura de archivo");
  }
42 }
```

8.5. Código MATLAB para procesar señales de aceleración y giróscopo

```
%% Parametros
hist_min = 175;                         %Valor de entrada histeresis de angulo
hist_max = 170;                         %Valor de salida histeresis de angulo
low_filter_size_acceleration = 5;       %Tama o ventana para filtrar
    aceleraciones
low_filter_size_angles = 3;             %Tama o ventana para filtrar angulos

%% Importar data de archivo .txt
A = csvread('test_input.txt');
Ax    = A(:,1);
Ay    = A(:,2);
Az    = A(:,3);
roll  = A(:,4);
pitch = A(:,5);
yaw   = A(:,6);

%% Crear hist resis de angulos en +-180
roll_hist = roll;
isInside = false;
for i = 2:length(roll)
    if abs(roll(i)) >= hist_min
        isInside = true;
        if roll_hist(i-1)>0
            roll_hist(i) = abs(roll(i));
        else
            roll_hist(i) = -abs(roll(i));
        end
    elseif isInside && (abs(roll(i)) >= hist_max)
        if roll_hist(i-1)>0
            roll_hist(i) = abs(roll(i));
        else
            roll_hist(i) = -abs(roll(i));
        end
    else
        isInside =false;
    end
end

pitch_hist = pitch;
isInside = false;
for i = 2:length(pitch)
    if abs(pitch(i)) >= hist_min
        isInside = true;
        if pitch_hist(i-1)>0
            pitch_hist(i) = abs(pitch(i));
        else
            pitch_hist(i) = -abs(pitch(i));
        end
    elseif isInside && (abs(pitch(i)) >= hist_max)
        if pitch_hist(i-1)>0
            pitch_hist(i) = abs(pitch(i));
        else
```

```
            pitch_hist(i) = -abs(pitch(i));
        end
    else
        isInside =false;
    end
end

yaw_hist = yaw;
isInside = false;
for i = 2:length(yaw)
    if abs(yaw(i)) >= hist_min
        isInside = true;
        if yaw_hist(i-1)>0
            yaw_hist(i) = abs(yaw(i));
        else
            yaw_hist(i) = -abs(yaw(i));
        end
    elseif isInside && (abs(yaw(i)) >= hist_max)
        if yaw_hist(i-1)>0
            yaw_hist(i) = abs(yaw(i));
        else
            yaw_hist(i) = -abs(yaw(i));
        end
    else
        isInside =false;
    end
end

%% Definir offset y deltaT
A_offset = csvread('accelerometer_offset.txt');
Ax_offset    = A_offset(:,1);
Ay_offset    = A_offset(:,2);
Az_offset_without_gravity = A_offset(:,3)+1000; %Offset de gravedad (1g);
deltaT = 1/17;
roll_offset     = roll_hist(1:200);
pitch_offset    = pitch_hist(1:200);
yaw_offset      = yaw_hist(1:200);

Ax_mean = mean(Ax_offset);
Ay_mean = mean(Ay_offset);
Az_mean = mean(Az_offset_without_gravity);
roll_mean = mean(roll_offset);
pitch_mean = mean(pitch_offset);
yaw_mean = mean(yaw_offset);

%Calcular vector de tiempo de la muestra.
time = 0:deltaT:deltaT*(length(Ax)-1);

%% Paso 1: Eliminar offset DC y overflow de angulos
Ax_1 = Ax - Ax_mean;
Ay_1 = Ay - Ay_mean;
Az_1 = Az - Az_mean;
roll_1 = roll_hist - roll_mean;
pitch_1= pitch_hist - pitch_mean;
yaw_1  = yaw_hist - yaw_mean;
```

```
for i = 2:length(roll_1)
    if roll_1(i)> 180
        roll_1(i)= roll_1(i)-360;
    elseif roll_1(i)< -180
        roll_1(i) = roll_1(i)+360;
    end
end

for i = 2:length(pitch_1)
    if pitch_1(i)> 180
        pitch_1(i)= pitch_1(i)-360;
    elseif pitch_1(i)< -180
        pitch_1(i) = pitch_1(i)+360;
    end
end

for i = 2:length(yaw_1)
    if yaw_1(i)> 180
        yaw_1(i)= yaw_1(i)-360;
    elseif yaw_1(i)< -180
        yaw_1(i) = yaw_1(i)+360;
    end
end


%% Paso 2: Filtro pasa bajo para se ales: atenuacion de ruido.
acceleration_b = (1/low_filter_size_acceleration)*ones(1,
    low_filter_size_acceleration);
angles_b = (1/low_filter_size_angles)*ones(1,low_filter_size_angles);

Ax_2 = filter(acceleration_b,1,Ax_1);
Ay_2 = filter(acceleration_b,1,Ay_1);
Az_2 = filter(acceleration_b,1,Az_1);
roll_2 = filter(angles_b,1,roll_1);
pitch_2 = filter(angles_b,1,pitch_1);
yaw_2 = filter(angles_b,1,yaw_1);

