



**Universidad
de La Laguna**

**ESCUELA SUPERIOR DE INGENIERÍA
Y TECNOLOGÍA**

TRABAJO DE FIN DE GRADO:

Sistema de Visión Artificial para Vehículos.

**Grado en Ingeniería Electrónica Industrial y
Automática**

Autor:

Gabriela Ibáñez Agea

Tutor:

Santiago Torres Álvarez

Fecha: 2021 - 2022

AGRADECIMIENTOS

Con la realización de este Trabajo de Fin de Grado quisiera agradecer a todas aquellas personas que me han mostrado su apoyo durante la carrera.

Agradecer a mis familiares, especialmente a mi madre, a mi abuela y a mi abuelo, por motivarme cada día e inculcarme desde el inicio la importancia de obtener una buena formación y garantizar así un futuro como profesional.

Agradecer a mis compañeros, en especial a Eladio Francisco Andrea, Daniel Miguel Basterrechea Rodríguez y Laura Suárez Afonso, por todas esas horas dedicadas al estudio y toda esa ayuda y apoyos brindados en los momentos de dificultades.

Agradecer al profesorado por su implicación, en especial a Santiago Torres Álvarez por su tutorización durante este trabajo y a Benjamín González Díaz por su apoyo y orientación durante la carrera.

Finalmente, agradecer a la Universidad de La Laguna, por otorgarme la oportunidad de formarme y convertirme en Ingeniera Técnico Industrial.

RESUMEN

En la actualidad, el empleo de vehículos para el transporte de las personas ha adquirido una gran importancia, volviéndose algo indispensable para el día a día de la gran mayoría. Desde su creación, se han tratado de realizar mejoras para contribuir a la seguridad de sus ocupantes con el fin de prevenir accidentes de tráfico, así como reducir la magnitud de los mismos en el caso de que se produjeran.

De esta manera, a lo largo de los años se han ido incluyendo medidas y dispositivos que contribuyen a este fin, logrando cada vez una mayor seguridad al volante. No obstante, a día de hoy las cifras de accidentes con víctimas, sean mortales o no, se mantienen en una tendencia constante y con poca previsión de disminuir.

Es en este punto donde toman relevancia nuevos dispositivos que puedan contribuir a disminuir estas cifras, aspecto del que surge la idea de realización del proyecto de este Trabajo de Fin de Grado. En este, se plantea el diseño de un sistema de grabación para vehículos orientado a la detección de impactos, de manera que se contribuya al esclarecimiento de los hechos y culpabilidades en caso de producirse un accidente de tráfico. Además, uno de los objetivos es que el diseño final sea de bajo coste, de forma que pueda ser implantado en una amplia flota de vehículos, favoreciendo así su uso como una herramienta común para aumentar la seguridad vial. El propio diseño del producto como prototipo será un primer paso de cara a su posible homologación para su uso comercial posterior.

En esta memoria, se ha tratado de realizar un estudio y diseño del sistema propuesto que, acompañado de una implementación simplificada de las funciones básicas que debería cubrir el dispositivo, demuestra las utilidades y aportaciones que se podrían obtener con su implementación en vehículos. Además, y aprovechando aún más las funcionalidades que pueden ofrecer los dispositivos y componentes empleados en el diseño, se realiza un estudio de las posibles líneas abiertas futuras del sistema planteado, considerando proyectos de mejora y ampliación que logren dotar de interesantes contribuciones al incremento de la seguridad vial.

ABSTRACT

Nowadays, the use of vehicles for the transport of people has acquired great importance, becoming something indispensable for the day of the majority. Since its creation, professionals have tried to make improvements to contribute to the safety of vehicles occupants in order to prevent traffic accidents, as well as reduce the magnitude of them whether they occur.

In this way, over the years different measures and control or monitoring devices that contribute to this purpose have been included, achieving a greater safety at the wheel. However, today the numbers of accidents with victims, whether fatal or not, remain in a constant trend and with little forecast of decreasing.

At this point, new devices that can contribute to reducing these numbers become relevant. This is the main idea to carry out the project of this Final Degree Project. In this document, the design of a recording system for vehicles oriented to the detection of impacts is proposed, so that it contributes to the clarification of the facts and culpabilities in case of a traffic accident.

In this report, it has tried to make a study and design of the proposed system, which accompanied by a simplified implementation of the basic functions that the device should cover, demonstrates the utilities and contributions that could be obtained with its implementation in vehicles. In addition, and taking even more advantage of the functionalities that can offer the devices and components used in the design, a study of the possible future open lines of the proposed system is carried out, considering interesting improvement and expansion projects that manage to increase the road safety.

ÍNDICE GENERAL

AGRADECIMIENTOS	1
RESUMEN	2
ABSTRACT	3
MEMORIA.....	11
1. INTRODUCCIÓN.....	12
1.1 PREÁMBULO.....	12
1.2 JUSTIFICACIÓN DE REALIZACIÓN DEL PROYECTO.....	15
1.3 OBJETIVOS PROPUESTOS.....	16
1.4 ESTRUCTURA DEL DOCUMENTO.....	17
2. CONCEPTOS PREVIOS AL DISEÑO	20
2.1 SISTEMAS DE GRABACIÓN PARA VEHÍCULOS. DASH CAM	20
2.2 RASPBERRY PI	21
2.2.1 DESCRIPCIÓN Y APLICACIONES	22
2.2.2 HARDWARE Y ESPECIFICACIONES	22
2.2.3 PUERTOS Y PINES DE COMUNICACIÓN.....	25
2.2.4 SOFTWARE Y LENGUAJES DE PROGRAMACIÓN	30
2.3 COMPONENTES ADICIONALES.....	35
2.3.1 CÁMARA. MÓDULO DE CÁMARA RASPBERRY PI V2	36
2.3.1 UNIDAD DE MEDIDA INERCIAL. MPU 6050.....	38
2.3.2 MÉTODO DE ALIMENTACIÓN	42
2.3.3 MÉTODO DE ALMACENAMIENTO.....	44
3. IMPLEMENTACIÓN Y ANÁLISIS DE RESULTADOS.....	46
3.1 CONSIDERACIONES PREVIAS	46
3.1.1 PROBLEMAS SURGIDOS Y SOLUCIÓN APLICADA.....	46

3.2	MONTAJE DEL MODELO.....	50
3.2.1	MONTAJE Y CONEXIONADO. RASPBERRY PI	50
3.2.2	MONTAJE Y CONEXIONADO. ARDUINO.....	53
3.3	CÓDIGOS EMPLEADOS	56
3.3.1	GRABACIÓN EN BUCLE CONDICIONADA.....	58
3.3.2	LECTURA DEL SENSOR.....	67
3.3.3	CREACIÓN DE FICHEROS DE DATOS	71
3.3.4	CALIBRACIÓN DEL SENSOR.....	73
4.	LÍNEAS ABIERTAS Y FUTURAS.....	75
4.1	PROPUESTAS DE MEJORA.....	75
4.1.1	METODOLOGÍA DE TOMA DE IMÁGENES.....	75
4.1.2	AJUSTE DE PARÁMETROS.....	76
4.2	PROYECTOS DE AMPLIACIÓN	78
4.2.1	ADICIÓN DE GRABACIÓN INTERNA O EXTERNA TRASERA.....	78
4.2.2	ALMACENAMIENTO EN NUBE	80
4.2.3	CONTROL POR VOZ	81
4.2.4	TRATAMIENTO DE IMÁGENES.	82
5.	CONCLUSIONES / CONCLUSIONS.....	84
6.	PRESUPUESTO DEL MODELO A IMPLEMENTAR.....	85
7.	BIBLIOGRAFÍA	86
	ANEXO I.	93
	ESTUDIO DE MERCADO DEL PRODUCTO EN LA ACTUALIDAD	93
	ANEXO II.....	102
	ESQUEMÁTICO DEL CONEXIONADO ELÉCTRICO DEL MODELO.	102
	ANEXO III.	105

CÓDIGO DE PROGRAMACIÓN DE LAS FUNCIONES BÁSICAS DEL SISTEMA PROPUESTO.	105
ANEXO IV.	111
CÓDIGO EMPLEADO PARA LA CALIBRACIÓN DEL MPU6050.	111
ANEXO V.	119
CÓDIGO EMPLEADO PARA JUSTIFICACIÓN DE ERROR COMETIDO CON LA PÉRDIDA DE FOTOGRAMAS.	119
ANEXO VI.	121
DATASHEET DE MPU-6050.	121
ANEXO VII.	144
DOCUMENTACIÓN GRÁFICA	144

ÍNDICE DE TABLAS

Tabla I. Comparativa de especificaciones entre los distintos modelos de Raspberry Pi	23
Tabla II. Especificaciones del Módulo de Cámara de Raspberry Pi V2.....	38
Tabla III. Especificaciones del MPU6050.....	40
Tabla IV. Valores recomendados de alimentación para distintos modelos de Raspberry Pi	42

ÍNDICE DE FIGURAS

Figura 1. Esquema estadístico extraído del Anuario Estadístico de Accidentes de 2020 proporcionado por la Dirección General de Tráfico [1].....	13
Figura 2. Gráfico comparativo del número de fallecidos anual desde 1960 hasta 2020, obtenido del Anuario Estadístico de Accidentes de 2020 proporcionado por la Dirección General de Tráfico [1].....	14
Figura 3. Gráfico comparativo del número de víctimas anual desde 1960 hasta 2020, obtenido del Anuario Estadístico de Accidentes de 2020 proporcionado por la Dirección General de Tráfico [1].....	14
Figura 4. Esquemático de pines de la Raspberry Pi 3 Model B, obtenida de la documentación oficial proporcionada por RaspberryPi.com [12].....	26
Figura 5. Representación de bits de una señal de comunicación UART, obtenida de fuente [14].....	27
Figura 6. Representación de las señales implicadas en el protocolo de comunicación I2C, obtenida de fuente [17]	29
Figura 7. SO proporcionados por “Raspberry Pi Imager” clasificados como de propósito general	31
Figura 8. SO proporcionados por “Raspberry Pi Imager” orientados a la reproducción multimedia.....	31
Figura 9. SO proporcionados por “Raspberry Pi Imager” orientados al uso de emuladores de consolas.	32
Figura 10. SO proporcionados por “Raspberry Pi Imager” orientados a la impresión 3D	32
Figura 11. SO proporcionados por “Raspberry Pi Imager” orientados a asistentes y sistemas de automatización domésticos	33
Figura 12. SO proporcionado por “Raspberry Pi Imager” orientado al empleo de un software de señalización digital.....	33

Figura 13. SO proporcionados por “Raspberry Pi Imager” orientados a determinados usos específicos.....	34
Figura 14. Representación de distancia focal y ángulo de visión, obtenida de [26]	37
Figura 15. Movimientos de cabeceo, alabeo y guiñada, obtenida de [30]	39
Figura 16. Movimiento de precesión realizado por un giróscopo, obtenida de [31]	40
Figura 17. Esquemático del GY-521, obtenida de [33].....	41
Figura 18. Comprobación de dispositivos detectados por comunicación i2c, con el resultado de la no detección del sensor.	47
Figura 19. Raspberry Pi 3 Model B.....	51
Figura 20. Módulo de Cámara Raspberry Pi V2	51
Figura 21. Tarjeta microSDXC SanDisk 64GB	51
Figura 22. Fuente de alimentación, 5V y 2,4 A.....	52
Figura 23. Cable Ethernet y HDMI.	52
Figura 24. Periféricos conector USB.....	52
Figura 25. Montaje final con Raspberry Pi.....	53
Figura 26. Arduino Mega	54
Figura 27. Módulo GY-521, con sensor MPU6050	54
Figura 28. Pines para el conexionado	54
Figura 29. Cables para el conexionado.....	55
Figura 30. Cable alimentación Arduino desde portátil.....	55
Figura 31. Montaje final con Arduino Mega	56
Figura 32. Librerías empleadas en el bucle de grabación condicionado.....	58
Figura 33. Bucle de grabación condicionada.....	60
Figura 34. Grabación del clip posterior a la detección del impacto	61

Figura 35. Código de una sola ejecución del hilo detector de impacto.....	62
Figura 36. Bucle detector de impacto del hilo secundario	63
Figura 37. Líneas de código previas a la entrada al bucle de grabación	64
Figura 38. Conversión de los clips obtenidos a formato .mp4	65
Figura 39. Obtención del archivo de video final, compuesto por los clips grabados.....	66
Figura 40. Librerías empleadas para la lectura del sensor.....	68
Figura 41. Inicialización de variables para la lectura del sensor.....	68
Figura 42. Acciones ejecutadas previas a la lectura del sensor.....	69
Figura 43. Bucle de lectura del sensor.....	70
Figura 44. Librerías empleadas para la creación de los ficheros de datos	71
Figura 45. Creación de los archivos y establecimiento de la comunicación serial 71	
Figura 46. Bucle de escritura de los valores obtenidos por comunicación serial.	72
Figura 47. Estructura del fichero de datos escrito	72
Figura 48. Valores modificados para la calibración	74
Figura 49. Offset tras la calibración	74

MEMORIA

1. INTRODUCCIÓN

1.1 PREÁMBULO

En la actualidad en nuestro país se producen accidentes de tráfico diariamente, cuya magnitud abarca daños que comprenden desde consecuencias personales leves hasta graves, llegando a alcanzar en gran medida defunciones o situaciones de gravedad en la salud de las personas implicadas en el mismo.