%% Paso 3: Quitar la componente de aceleracion de gravedad
gravity_Ax = 1000*sin(pitch_2*pi/180);
gravity_Ay = 1000*sin(roll_2*pi/180);
gravity_Az = 1000*(cos(roll_2*pi/180)+ cos(pitch_2*pi/180)-1 );

%% Resto de aceleracioes de gravedad para obtener la aceleracion experimentada
    por el dispositivo
Ax_3 = (Ax_2 + gravity_Ax)*9.8/1000;
Ay_3 = (Ay_2 + gravity_Ay)*9.8/1000;
Az_3 = (Az_2 + gravity_Az)*9.8/1000;


%% Paso 4: Filtro pasa bajo para acelereracion final
acceleration_b2 = (1/10)*ones(1,10);
Ax_4 = filter(acceleration_b2,1,Ax_3);
Ay_4 = filter(acceleration_b2,1,Ay_3);
Az_4 = filter(acceleration_b2,1,Az_3);

%% Paso 5: Filtro pasa alto para integracion
```

```
fc = 0.1/30; %Cut off Frequency
order = 6; %6th Order Filter
[b2 a2] = butter(order,fc,'high');

Ax_4f = filtfilt(b2,a2,Ax_4);
Ay_4f = filtfilt(b2,a2,Ay_4);
Az_4f = filtfilt(b2,a2,Az_4);

%%Paso 6: Primera Integracion (Aceleracion -> Velocidad)
velX = cumtrapz(time,Ax_4f);
velY = cumtrapz(time,Ay_4f);
velZ = cumtrapz(time,Az_4f);

%%Paso 7: Filtro pasa alto para integracion
fc = 0.1/30; %Cut off Frequency
order = 6; %6th Order Filter
[b2 a2] = butter(order,fc,'high');

velX_2 = filtfilt(b2,a2,velX);
velY_2 = filtfilt(b2,a2,velY);
velZ_2 = filtfilt(b2,a2,velZ);

%%Paso 8: Segunda Integracion (Velocidad -> Desplazamiento)
displacement=zeros(length(Ax),3);
displacement(:,1) = cumtrapz(time,velX_2);
displacement(:,2) = cumtrapz(time,velY_2);
displacement(:,3) = cumtrapz(time,velZ_2);

%%Paso 9: Escribir desplazamiento y  ngulos  filtrados en un .txt
angles=zeros(length(Ax),3);
angles(:,1) = roll_2;
angles(:,2) = pitch_2;
angles(:,3) = yaw_2;

data=horzcat(displacement,angles);
fileID = fopen('test_output.txt','w');
fprintf(fileID,'%f,%f,%f,%f,%f,%f\n',data');
fclose(fileID);
```

8.6. Código C Sharp para simular desplazamientos en Unity3D

8.6.1. Código principal para trasladar y rotar

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;
using System.IO;
```

```
public class CarController : MonoBehaviour {

  protected FileInfo theSourceFile = null;
  protected StreamReader reader = null;
  protected string text = " "; // assigned to allow first line to be read
   below
  float x=0;
  float y=0;
  float z=0;
  float previousX;
  float previousY;
  float previousZ;

  void Start () {
    theSourceFile = new FileInfo ("Assets/test_output.txt");
    reader = theSourceFile.OpenText();
    transform.position = new Vector3(0f,0.05f,0f); //Origen de auto en
   simulador.
  }

  void FixedUpdate () {
    //Lectura en cada frame de una linea del archivo .txt, i.e, acel y angulos
   .
    text = reader.ReadLine();
    if (text != null) {
      string[] data = text.Split(',');
      previousX = x;
      previousY = y;
      previousZ = z;
      x = float.Parse(data[0])/100;
      y = float.Parse(data[1])/100;
      z = float.Parse(data[2])/100;
      float roll = float.Parse(data[3]);
      float pitch = float.Parse(data[4]);
      float yaw = float.Parse(data[5]);

      //Posicionar segun angulos y trasladar segun desplazamiento.
      transform.eulerAngles = new Vector3(pitch, -yaw, roll);
      transform.Translate( y-previousY, 0.0f, x-previousX,Space.Self);

    }
  }
}
```

8.6.2. Código para posicionar cámara según movimiento del auto

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CameraFollow : MonoBehaviour {
```

```
6
    public Transform target;      // Posicion a seguir
8   public float smoothing = 5f; // Velocidad de la camara
    Vector3 offset;               // Offset inicial del objeto
10


12  void Start ()
    {
14    // Calcular el offset inicial
      offset = transform.position - target.position;
16  }


18
    void FixedUpdate ()
20  {
      //Crear la posicion de la camara segun la posicion del objeto
22    Vector3 targetCamPos = target.position + offset;

24    //Mover camara transitoriamente.
      transform.position = Vector3.Lerp (transform.position, targetCamPos,
     smoothing * Time.deltaTime);
26  }
}
```