La página web oficial de la Dirección General de Tráfico proporciona distintas estadísticas de las cifras recogidas diariamente y anualmente, además de otros tipos de datos y comparativas relacionados. El Anuario Estadístico de Accidentes [1] del último año que proporcionan en su página, siendo este el 2020, establece que se han producido un total de 72.959 accidentes con víctimas involucradas, de los cuales han resultado con víctimas mortales una cantidad de 1.275 accidentes. En adición, en el año 2020, se han producido un total de 94.562 personas heridas en accidentes de tráfico, y un total de 1.370 fallecidos. Estas cifras, se muestran desglosadas en detalle en el esquema de la Figura 1, extraído de la misma fuente.

La fuente citada proporciona además una serie de gráficos comparativos de los datos estudiados desde 1960 hasta 2020, los cuáles se pueden apreciar en las Figuras 2 y 3.

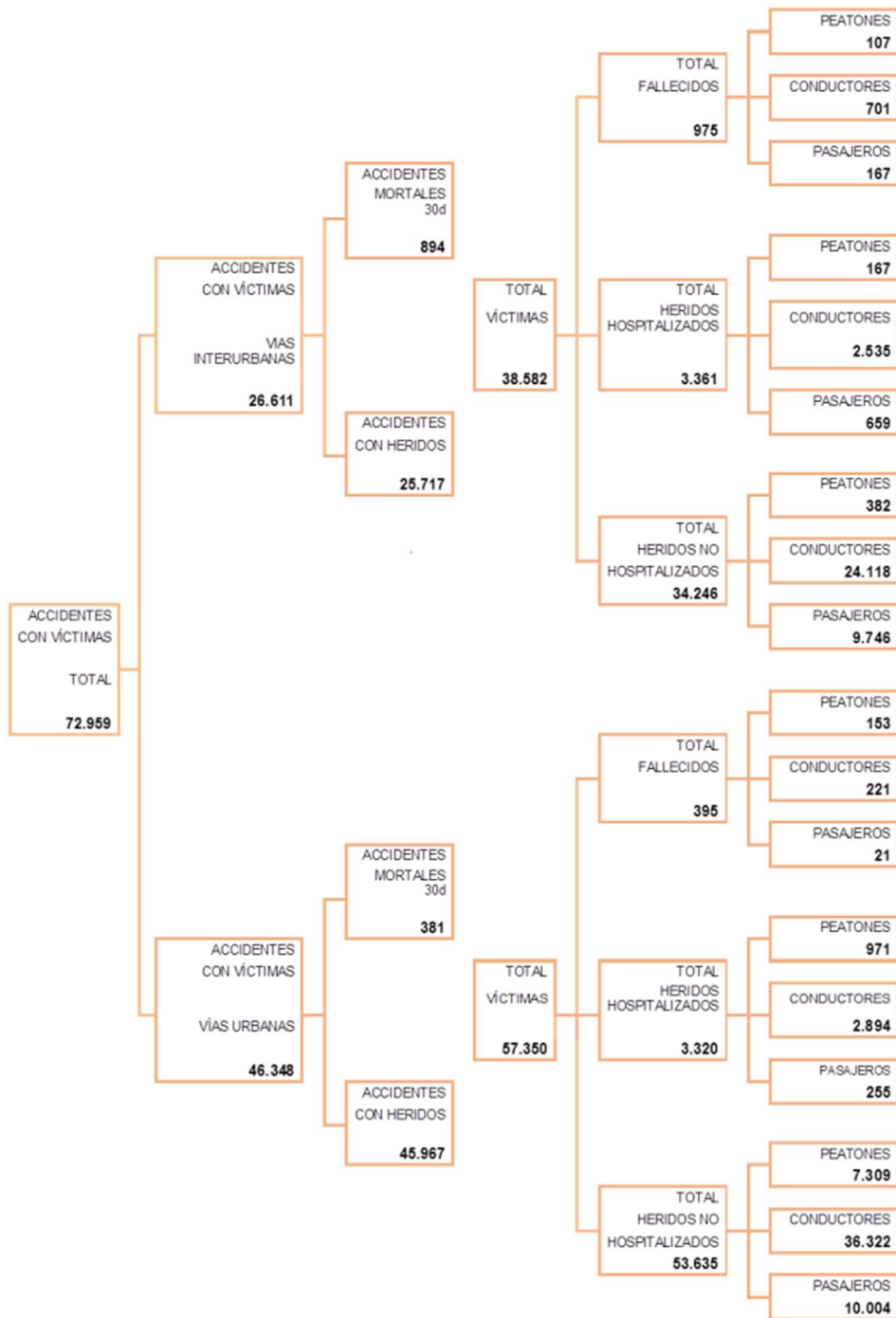


Figura 1. Esquema estadístico extraído del Anuario Estadístico de Accidentes de 2020 proporcionado por la Dirección General de Tráfico [1]

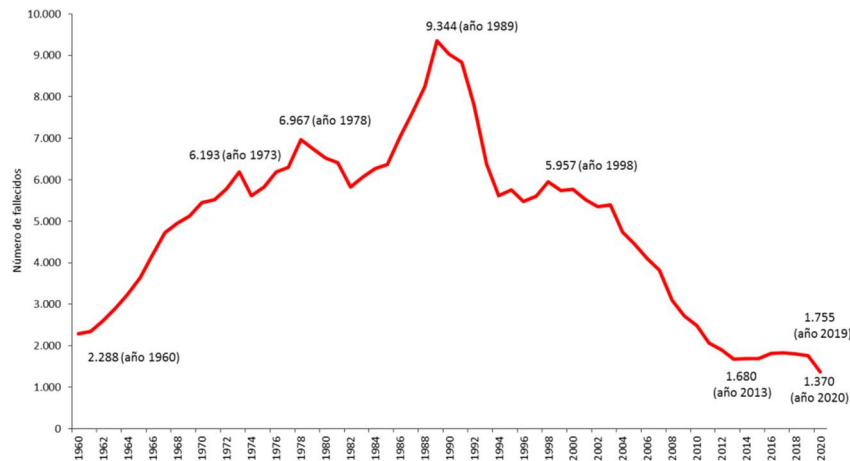


Figura 2. Gráfico comparativo del número de fallecidos anual desde 1960 hasta 2020, obtenido del Anuario Estadístico de Accidentes de 2020 proporcionado por la Dirección General de Tráfico [1]

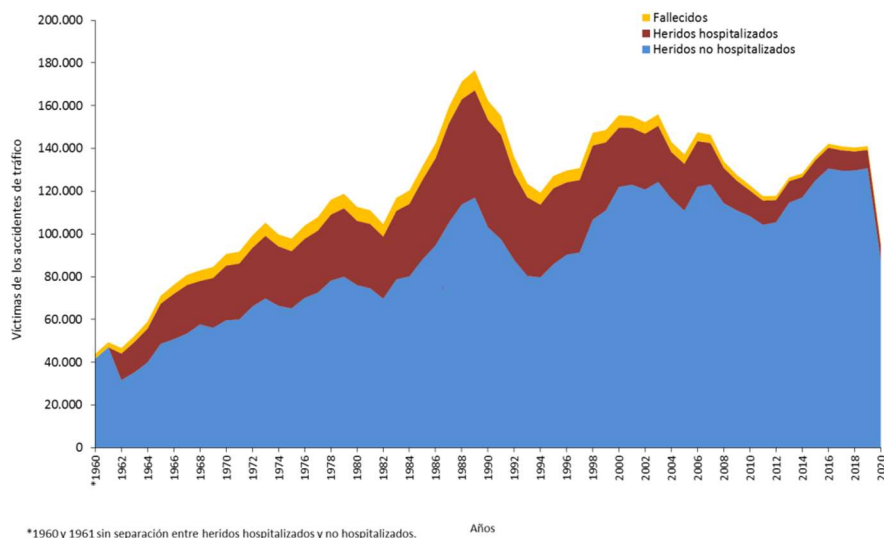


Figura 3. Gráfico comparativo del número de víctimas anual desde 1960 hasta 2020, obtenido del Anuario Estadístico de Accidentes de 2020 proporcionado por la Dirección General de Tráfico [1]

Si bien la cifra de fallecidos producidos en accidentes de tráfico se ha visto reducida desde la década de los 90s, situándose estos últimos años en los valores mínimos obtenidos en todo el estudio, el número de víctimas producidos en accidentes de tráfico ha incrementado y muestra una tendencia constante desde 2016. No obstante, para ambas gráficas se ha presentado un pico mínimo en el último año de estudio, el 2020, cuya causa probable se debiera a la situación de confinamiento producido en

España comprendido en las fechas del 15 de marzo al 21 de junio del 2020, derivado de la pandemia sufrida mundialmente a causa del Covid-19.

Los datos observados de las gráficas corroboran la importancia de la aplicación de medidas de seguridad en los vehículos, pues la aparición de distintos sistemas, como pueden ser la obligatoriedad del uso de cinturón de seguridad en todos los asientos desde el año 1992, la obligatoriedad de implementar, en el momento de fabricación de vehículos, el sistema de frenado ABS desde el año 2004 y el airbag desde el año 2006 [2], entre otras mejoras; han contribuido a la reducción de las cifras contempladas previamente. En adición, con el paso de los años se han ido mejorando distintos aspectos en el diseño de vehículos, así como en las pruebas y estudios de seguridad realizados, además de un mayor control y concienciación del cumplimiento de las normas de circulación vial por parte de los usuarios. Todas estas medidas han sido de ayuda a la hora de reducir la cantidad de víctimas mortales, pero aún se continúan produciendo grandes cifras de accidentes de tráfico que impliquen víctimas de distintas gravedades.

Es en este punto en el que toman relevancia nuevos dispositivos a implementar en vehículos para contribuir a la seguridad de los mismos, a la ayuda de la conducción, a la emisión de alertas al conductor ante imprevistos en la conducción, a la aportación de evidencias tras un accidente de tráfico, etc.

1.2 JUSTIFICACIÓN DE REALIZACIÓN DEL PROYECTO

Derivado de la situación contemplada en el preámbulo expuesto en el apartado anterior, este proyecto y la realización del presente Trabajo de Fin de Grado nace con el propósito de lograr el diseño e implementación de un sistema que sea capaz de aportar evidencias en aquellos casos en los que se produzca un accidente de tráfico, de manera que estas ayuden a esclarecer los hechos ocurridos.

Se busca lograr la creación de un sistema de visión artificial de vehículos orientado a la toma de imágenes correspondientes a los momentos anteriores y posteriores a la detección de un impacto, de manera que se disponga de evidencias

gráficas de las causas del accidente, así como datos que puedan resultar relevantes, como pueden ser la velocidad del vehículo o la localización de los hechos, entre otros.

Adicionalmente, se pretende realizar la implementación de un modelo del sistema propuesto que resulte en un proyecto económico, aunque ello suponga la rebaja las funcionalidades a implementar, y que pueden proveer otros sistemas presentes en el mercado, a las funcionalidades básicas requeridas para el fin contemplado.

No obstante, a pesar de que este Trabajo de Fin de Grado está orientado a la realización de un modelo funcional básico, igualmente se contemplarán posibles proyectos de mejora y de ampliación que logren la adición de otras funcionalidades que puedan resultar de interés.

1.3 OBJETIVOS PROPUESTOS

Con la realización del proyecto propuesto en el presente Trabajo de Fin de Grado se pretende lograr una serie de objetivos, los cuales se pueden clasificar como objetivos generales, específicos y personales.

El objetivo general de la realización de este trabajo es lograr el diseño de un sistema de visión artificial para vehículos orientado a la toma de imágenes que sirvan de justificación en caso de impacto o accidente, adquiriendo y aplicando los conocimientos y documentación que se requiera para ello. Adicionalmente, se pretende justificar este diseño mediante la realización de una implementación simplificada. De igual manera, será de interés realizar una evaluación al proyecto y resultados obtenidos, así como otorgar propuestas de mejora y de ampliación al sistema realizado.

Los objetivos específicos se listan a continuación:

- Adquirir conocimientos sobre el funcionamiento de los Sistemas de Visión Artificial para Vehículos que se encuentran en la actualidad.
- Realizar el diseño e implementación simplificada de un modelo empleando como elemento principal una Raspberry Pi.

- Comprender el funcionamiento de la Raspberry Pi, en lo referente a hardware y software.
- Escoger aquellos elementos y componentes compatibles con la Raspberry Pi; y establecer comunicaciones entre ellos para lograr el correcto traspaso y tratamiento de los datos.
- Adquirir conocimientos específicos sobre los distintos campos de aplicación: electrónica e informática orientada a la programación, en su mayoría.
- Realizar una correcta interpretación de la documentación empleada en la realización de este proyecto, más concretamente los *Datasheet* de los componentes.
- Solventar aquellos imprevistos y problemas que surjan durante la implementación del modelo.

Es necesario destacar que este proyecto está orientado únicamente a la realización de un modelo que satisfaga las funcionalidades básicas de un Sistema de Visión Artificial para Vehículos, en relación con su diseño e implementación. Es por ello que no se realizará un estudio ni aplicación de normativa referente a estos tipos de sistemas, en cuanto a la toma de imágenes en vía pública se refiere.

En relación a los objetivos personales, se pretende lograr la realización de un primer proyecto, aplicando y adquiriendo para ello los conocimientos que resulten necesarios. En adición, se pretende realizar el presente Trabajo de Fin de Grado a modo de finalización del periodo académico, siendo este el último trabajo realizado previo a la obtención del título de Ingeniera Técnico Industrial, especializada en Electrónica Industrial y Automática.

1.4 ESTRUCTURA DEL DOCUMENTO

El presente documento se divide en una serie de apartados, en los cuales se detallan las distintas fases que se han seguido para lograr la realización del proyecto propuesto.

De esta manera, se comienza realizando un análisis de los motivos, causas y justificación en relación a la realización de un proyecto que englobe el diseño e implementación de un Sistema de Visión Artificial para Vehículos orientado a la toma de imágenes. Eso aparece comprendido en los subapartados “1.1 Preámbulo”, “1.2 Justificación de Realización del Proyecto” y “1.3 Objetivos Propuestos”, todos incluidos en el primer apartado, denominado “1. Introducción”.

Seguidamente se detallan aquellos conceptos teóricos necesarios para comprender el funcionamiento del sistema propuesto, y se exponen de manera teórica los componentes con los que se pretende lograr el diseño e implementación del proyecto objeto del presente Trabajo de Fin de Grado. Esto se engloba en los subapartados incluidos en el segundo apartado, denominado “2. Conceptos Teóricos Previos al Diseño”. Además, para cada uno de los conceptos descritos se indica el componente, lenguaje de programación o sistema operativo escogido para la implementación.

A continuación, se presenta la aplicación de lo mencionado en apartados anteriores para lograr la implementación simplificada del proyecto propuesto. De manera justificativa, se adjunta documentación gráfica del proceso seguido y de los resultados obtenidos. Todo ello se encuentra expuesto en el apartado denominado “3. Implementación y Análisis de Resultados”.

Para concluir la memoria, se realiza una autoevaluación del proyecto final obtenido y se presentan propuestas de mejora, así como de ampliación de funcionalidades. Esto se puede consultar en el apartado denominado “4. Líneas Abiertas y Futuras”.

Adicionalmente, se adjuntan anexos en los que se detallan en mayor medida aspectos relevantes para la realización del proyecto.

- Anexo I. Estudio de Mercado del Producto en la Actualidad.
- Anexo II. Esquemático del Conexionado Eléctrico del Modelo
- Anexo III. Código de Programación de las Funciones Básicas del Sistema Propuesto.
- Anexo IV. Código Empleado para la Calibración del Modelo.

- Anexo V. Código Empleado para Justificación de Error Cometido con la Pérdida de Fotogramas.
- Anexo VI. Datasheet de los MPU-6050.
- Anexo VII. Documentación Gráfica.

2. CONCEPTOS PREVIOS AL DISEÑO

En este apartado se tratan los conceptos teóricos generales de los componentes y elementos necesarios para el diseño y la implementación de un sistema de visión artificial para vehículos orientado a la toma de imágenes, así como se realiza una introducción del mismo y su situación en la actualidad.

2.1 SISTEMAS DE GRABACIÓN PARA VEHÍCULOS. DASH CAM

En la actualidad, existen distintos sistemas de grabación para vehículos en el mercado, que abarcan distintas calidades, tamaños, funcionalidades y precios.

A estos dispositivos se les atribuye el anglicismo “*Dash Cam*”, cuya traducción literal es “Cámara de Tablero”. Esto se debe a que su ubicación en los vehículos se encuentra usualmente en sus tableros, pudiendo llegar a situarse en la parte interna superior de la luna. Esta ubicación es de gran importancia a la hora de adquirir documentación gráfica clara y sin obstáculos en su campo de visión.

Su uso mayoritario se localiza en Rusia, aunque actualmente se está extendiendo por Estados Unidos, Australia, distintos países de Europa y de Asia. No obstante, se limita su uso en Suiza, Alemania, Australia y Polonia; y se prohíbe su uso en Austria. En España no se prohíbe su uso, pero sí cualquier acción en el dispositivo que pudiera afectar a la seguridad durante la conducción [3].

A causa de esta continua expansión de su uso y popularización del producto, el mercado correspondiente resulta cada vez más competitivo, pues las empresas distribuidoras de estos sistemas buscan añadir funcionalidades innovadoras, a una calidad y precios ajustados, tratando así que el producto destaque entre la amplia gama de la que se dispone actualmente.

Es por ello que se pueden encontrar, a precios reducidos, y por consiguiente calidad de componentes reducida, *Dash Cams* que proporcionen funcionalidades como

pueden ser la adición de GPS, comandos por voz, control de velocidad, grabación interna o exterior trasera, grabación y visualización por *streaming*, guardado de archivos en almacenaje en la nube, protección activa con el coche apagado, entre otras muchas.

Adjuntado en el Anexo I se encuentra un breve estudio de mercado realizado con la finalidad de lograr una documentación previa a la realización del diseño del sistema propuesto, de manera que se adoptaran conocimientos y valores orientativos en relación a este producto. Esta consulta de documentación se realizó en octubre de 2021.

Tras la adquisición de conocimientos sobre el producto que se quiere diseñar e implementar, se ha concluido que los componentes mínimos requeridos para lograr una funcionalidad básica del sistema, objeto del presente Trabajo de Fin de Grado tal y como se expone en el apartado “1.3 *Objetivos Propuestos*”, son una cámara con la que se obtenga documentación gráfica, un sensor con el que se puedan detectar impactos y un dispositivo que permita realizar el control de los elementos comentados.

2.2 RASPBERRY PI

Como elemento principal para el funcionamiento del sistema propuesto, se precisa de un dispositivo capaz de realizar el control de los sensores con los que se obtendrá información externa, comunicación con los mismos y tratamiento y procesamiento de los datos obtenidos.

Para lograr esta funcionalidad existen distintas alternativas: diseño de una placa de circuito impreso a medida o empleo de una placa de circuito impreso prediseñada, como pueden ser los distintos modelos de Raspberry Pi, de Arduino o las muchas alternativas de estos que se encuentran en el mercado.

Debido a la versatilidad que otorga la Raspberry Pi a la hora de realizar cualquier proyecto propuesto que no requiera de gran complicación, se ha decidido emplear este dispositivo como el componente principal de este sistema.

2.2.1 DESCRIPCIÓN Y APLICACIONES

La Raspberry Pi es una placa de circuito impreso (PCB) prediseñada que dispone de diversos componentes electrónicos, capaz de realizar las mismas funcionalidades que un ordenador, pero a un menor precio y tamaño. Fue diseñada y comercializada en el año 2012 en Reino Unido por la fundación sin ánimo de lucro “Raspberry Pi Foundation” y distribuida con el objetivo de fomentar la creación de proyectos y contribuir a la enseñanza de lenguajes de programación e informática en el ámbito educativo [4] [5].

Este dispositivo dispone de un *hardware* y *software* similares al de un ordenador, admitiendo por tanto una gran variedad de lenguajes de programación y sistemas operativos. Se distribuyen diversos modelos compatibles entre ellos, con variaciones en sus especificaciones de forma que se permita ajustar las funcionalidades que otorga la placa a las necesidades de cada proyecto, pudiendo abarcar una gran cantidad de aplicaciones bien sea empleando únicamente la Raspberry Pi o bien añadiendo componentes adicionales [5] [6].

Es por ello que existe un gran abanico de aplicaciones que puede abarcar una Raspberry Pi, entre las cuales podemos encontrar desde proyectos básicos dirigidos al estudio y comprensión tanto de componentes electrónicos como de conceptos informáticos, hasta proyectos de investigación en los que se emplean estos dispositivos para la toma de datos, pasando por otros proyectos usualmente caseros en los que se emplean para la implementación de sistemas de videovigilancia o de consolas y reproductores multimedia, entre otros [5] [6].

2.2.2 HARDWARE Y ESPECIFICACIONES

Debido a la existencia de distintos modelos de Raspberry Pi, se producen diferencias en el *hardware* empleado para cada uno, en ocasiones coincidiendo mismos componentes mientras que en otras variando notoriamente. Es por ello que en este apartado se realiza una visión general y comparativa de los diferentes modelos que se encuentran en el mercado, enfatizando en los componentes más significativos en relación con el proyecto propuesto.

Es necesario aclarar que existen modelos desfasados, por lo que se considerarán únicamente los expuestos en la página oficial en el momento de la realización del estudio (a fecha de abril de 2022). Estos aparecen listados a continuación [7]:

- Raspberry Pi 1 Model A+
- Raspberry Pi 1 Model B+
- Raspberry Pi 2 Model B
- Raspberry Pi 3 Model A+
- Raspberry Pi 3 Model B
- Raspberry Pi 3 Model B+
- Raspberry Pi 4 Model B
- Raspberry Pi Zero
- Raspberry Pi Zero W
- Raspberry Pi Zero 2 W

Los parámetros de estudio empleados para realizar la selección del modelo serán la velocidad y cantidad de núcleos del procesador, que afectarán a la rapidez con la que la Raspberry Pi realice las tareas designadas; la memoria RAM, que determinará la cantidad de operaciones que se pueden realizar de manera simultánea; el tipo de almacenamiento, en el que se almacenará tanto el sistema operativo como las aplicaciones, código y archivos generados y obtenidos a partir de la cámara y sensores; los puertos y pines de entrada y salida, que se emplearán para realizar las conexiones con los componentes adicionales; y los puertos de comunicación disponibles que puedan resultar de interés, como el uso de bluetooth o de WiFi. En la tabla I se puede observar una comparativa de las especificaciones mencionadas relativas a cada modelo de estudio [8] [9] [10].

Tabla I. Comparativa de especificaciones entre los distintos modelos de Raspberry Pi							
Modelo	Núcleos	Velocidad	RAM	Bluetooth	WiFi	USB	Precio
1 Model A+	1	700 MHz	512 MB	No dispone	No dispone	1 puerto 2.0	20 \$
1 Model B+	1	700 MHz	512 MB	No dispone	No dispone	4 puertos 2.0	35 \$

2 Model B	4	900 MHz	1000 MB	No dispone	No dispone	4 puertos 2.0	35 \$
2 Model A+	4	1400 MHz	512 MB	Bluetooth 4.2 (BLE)	Wi-Fi 2.4/5GHz	2 puertos 2.0	25 \$
3 Model B	4	1200 MHz	1000 MB	Bluetooth 4.2 (BLE)	Wi-Fi 2.4/5GHz	4 puertos 2.0	35 \$
3 Model B+	4	1400 MHz	1000 MB	Bluetooth 4.2 (BLE)	Wi-Fi 2.4/5GHz	4 puertos 2.0	35 \$
4 Model B	4	1500 MHz	1000 MB – 8000 MB	Bluetooth 5.0	Wi-Fi 2.4/5GHz	4 puertos 2.0 - 3.0	35 \$ - 55 \$
Zero	1	1000 MHz	512 MB	No dispone	No dispone	0 puertos	5 \$
Zero W	1	1000 MHz	512 MB	Bluetooth 4.1 (BLE)	802.11 b/g/n wireless LAN	0 puertos	10 \$
Zero 2 W	4	1000 MHz	512 MB	Bluetooth 4.2 (BLE)	Wi-Fi 2.4GHz	0 puertos	15 \$

Todos los modelos de Raspberry Pi que se contemplan en el estudio realizado disponen de pines de propósito general (GPIO), destinados a la conexión con módulos y componentes externos; de un conector CSI para la adición de un módulo de cámara, que varía en tamaño para los modelos Zero, Zero W y Zero 2 W, y de un puerto de tarjeta microSD [8] [9] [10].

Por otro lado, disponer de conexión Bluetooth y WiFi puede resultar de especial interés, bien a la hora de facilitar el envío de archivos de grabación que considere el usuario, o bien para añadir mejoras al proyecto propuesto, tales como la reproducción

por *streaming* de las imágenes obtenidas. Asimismo, se pueden emplear las conexiones USB para la adición de funcionalidades como mejoras de proyecto, tales como la adición de una segunda cámara de grabación para obtener imágenes internas del vehículo.

Las mejoras comentadas, así como otras propuestas, se detallarán en el subapartado denominado “4.2 *Propuestas de mejora*”, incluido en el apartado “4. *Conclusiones y Análisis de Resultados*”.

En la actualidad, existe una crisis de semiconductores que afecta al mercado mundialmente, en según qué productos. Uno de los que se han visto más afectados es el mercado de las Raspberry Pi, resultando imposible realizar la compra de este dispositivo.

Debido a este motivo y para lograr la realización de la implementación del sistema propuesto, se recurrió al depósito que presenta la Universidad de La Laguna, siendo la **Raspberry Pi 3 Model B** el único modelo del que disponían. Es por ello que este será el modelo que se empleará en la implementación.

2.2.3 PUERTOS Y PINES DE COMUNICACIÓN

La Raspberry Pi dispone de una serie de pines, cuarenta en total, que permiten realizar comunicación con otros dispositivos que se desee añadir para cumplir nuevas funcionalidades. Estos pines generalmente son comunes para los distintos modelos, presentando en ocasiones leves diferencias [11].

Los pines referentes a la Raspberry Pi 3 Model B, modelo empleado para la implementación del sistema, aparecen representados en la Figura 4, extraída del *datasheet* y documentación obtenidas de la página oficial de Raspberry Pi [12].

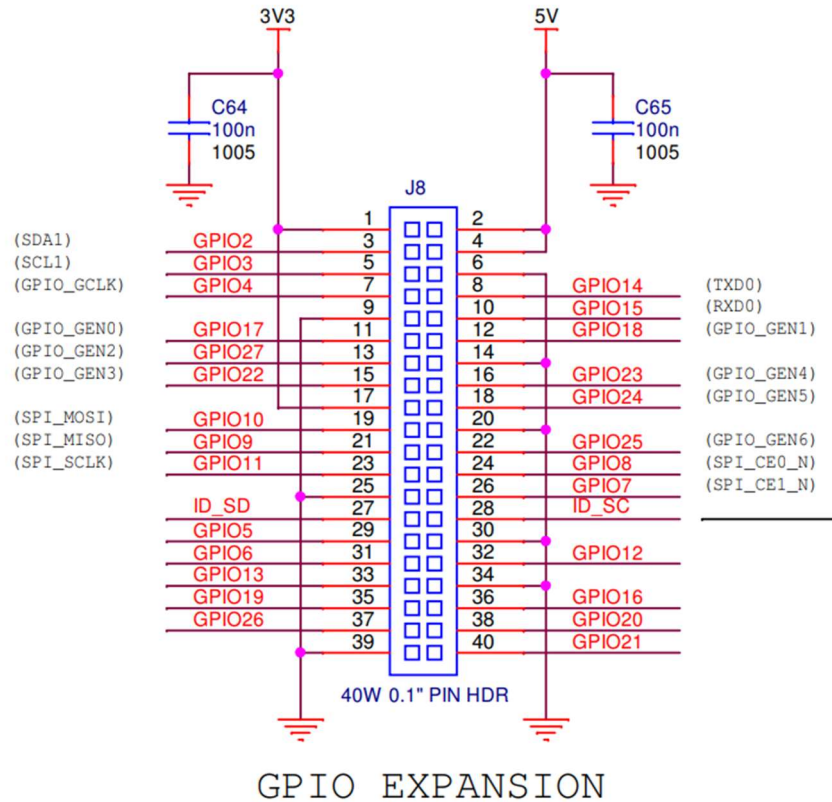


Figura 4. Esquemático de pines de la Raspberry Pi 3 Model B, obtenida de la documentación oficial proporcionada por RaspberryPi.com [12]

En la figura se puede observar la disposición de pines que se listan a continuación. Aparecen representados en **negrita** aquellos relevantes para la realización del proyecto propuesto [13].

- Pines de alimentación de 5V: números 2 y 4.
- **Pines de alimentación de 3,3V**: números 1 y 17.
- **Pines de tierra**: números 6, 9, 14, 20, 25, 30, 34 y 39.
- **Pines de comunicación UART**: números 8 y 10.
- **Pines de comunicación I2C**: números 3, 5, 27 y 28.
- Pines de comunicación SPI: números 19, 21, 23, 24 y 26.
- Pines de modulación de pulso PCM: números 12, 35, 38 y 40.

Los protocolos de comunicación no destacados en **negrita** no se tratarán en este proyecto debido al ámbito de estudio del mismo. De esta forma, los empleados se explican brevemente a continuación:

- **Comunicación UART [14]:**

La UART (*Universal Asynchronous Receiver - Transmitter*) se trata de un tipo de comunicación en serie asíncrona, por lo que tanto el emisor (denominado como Tx) como el receptor (denominado como Rx) deben estar configurados a una misma velocidad de bits, cuya unidad se denomina baudios, para garantizar la correcta transmisión de la información.

Generalmente, el estado de reposo se representa manteniendo la señal en alta. Dado que se trata de una comunicación asíncrona, careciendo por ello de señal de reloj compartida, para indicar el comienzo del envío de información es necesario emplear un bit de inicio, que pasa del estado de reposo en alta a baja, y viene seguido de los bits de datos, que transmiten la información. De igual manera, es necesario emplear un bit de parada que indique el final del envío de datos. Este se reconoce o bien por el paso al estado de reposo en alta o bien por la permanencia en alta durante un tiempo de un bit.

De manera adicional, se puede emplear un bit de paridad con el objetivo de detectar errores. Con este bit, el emisor indica al receptor si los bits de datos disponen de una cantidad de unos par o impar, permitiendo de esta forma corroborar si esto coincide con lo recibido por el receptor.

En la Figura 5 se representa en detalle la composición de bits que dispone una señal transmitida mediante comunicación UART.



Figura 5. Representación de bits de una señal de comunicación UART, obtenida de fuente [14]

Se puede observar cómo el bit de inicio viene representado con la señal en baja, mientras que el de parada en alta. Entre ellos se encuentran los bits de datos, con la información que se desea transmitir. Si se deseara añadir el bit de paridad, éste aparecería representado previo al bit de parada, bien en baja o en alta según se desee indicar paridad par o impar, respectivamente.

En el proyecto propuesto, este tipo de comunicación se emplearía con la adición del módulo de GPS. No obstante, se ha empleado de manera adicional para establecer la comunicación entre el Arduino Mega y el ordenador portátil para la toma de datos del sensor y escritura de archivos de texto correspondientes. El motivo de esto se justificará posteriormente en su apartado correspondiente.

- **Comunicación I2C [15] [16] [17]:**

La I2C se trata de un tipo de comunicación en serie síncrona, que se realiza entre dos o más dispositivos denominados maestro y esclavos, siendo estos últimos normalmente sensores.

Lo que caracteriza a estos dispositivos es que ambos pueden enviar y recibir datos, mientras que únicamente el maestro puede modificar la señal de reloj, es decir, es quién decide cuándo iniciar y finalizar la comunicación. Esto permite la posibilidad de establecer conexión con diversos dispositivos, siempre y cuando sea con uno cada vez.

Al tratarse de un método de comunicación síncrono, se disponen de dos señales: la de reloj (SCL) y la de datos (SDA). Dado que el maestro es quién controla la comunicación, resulta necesario establecer una serie de bits que dispongan de distintas informaciones. En primer lugar, se establecen al comienzo y al final de la comunicación un pin de inicio y un pin de parada. Seguidamente, el maestro indica mediante una serie de bits la dirección del esclavo con el que desea establecer conexión, así como si desea realizar una lectura (bit en alta) o una escritura (bit en baja) del mismo. Cabe destacar que

cada esclavo dispondrá de una dirección distinta. Posterior a ello, se establece un bit para corroborar la correcta conexión con el esclavo (ACK), y una vez ésta es comprobada se procede con el envío de información, que continúa con otro bit que afirma su correcta recepción o envío, según sea escritura o lectura, y se finaliza con un bit de parada que establece la señal en modo de reposo, es decir, en alta.

En la Figura 6 se muestra a modo de ejemplo la forma de onda que tendrían las señales para una comunicación I2C en la que se desee realizar una escritura del esclavo.

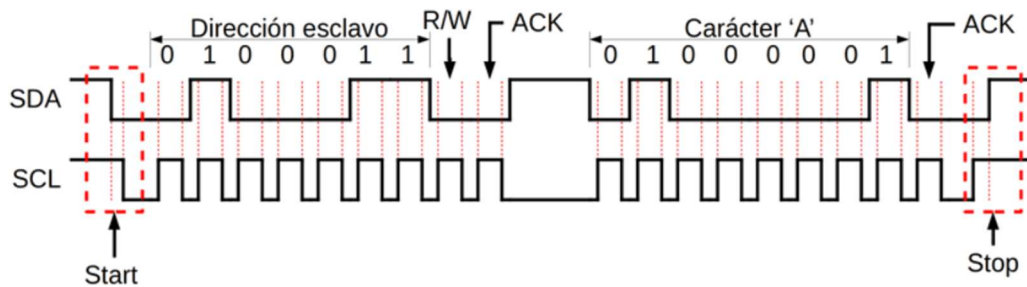


Figura 6. Representación de las señales implicadas en el protocolo de comunicación I2C, obtenida de fuente [17]

Se observa cómo, para iniciar la señal se precisa que ambas señales estén en baja, en primer lugar, el SDA y posterior a ella el SCL. A continuación, se indica la dirección del esclavo y la orden de escritura del mismo, mediante la señal en baja. Tras la confirmación recibida con el bit ACK, se procede al envío de la información, en este caso el carácter A, y se confirma su correcta recepción mediante el bit ACK. Para determinar el final de la comunicación se precisa que ambas señales se mantengan en estado alta, en primer lugar la de SCL y seguidamente la de SDA.

Este tipo de comunicación se empleará con la adición del módulo que comprende la unidad de medida inercial, tratado en mayor detalle en los apartados referentes a la misma.

2.2.4 SOFTWARE Y LENGUAJES DE PROGRAMACIÓN

Para poder emplear la Raspberry Pi en el proyecto propuesto, resulta de vital importancia escoger un sistema operativo con el que trabajar. Además, también resulta importante escoger un lenguaje de programación con el que realizar el código del programa del funcionamiento del sistema propuesto, el cuál será ejecutado por la Raspberry Pi.

Tal y como se comentó en la introducción de este apartado, la Raspberry Pi proporciona versatilidad a la hora de emplear distintos sistemas operativos y, como se verá posteriormente en el presente apartado, a la hora de emplear distintos lenguajes de programación en la realización de programas.

En relación a los sistemas operativos que se pueden emplear en este dispositivo, destacan los proporcionados por la página oficial. En esta, se recomienda el empleo de un programa denominado “Raspberry Pi Imager” [18] para la instalación del sistema operativo en la tarjeta microSD, el cuál proporciona una lista de distintas opciones para escoger. Estos sistemas operativos están orientados a distintas funcionalidades de empleo de la Raspberry Pi, entre los cuales destaca “Raspberry Pi Os”, anteriormente denominado “Raspbian”. En las Figuras 7, 8, 9, 10, 11, 12 y 13 se expone de manera gráfica el listado completo de sistemas operativos que proporciona el programa descrito, así como la clasificación que se les atribuye. Estas imágenes se han obtenido directamente haciendo uso del programa.

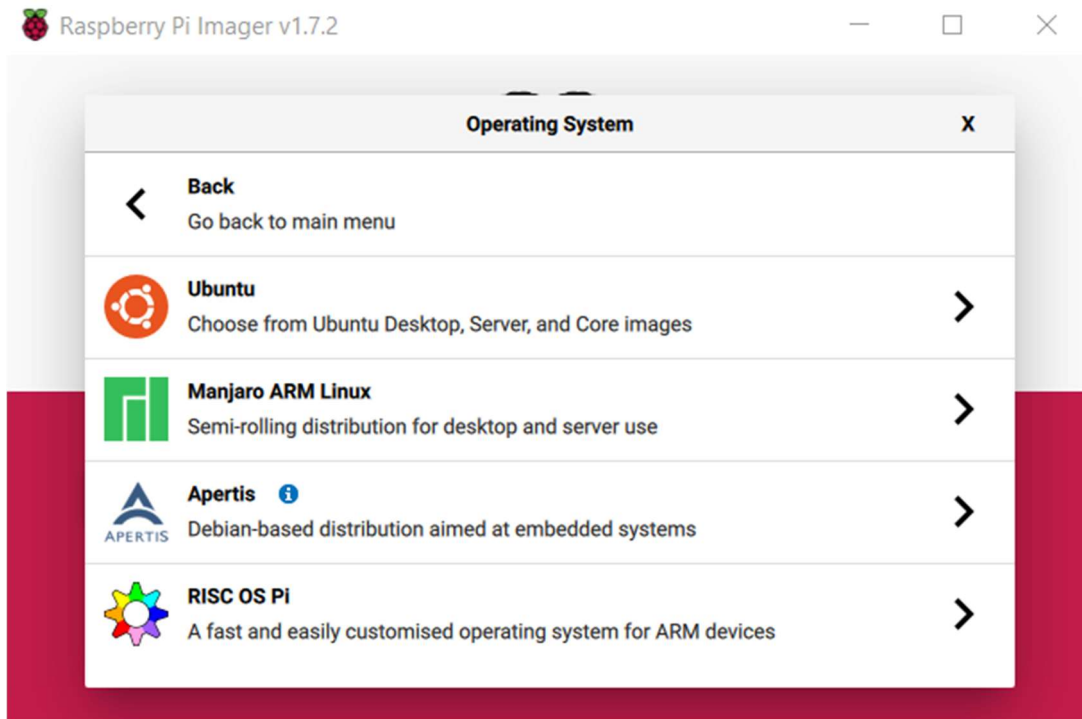


Figura 7. SO proporcionados por "Raspberry Pi Imager" clasificados como de propósito general

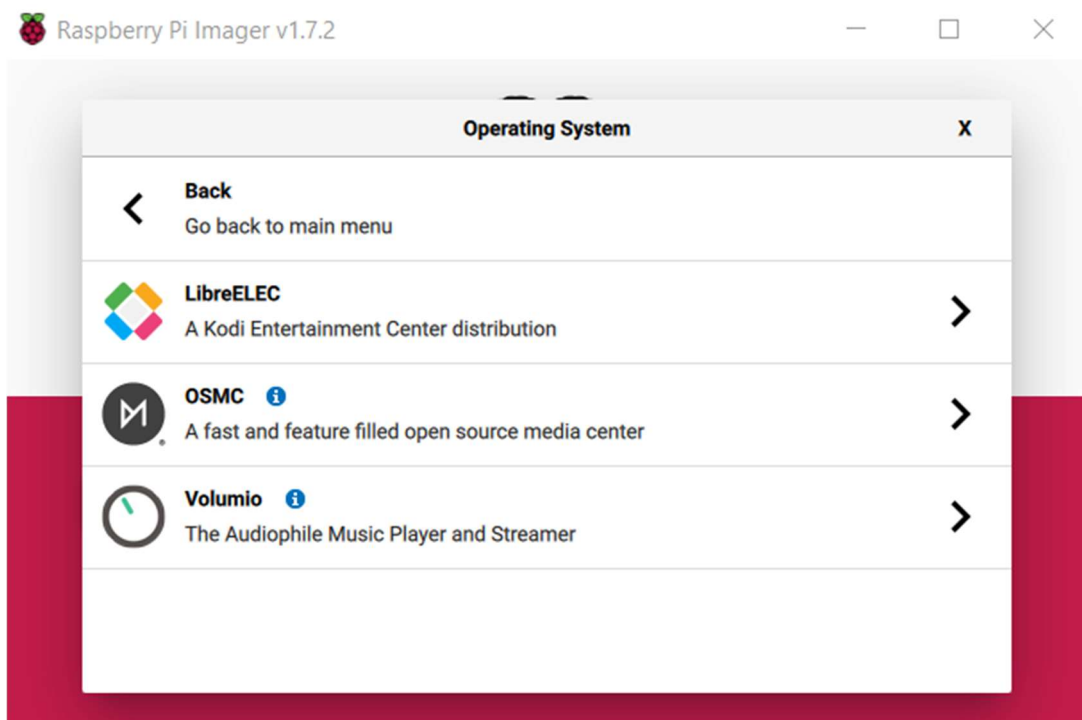


Figura 8. SO proporcionados por "Raspberry Pi Imager" orientados a la reproducción multimedia

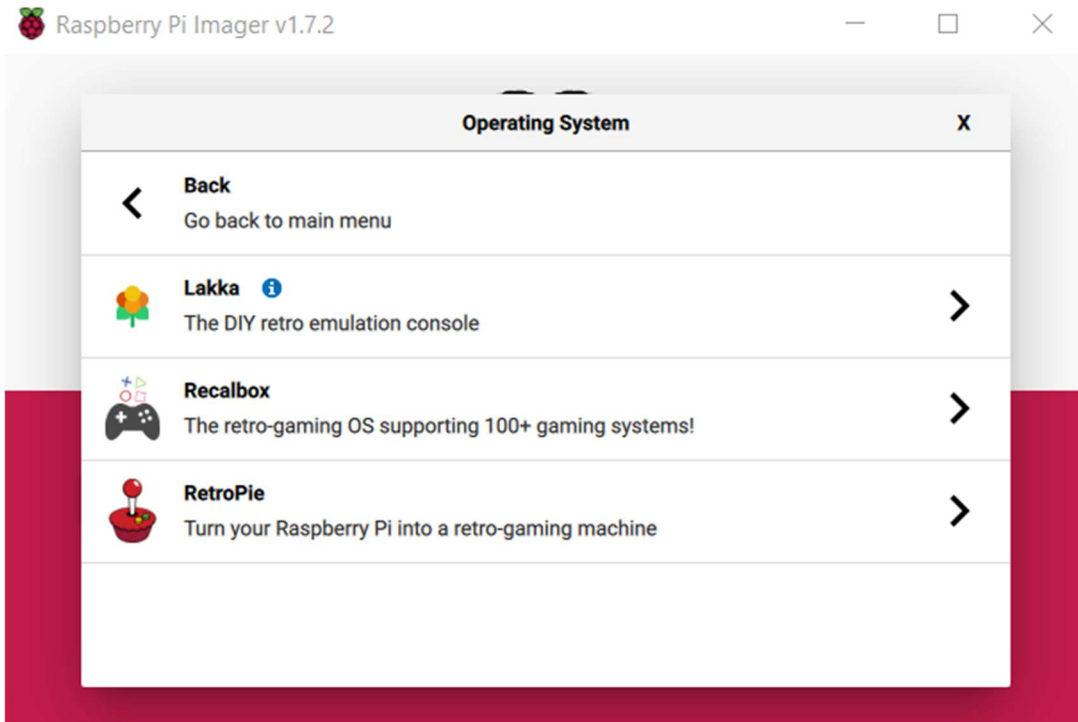


Figura 9. SO proporcionados por “Raspberry Pi Imager” orientados al uso de emuladores de consolas.

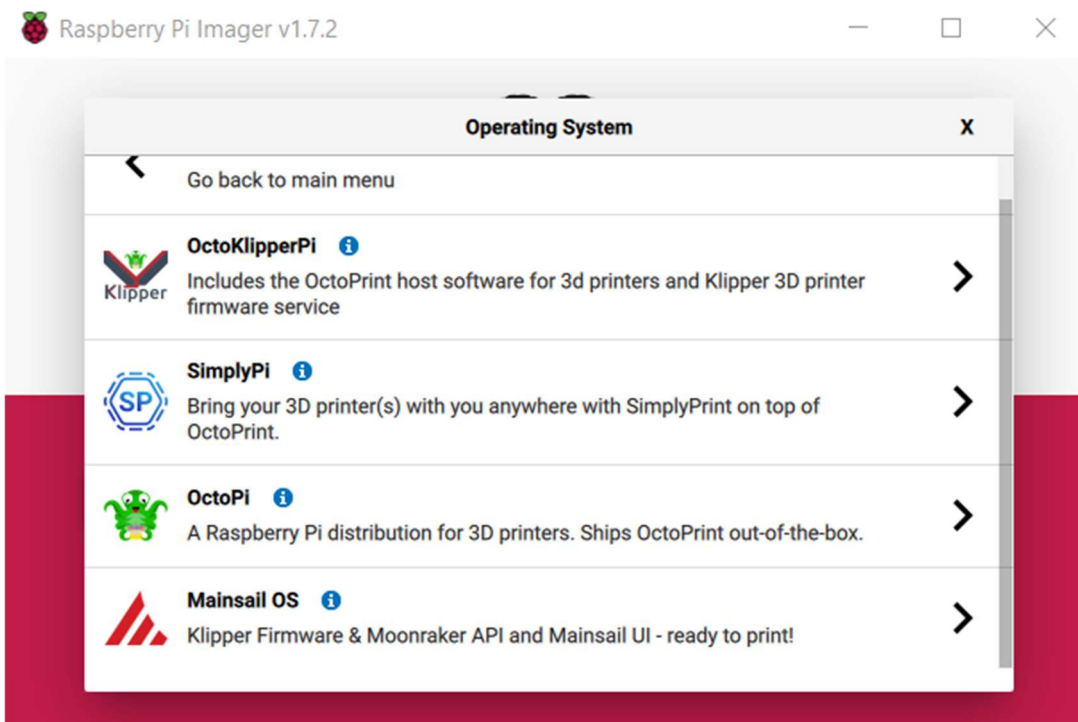


Figura 10. SO proporcionados por “Raspberry Pi Imager” orientados a la impresión 3D

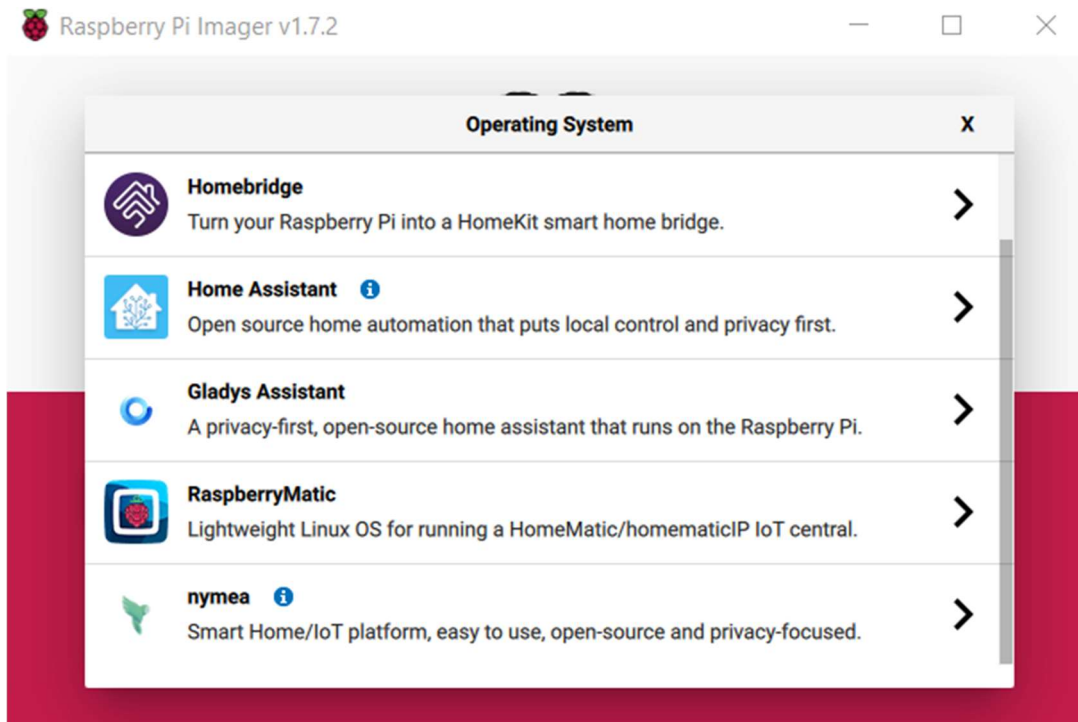


Figura 11. SO proporcionados por “Raspberry Pi Imager” orientados a asistentes y sistemas de automatización domésticos

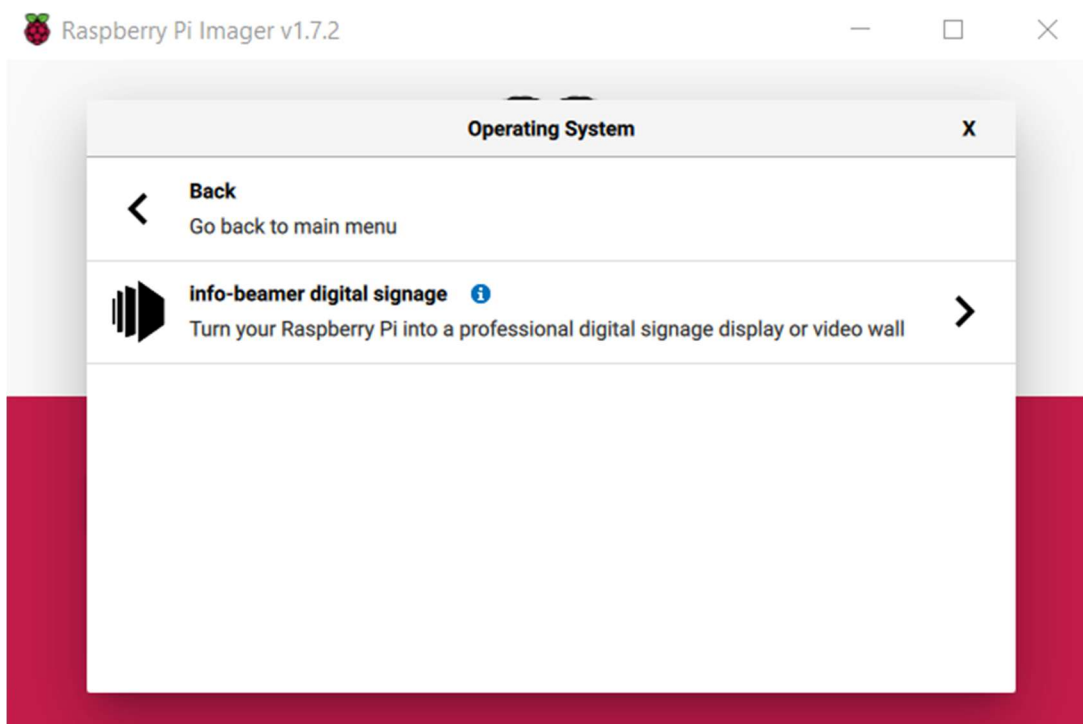


Figura 12. SO proporcionado por “Raspberry Pi Imager” orientado al empleo de un software de señalización digital.

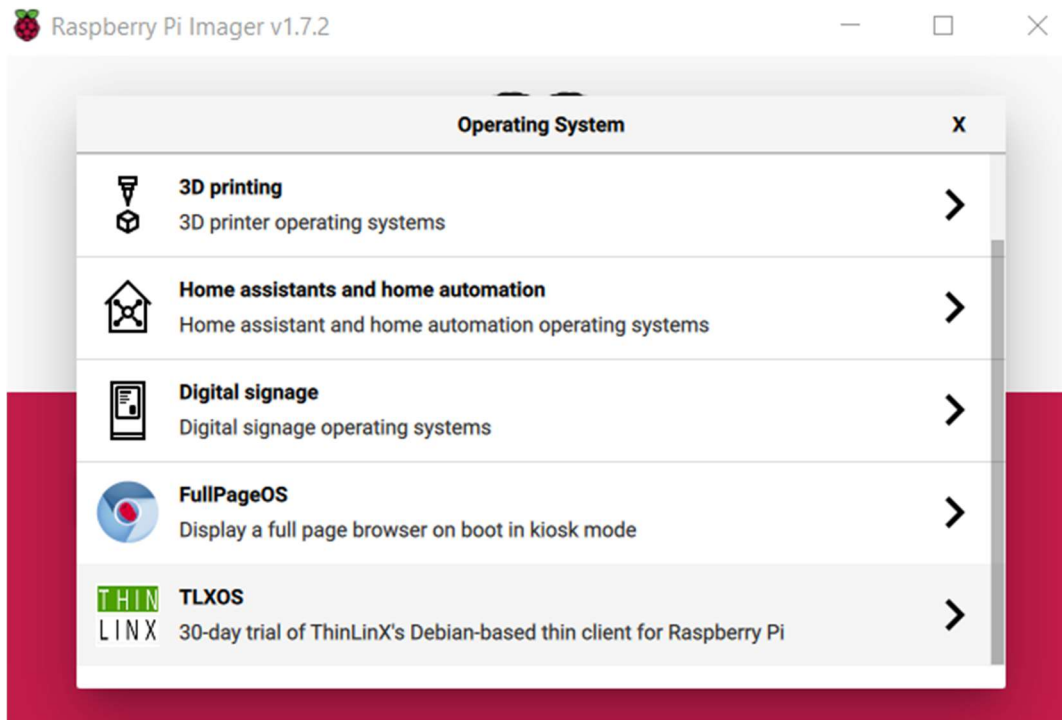


Figura 13. SO proporcionados por “Raspberry Pi Imager” orientados a determinados usos específicos

Además de los expuestos en la documentación gráfica aportada, se pueden instalar otros sistemas operativos empleando medios externos e independientes al uso de “Raspberry Pi Imager”. Algunos de los más destacados son Windows 10 para la versión de Raspberry Pi 4 de 8GB o Windows 10 IoT Cores para el resto de los modelos, LibreELEC, OSMC, Pidora o Fedora, entre otros [19].

“**Raspberry Pi Os**”, el sistema operativo oficial para Raspberry Pi y recomendado por la página oficial, es el que se ha empleado en la realización del proyecto. Sus creadores son Mike Thompson y Peter Green, su lanzamiento fue en 2012, y se trata de una distribución de Linux que se basa en Debian [20].

Este sistema operativo ofrece todas las funcionalidades de la consola de comandos de Linux, así como una interfaz sencilla para el usuario, presentando el menú “raspi-config”, con el que se permite realizar la configuración de la Raspberry Pi sin la necesidad de emplear la consola de comandos. Entre las posibles configuraciones se encuentran la habilitación de las distintas comunicaciones, la realización de la partición de la memoria SD para ampliar la capacidad destinada al almacenamiento limitando la destinada al

sistema operativo y configuraciones varias relativas al teclado y pantallas conectadas, a la propia visualización del interfaz, etc.

Además, este sistema operativo contiene de manera predefinida la posibilidad de establecer conexiones mediante SSH o VNC. El SSH, o *Secure Shell* en inglés, es un protocolo diseñado para permitir la conexión a un servidor remoto de manera segura. Por otro lado, el VNC, o *Virtual Network Computing*, es un programa que permite establecer comunicación entre dos dispositivos, de manera que el principal ejecute el servidor y el cliente pueda observar su contenido de manera remota. Estas dos opciones resultan de utilidad a la hora de utilizar la Raspberry Pi sin necesidad de conectar una pantalla, estableciendo una conexión remota con otro dispositivo [21] [22].

Por otro lado, para la realización del programa que ejecutará la Raspberry Pi que cumpla con las funcionalidades del sistema propuesto, también se dispone de una amplia gama de lenguajes de programación entre los que escoger. Algunos de los más relevantes y conocidos son JavaScript, Java, C++ y **Python** [23], siendo este último el escogido para la realización de la implementación debido a la sencillez de su lenguaje y a la comunidad activa que dispone, que proporciona gran cantidad de documentación y librerías que resultarán de interés en el código realizado.

2.3 COMPONENTES ADICIONALES

Para lograr cumplir con las funcionalidades que requiere el proyecto propuesto, es necesario añadir determinados componentes que cumplan con la toma de datos necesarios, así escoger el método de alimentación considerando sus requerimientos. En este apartado se comentarán los componentes mínimos adicionales que se precisan para la creación del diseño base, omitiendo aquellos que pudieran resultar de interés para lograr otras aplicaciones adicionales en nuestro sistema.

2.3.1 CÁMARA. MÓDULO DE CÁMARA RASPBERRY PI V2

Debido a que el propósito del sistema propuesto es lograr la toma de documentos gráficos correspondientes a vídeos, tras la detección de un impacto del vehículo, la adición de una cámara en la implementación del modelo resulta fundamental a la hora de lograr las funcionalidades básicas propuestas.

Para ello y en primer lugar, la cámara debe ser compatible con el modelo de Raspberry Pi escogido, y debe proporcionar una calidad mínima en las imágenes que se obtengan. Para determinar esta calidad, es necesario conocer las especificaciones que mantengan relación con la misma y que sean de interés para el sistema propuesto.

En primer lugar, la resolución de imagen de la cámara determinará la nitidez con la que se percibe visualmente la imagen, afectando de manera directa a la cantidad de píxeles que ésta contenga. De esta forma, a mayor cantidad de píxeles mayor nitidez y, por tanto, mayor calidad obtenida [24]. Aunque en el mercado se puedan encontrar cámaras de más de 12 megapíxeles, en el proyecto propuesto no es necesario abarcar esas cifras, pues se pretende obtener un equilibrio entre la calidad obtenida y el gasto económico que esto suponga. Es por ello que, para el sistema propuesto, se establecen como valores óptimos entre 6 y 8 megapíxeles, pues estos otorgarán una calidad de imagen suficiente.

Otras especificaciones que resultan de gran relevancia son la distancia focal y el ángulo de visión que alcanza la cámara, estando estas directamente relacionadas. La distancia focal representa la distancia que hay entre el sensor y la lente. A mayor distancia focal, menor será el ángulo de visión y más estrecha se apreciará la imagen [25]. Estas medidas aparecen representadas gráficamente en la Figura 14, obtenida de la referencia [26].

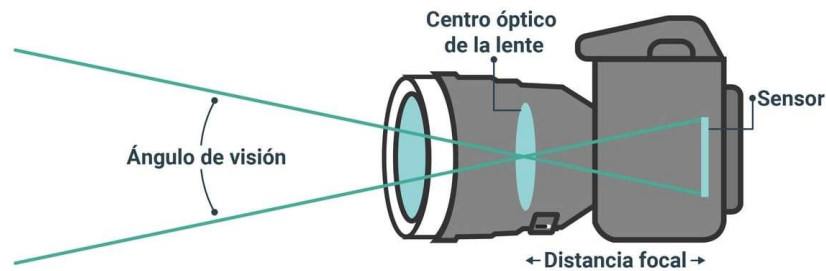


Figura 14. Representación de distancia focal y ángulo de visión, obtenida de [26]

Por otro lado, la resolución de vídeo y el número de fotogramas que se toman por segundo (fps) también resultan relevantes para la toma de imágenes en forma de vídeo. La resolución de vídeo indica la relación entre la cantidad de píxeles horizontales y verticales que se toman durante el vídeo, siendo las resoluciones estándar las que se listan a continuación [26] [27]:

- **QVGA:** 320 x 240
- **VGA:** 640 x 480
- **SVGA:** 800 x 600
- **XGA:** 1024 x 768
- **HD 720:** 1280 x 720
- **HD 1080:** 1920 x 1080
- **2K:** 2048 x 1080 – 2048 x 1536 – 2560 x 1440 – 2560 x 1600
- **4K:** 4096 x 3112 – 3656 x 2664 – 3840 x 2160

Finalmente, otras características relevantes, aunque no críticas, son el peso y dimensiones que presente la cámara, pues al tratarse de un sistema portátil cuyo uso está destinado a vehículos es preferible que estos valores sean lo más ajustados posibles.

Es importante escoger una cámara que, además de cumplir con los requisitos recién expuestos, sea compatible con el modelo de Raspberry Pi empleado. Para ello, se debe escoger una cámara que permita establecer una conexión y envío de información con el dispositivo comentado, empleando bien sea el puerto CSI o bien un puerto USB de la Raspberry.

En vista de los requisitos mínimos que debe cumplir la cámara que se emplee para la implementación del modelo propuesto, se ha decidido escoger el módulo de Cámara de Raspberry Pi V2 [28]. Sus especificaciones técnicas aparecen representadas en la Tabla II.

Tabla II. Especificaciones del Módulo de Cámara de Raspberry Pi V2.	
Resolución de cámara	8 MegaPíxeles (3280 x 2464)
Resolución de vídeo	1080p a 30 fps 720p a 60 fps 640x480p a 90 fps
Lente	Foco Fijo
Peso	3 gramos
Dimensiones	25mm x 23mm x 9mm
Conector	Puerto CSI

2.3.1 UNIDAD DE MEDIDA INERCIAL. MPU 6050

Además del empleo de una cámara, es necesario añadir otros sensores que permitan realizar la toma de datos que otorguen la información necesaria para el propósito que se quiere cumplir. En concreto, para la realización de un modelo que cumpla con las funcionalidades básicas requeridas, es necesario la adición de un sensor que sea capaz de detectar impactos.

Para ello, se empleará una unidad de medida inercial, dispositivo electrónico compuesto por acelerómetros, giróscopos y en ocasiones magnetómetros capaces de medir valores de aceleración, velocidad y orientación del sistema en el que se integre. Las unidades de medida inercial pueden realizar mediciones en uno, dos, tres o más ejes, según la cantidad de componentes que dispongan [29].

Estos dispositivos se emplean mayoritariamente en sistemas de vuelo como pueden ser aviones o drones, para determinar los movimientos de cabeceo, alabeo y guiñada que estos sufran, así como para conocer la posición y orientación de los mismos. Aún cuando la detección de estos movimientos no resulta de interés para la realización del sistema propuesto, a modo ilustrativo se muestran en la Figura 15 al tratarse de la principal aplicación de estos dispositivos.

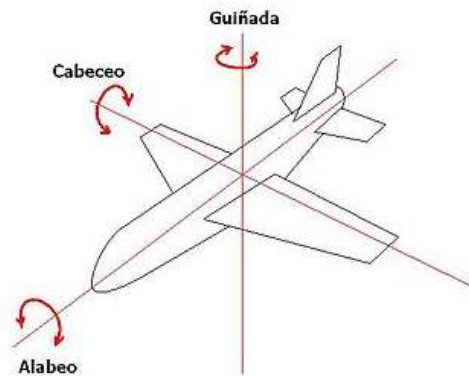


Figura 15. Movimientos de cabeceo, alabeo y guiñada, obtenida de [30]

En relación a los componentes que una unidad de medida inercial contiene, un acelerómetro es un dispositivo capaz de proporcionar datos correspondientes a su propia aceleración lineal con respecto al espacio. Existen diferentes tipos, según el principio físico de funcionamiento en el que se basen, pudiendo estos ser capacitivos, piezoeléctricos, mecánicos o de efecto Hall.

Por otro lado, un giróscopo es un dispositivo capaz de obtener su orientación respecto al espacio, así como valores de velocidad angular. Para ello, se basa en dos principios: la inercia giroscópica, característica que logra la posición paralela del giróscopo con respecto al ecuador de la Tierra, y la precesión, desplazamiento del eje de giro del giróscopo a causa de la aplicación de una fuerza externa. Para aclarar el tipo de desplazamiento que sufre el eje, se muestra de manera ilustrativa este movimiento en la Figura 16.

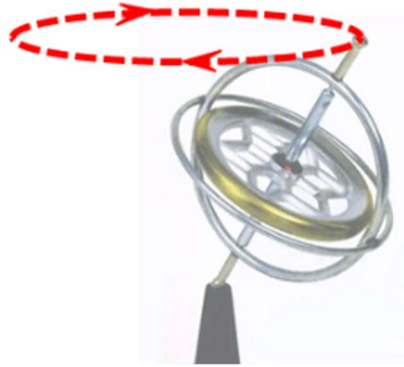


Figura 16. Movimiento de precesión realizado por un giroscopio, obtenida de [31]

Un magnetómetro es un dispositivo que permite realizar la medición de fuerzas y campos magnéticos. Un claro ejemplo de estos dispositivos es la brújula, que es un instrumento destinado a medir el campo magnético de La Tierra, indicando de esta manera las direcciones de Norte, Sur, Este y Oeste. Dado que este dispositivo no se empleará en la implementación no se entrará en detalle del mismo.

La unidad inercial de medida escogida para la realización de la implementación del sistema propuesto es la **MPU6050** [32]. Dispone de un magnetómetro, tres acelerómetros y tres giróscopos, con los que se permite obtener parámetros en tres ejes distintos: x, y, z. Además, integra un procesador digital de movimientos (DPM), con el que permite realizar el procesado de movimiento en 3D, reconocimiento de gestos, detección de golpes, etc. En la Tabla III se muestran las principales especificaciones que han resultado relevantes para la elección de la unidad de medida inercial.

Tabla III. Especificaciones del MPU6050.	
Rango Salida Acelerómetro:	$\pm 2g, \pm 4g, \pm 8g, \pm 16g$
Rango Salida Giróscopo:	$\pm 250 \text{ }^\circ/\text{s}, \pm 1000 \text{ }^\circ/\text{s}, \pm 2000 \text{ }^\circ/\text{s}$
Voltaje Alimentación:	2,375 V – 3,46 V
Comunicación:	I2C
Estructura:	MEMS (<i>Microelectromechanical System</i>)

Esta unidad de medida inercial se encuentra incluida en el módulo **GY-521** [33], que además de disponer de ella contiene un regulador de tensión con un rango de voltaje de entrada de 3,3V – 5,0V. Los pines que dispone este módulo se muestran listados a continuación, donde se muestran resaltados los que han sido empleados para la implementación:

- **VCC**: pin para la alimentación del módulo.
- **GND**: pin de tierra del módulo.
- **SCL**: pin para la comunicación I2C (conexión como *slave*).
- **SDA**: pin para la comunicación I2C (conexión como *slave*).
- **XDA**: pin auxiliar para la conexión de otros sensores (conexión como *master*).
- **XCL**: pin auxiliar para la conexión de otros sensores (conexión como *master*).
- **AD0**: pin para determinar la dirección de registro del MPU6050, siendo esta 0x68 si se conecta a tierra o 0x69 si se conecta a VCC.
- **INT**: pin para interrupción de salida digital.

Finalmente, el conexionado interno del módulo se muestra en la siguiente figura, en la que se puede apreciar el regulador de tensión que dispone, así como las distintas conexiones de los pines externos con los correspondientes a la unidad de medida inercial.

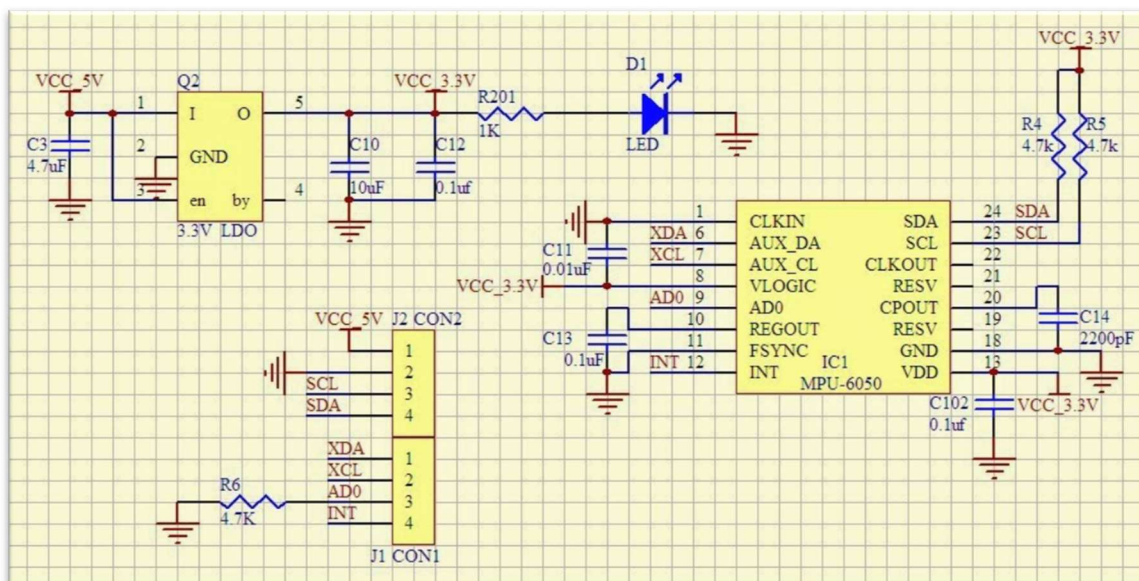


Figura 17. Esquemático del GY-521, obtenida de [33]

2.3.2 MÉTODO DE ALIMENTACIÓN

La alimentación que requiere el sistema de visión artificial se basará en considerar el voltaje y amperaje adecuado para el correcto funcionamiento de la Raspberry Pi. Los distintos componentes empleados recibirán la alimentación desde la propia placa a través de los pines correspondientes, bien precisen 3,3V o 5,0V.

Los valores requeridos para el correcto funcionamiento de la Raspberry Pi varían según modelo y rendimiento, siendo los recomendados aquellos que se aprecian en la Tabla IV [8].

Tabla IV. Valores recomendados de alimentación para distintos modelos de Raspberry Pi		
Modelo	Amperaje (A)	Voltaje (V)
Raspberry Pi 1 Model A+	0,7	5,1
Raspberry Pi 1 Model B+	1,8	
Raspberry Pi 2 Model B		
Raspberry Pi 3 Model A+	2,5	
Raspberry Pi 3 Model B		
Raspberry Pi 3 Model B+		
Raspberry Pi 4 Model B	3,0	
Raspberry Pi Zero	1,2	
Raspberry Pi Zero W		
Raspberry Pi Zero 2W	2,0	

No obstante, el fabricante recomienda emplear unos valores comunes para todos estos modelos en caso de desear utilizar la fuente de alimentación oficial, siendo estos de 5,1V y 2,5A.

Es de vital importancia cumplir con los valores indicados por el fabricante, dado que no lograr una alimentación correcta puede suponer diversos problemas a la hora de

trabajar con la Raspberry Pi, entre los cuales se encuentran la posible corrupción de la tarjeta de memoria empleada (SD o microSD según modelo), congelamiento de programas o mal funcionamiento general.

Por otro lado, se pueden aplicar diferentes medios para alimentar la Raspberry Pi. En primer lugar, existe el medio convencional en el que se emplearía el puerto microUSB que viene incorporado en la placa con la finalidad de recibir la alimentación y transmitirla de manera segura a todos los componentes. Otras posibilidades son la de alimentar la placa de manera directa a través de los pines de propósito general (GPIO) o mediante el uso de un puerto USB. Para este proyecto se optará por emplear el puerto microUSB, dado que es la forma oficial de alimentación y dado que utilizar otros métodos resultarían en obviar el sistema de protección interno que dispone la Raspberry en caso de sobreintensidades o de sobretensiones [35].

Debido a que la Raspberry Pi va a ser empleada para la implementación de un sistema de visión artificial para vehículos, es necesario considerar diferentes opciones para la alimentación de la misma, siendo las estudiadas emplear una batería externa o bien optar por la alimentación directamente desde el vehículo, habiendo escogido esta última para lograr una mayor autonomía, así como por la comodidad del propio usuario.

Dentro de las posibles opciones al emplear alimentación tomada directamente desde el vehículo se encuentran emplear el puerto de salida USB o emplear el conector mechero, los cuales proporcionan, generalmente, unos valores de salida de 5V a 500mA frente a 12 – 14V a 20A respectivamente [36]. Considerando los valores recomendados para el correcto funcionamiento de la Raspberry Pi, se ha optado por emplear un adaptador para la toma de mechero que proporcione a su salida los valores requeridos para el modelo escogido, en nuestro caso **2,5A** y **5 V** al tratarse de la Raspberry Pi 3 Model B, y que acepte a su entrada el rango que proporciona dicha toma.

2.3.3 MÉTODO DE ALMACENAMIENTO

Para la realización del sistema propuesto es preciso establecer un método de almacenamiento en el que, en primer lugar, se pueda instalar el sistema operativo y almacenar todo lo correspondiente al mismo, así como las aplicaciones y programas que se emplearán para la realización y ejecución del código; y en segundo lugar se almacenen todos los archivos creados u obtenidos durante la ejecución del código programado, referentes a la documentación gráfica y parámetros obtenidos mediante los sensores del sistema. El método de almacenamiento puede ser común para estos dos aspectos, o bien se pueden determinar distintos medios. Es por ello que se analizarán las distintas posibilidades existentes para cada uno de los aspectos.

En primer lugar, la memoria de almacenamiento destinada a todos los archivos requeridos para la correcta inicialización del sistema operativo, así como toda aquella información relevante a las aplicaciones y programas instalados en el mismo, puede emplear como medio o bien una tarjeta microSD o bien un USB o disco duro externo. Debido a que el sistema está orientado a la toma de imágenes en vehículos en movimiento se empleará para esto una tarjeta microSD, pues resulta más compacto que un USB o disco duro externo, y más sencillo y simple que el uso de una nube. No obstante, esto último y otras opción se contemplarán en el apartado denominado “4.2 Proyectos de Ampliación”.

De esta manera, y considerando como método escogido para la implementación el uso de una tarjeta microSD para el almacenamiento de los dos aspectos estudiados, y además considerando que esta tarjeta deberá establecer comunicación con la Raspberry Pi, resulta relevante conocer las características que debe cumplir para ser compatible con el dispositivo.

En la documentación proporcionada en la página oficial de Raspberry Pi, establecen como recomendación el empleo de una tarjeta microSD de 8Gb o superior [37]. No obstante, según distintas publicaciones consultadas el máximo permitido por el dispositivo en cuestión es de 64 Gb, y en vista de que se precisará de memoria de almacenamiento para el guardado de los vídeos que tome la cámara, este es el valor que se ha decidido emplear.

Además de la capacidad de almacenamiento de la que dispone la tarjeta microSD, para este proyecto en concreto resulta relevante la velocidad tanto de escritura como de lectura que permita, pues esto afectará de manera directa a la velocidad de lectura y generación de ficheros que se produzca durante el funcionamiento del sistema propuesto.

De esta manera, y tras la comparativa realizada de las tarjetas disponibles en el proveedor determinado, y considerando las distintas características y sus respectivos precios, se ha decidido emplear una tarjeta microSDXC **Sandisk de 64GB, 100 MB/s de lectura y 60 MB/s de escritura** [38].

La diferencia entre una tarjeta microSD y una tarjeta microSDXC es la capacidad máxima capaz de alcanzar por cada una, siendo para la primera 32 GB y para la segunda 128 GB. Es necesario considerar que al emplear una tarjeta microSDXC, como es el caso de la implementación realizada, se debe realizar el formateo de la tarjeta a formato fat32, pues de otra manera no resultará compatible con la Raspberry Pi empleada. Para realizar el formateo de la tarjeta se puede emplear los métodos proporcionados por Windows, siendo estos el empleo del explorador de Windows, el administrador de discos, diskpart a través de la consola de comandos o el Powershell de Windows. De manera adicional, se pueden emplear programas de terceros diseñados para tal fin. En el caso de este proyecto, se recurrió al empleo del programa “guiformat.exe” para realizar el formateo de la tarjeta a formato fat32, pues los métodos proporcionados por Windows no resultaron de utilidad.

Finalmente, tal y como se comentó en el apartado denominado “2.2.4 *Software y Lenguajes de Programación*”, estando este comprendido en el apartado denominado “2.2 *Elemento Principal. Raspberry Pi*”, tras realizar la instalación del sistema operativo así como todas las configuraciones iniciales de la Raspberry Pi resulta necesario realizar la partición de la memoria de almacenamiento. Esto se realiza con el fin de que los espacios destinados para el guardado de toda aquella información relativa al sistema operativo y para todos los demás ficheros y programas se mantengan diferenciados, de manera que se reduzca y se ajuste el almacenamiento empleado para el sistema operativo, logrando de esta manera la optimización de la capacidad de almacenamiento de la tarjeta.

3. IMPLEMENTACIÓN Y ANÁLISIS DE RESULTADOS

A modo justificativo, se ha realizado una implementación simplificada de un modelo que cubre las funcionalidades básicas de un sistema de visión artificial para vehículos orientado a la toma de imágenes en situaciones de accidente o impacto.

La implementación se ha realizado siguiendo las siguientes fases:

- Fase 1. Elección de componentes (Realizada en el apartado denominado “2. *Conceptos Previos al Diseño*”, estudiado previo al actual)
- Fase 2. Creación del código.
- Fase 3. Realización de pruebas y reajustes.

3.1 CONSIDERACIONES PREVIAS

Debido a la escasez de semiconductores que afecta al mercado actualmente, a fecha de realización del presente Trabajo de Fin de Grado, ha resultado imposible lograr la compra de ciertos componentes, presentando de esta forma una complicación añadida a la realización de la implementación.

Esta situación ha provocado la aparición de ciertos problemas, que junto a la solución aplicada se comentarán previo a la explicación del modelo planteado.

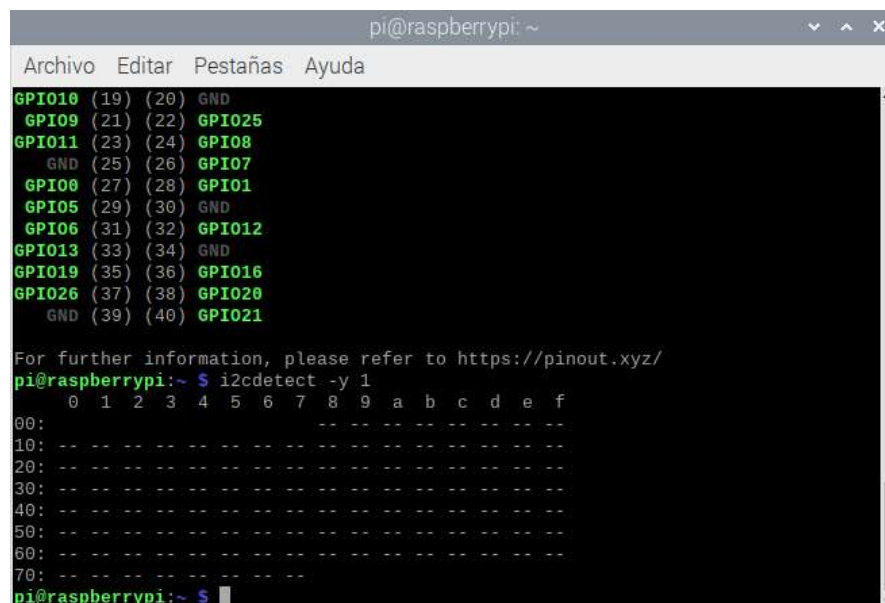
3.1.1 PROBLEMAS SURGIDOS Y SOLUCIÓN APLICADA

Tal y como se comentó previamente, resultó imposible realizar la compra de ciertos componentes. Concretamente, se presentó una escasez absoluta en el mercado de Raspberry Pi, componente principal del sistema propuesto.

Es por ello que la Raspberry Pi empleada fue otorgada por la Universidad de La Laguna, siendo la única en disposición. Al tratarse de un componente antiguo, cuyas condiciones de uso y mantenimiento se desconocen, presentó distintos problemas a la hora de realizar la implementación.

En primer lugar, el módulo WiFi presentaba fallos a la hora de establecer la conexión. Este problema no resultó de gran magnitud, pues se corrigió de manera sencilla mediante la conexión directa por cable Ethernet.

Por otro lado, surgió un problema de mayor magnitud a la hora de intentar establecer la conexión mediante comunicación I2C con la unidad de medida inercial. Al momento de conectar el sensor con los pines de la Raspberry Pi, como se indica en la documentación asociada al sensor (ver *Datasheet* [32] ó Anexo VI), y realizar la búsqueda de dispositivos conectados a los pines para la comunicación I2C mediante la consola de comandos, no se detectan dispositivos conectados. En la Figura 18 se muestra la no detección del sensor mediante la comunicación I2C.



```
pi@raspberrypi: ~  
Archivo Editar Pestañas Ayuda  
GPIO10 (19) (20) GND  
GPIO9 (21) (22) GPIO25  
GPIO11 (23) (24) GPIO8  
GND (25) (26) GPIO7  
GPIO0 (27) (28) GPIO1  
GPIO5 (29) (30) GND  
GPIO6 (31) (32) GPIO12  
GPIO13 (33) (34) GND  
GPIO19 (35) (36) GPIO16  
GPIO26 (37) (38) GPIO20  
GND (39) (40) GPIO21  
  
For further information, please refer to https://pinout.xyz/  
pi@raspberrypi:~$ i2cdetect -y 1  
 0 1 2 3 4 5 6 7 8 9 a b c d e f  
00: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  
70: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  
pi@raspberrypi:~$
```

Figura 18. Comprobación de dispositivos detectados por comunicación i2c, con el resultado de la no detección del sensor.

Para tratar de solventar este problema, se realizó la instalación, mediante la línea de comandos `sudo apt-get update` seguida de la línea de comandos `sudo apt-get install i2c-tools python-smbus`, de los paquetes necesarios para establecer la correcta comunicación i2c, siendo estos: `i2c-tools` y `python-smbus`. Este último presentó problemas de instalación, indicando en el terminal que se recurriera a un método de instalación externo, pues se encontraban restos del paquete pero era imposible realizar su instalación.

Para ello, se buscó el archivo en concreto en la web www.packages.debian.org [39] y se procedió a su instalación siguiendo las indicaciones otorgadas. Esto requería realizar la adición de “*deb http://ftp.es.debian.org/debian buster main*” al archivo denominado *sources.list*, ubicado en la dirección */home/pi/etc/apt*. De esta manera, al tratar de instalar el paquete *Python-smbus*, se emplearía la dirección añadida al archivo modificado para descargar e instalar dicho paquete.

No obstante, al tratar de realizar la instalación saltó un error relacionado a la falta de determinadas claves públicas. Para solventar este problema se empleó la línea de comando *sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys XX*, donde *XX* representa el número de la clave requerida, obtenida de la respuesta otorgada por el terminal [40].

Una vez actualizadas las claves públicas requeridas por el terminal, se logró finalmente instalar el paquete *Python-smbus*.

Seguidamente se procedió a verificar la detección del sensor por parte de la Raspberry Pi mediante la comunicación I2C, obteniendo nuevamente el resultado inicial: la no detección del sensor. En ambas situaciones se trató de detectar el sensor en los dos registros que permite según su *datasheet*, siendo estos los *0x68* y *0x69*, obteniendo para ambos el mismo resultado.

En vista de que el problema no se originaba en los paquetes requeridos para establecer la comunicación I2C, se buscó cuál podría ser la causa, analizando la posibilidad de que se tratara de una incompatibilidad entre las frecuencias empleadas en el bus de la comunicación [41].

Para tratar de solventar este problema, se modificó el archivo “*config.txt*” ubicado en el directorio */boot*. Se aplicaron distintas frecuencias, incluidas las determinadas en el *datasheet* del sensor, y además se estableció una frecuencia de núcleo determinada para evitar fluctuaciones que pudieran estar originando el problema. No obstante, el resultado fue el mismo que al comienzo, la no detección del sensor.

En vista a las pruebas realizadas y para descartar que el fallo fuera del sensor, se realizó el conexionado con un Arduino Mega y se realizó la conexión. Se comprobó que se establecía una correcta comunicación mediante I2C entre el sensor y el Arduino, por lo que se determina que el fallo es correspondiente a la Raspberry Pi.

Tras las pruebas realizadas comentadas previamente, se determina que la causa probable de la no detección del sensor pueda deberse a un fallo en el hardware, bien sea en los pines, en el conexionado de buses de la Raspberry Pi o en su resistencia pull-up interna. Esto se puede afirmar al tratarse de una Raspberry Pi antigua, cuyo trato y usos previos a los realizados en esta implementación se desconocen, y que además ha presentado fallos referentes al hardware en su módulo WiFi, que en ocasiones establece conexión y en ocasiones no.

Debido a que la unidad de medida inercial resulta de gran importancia en el funcionamiento del sistema propuesto, se ha decidido establecer la conexión entre la Raspberry Pi y el sensor empleando el Arduino Mega como intermediario. De esta manera, se realizaría una conexión I2C entre el Arduino y el sensor para realizar la lectura de los valores, y una comunicación serial entre el Arduino y el ordenador portátil, de manera que se puedan guardar estos valores en ficheros creados para tal fin. Finalmente, estos ficheros se emplearán en la Raspberry Pi de manera que ésta pueda obtener los datos mediante la lectura de los ficheros.

En el diseño e implementación original, con componentes que no presenten fallos, se puede obviar el uso del Arduino Mega como intermediario. Pero en esta implementación se empleará para lograr justificar el funcionamiento del sistema, así como la dependencia de la toma de imágenes con los datos obtenidos de la unidad de medida inercial.

Por otro lado, siendo éste el último problema surgido, debido a la escasez de semiconductores comentada no se encontró en el mercado el módulo GPS que se emplearía para determinar la localización del accidente. Puesto que no resulta crítico el uso de este sensor para el funcionamiento básico del sistema, se decidió no emplearlo en la implementación.

El uso de este sensor en la implementación se considera no crítico debido a que los datos no resultan relevantes en la dependencia entre la toma de imágenes y datos externos obtenidos. Además, el tratamiento de dichos datos se realizaría de manera análoga al sensor estudiado, en lo que al código de programación se refiere, por lo que su implementación únicamente aportaría un mejor acabado del modelo, que al tratarse de una implementación simplificada se considera aceptable su omisión.

3.2 MONTAJE DEL MODELO

Previo a la realización del código, así como la realización de pruebas y verificación de su funcionamiento, se procedió a realizar el montaje y conexionado entre los distintos dispositivos y componentes requeridos para la implementación. En este subapartado se detallará de manera ilustrativa todo el material empleado, así como el montaje final obtenido. De manera adicional a lo mostrado en los siguientes subapartados, se ha empleado un portátil Asus y una pantalla. Cabe destacar que la documentación gráfica correspondiente al conexionado de los pines del sensor con el Arduino se aporta de manera detallada en su correspondiente anexo.

Todas las imágenes comprendidas en los siguientes subapartados denominados “3.2.1 Montaje y Conexionado. Raspberry Pi” y “3.2.2 Montaje y Conexionado. Arduino Mega” han sido obtenidas durante la realización de la implementación, considerándose estas procedentes de fuente propia.

3.2.1 MONTAJE Y CONEXIONADO. RASPBERRY PI

Para la realización del montaje del modelo, así como el conexionado con la Raspberry Pi, se han empleado los dispositivos, elementos y componentes que se muestran en las siguientes Figuras 19, 20, 21, 22, 23 y 24.

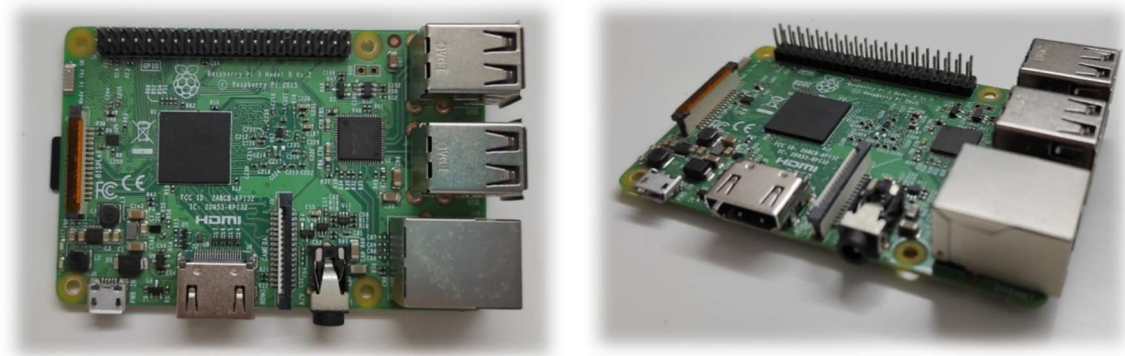


Figura 19. Raspberry Pi 3 Model B

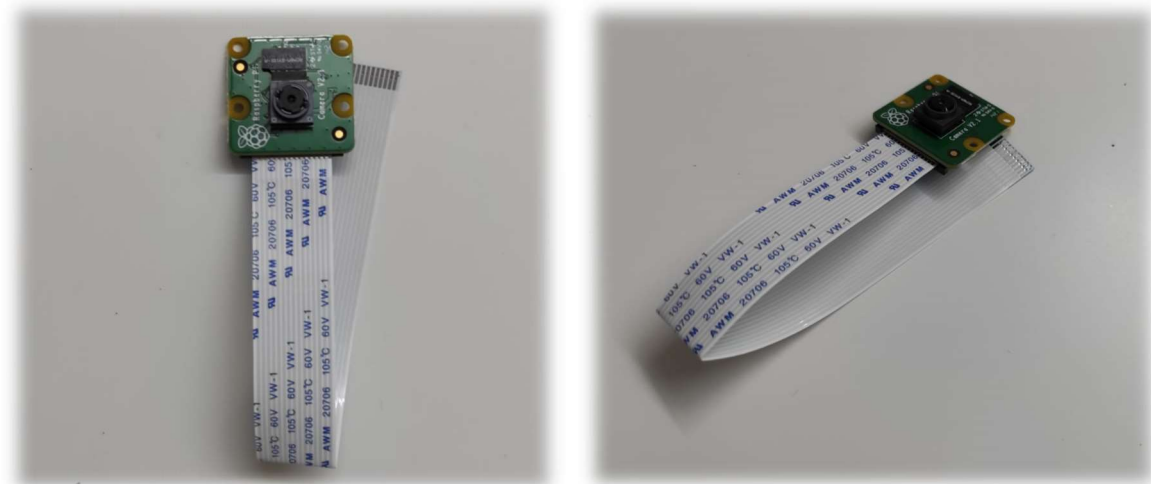


Figura 20. Módulo de Cámara Raspberry Pi V2

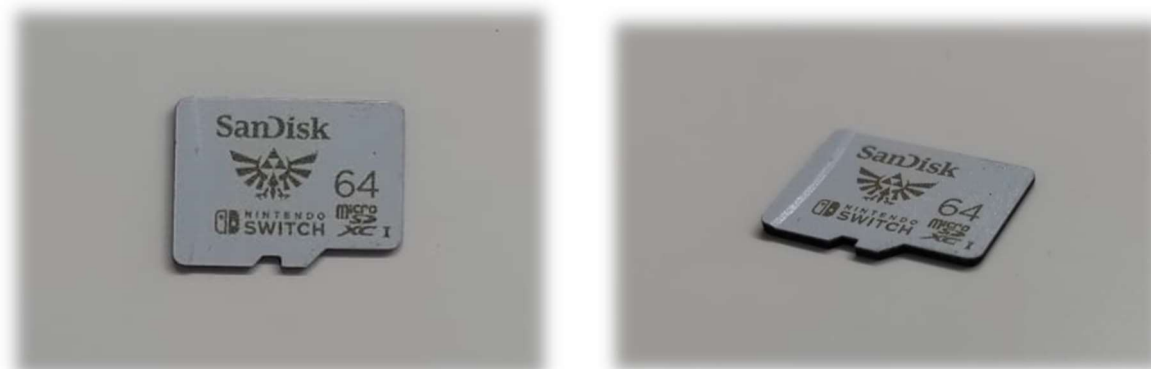


Figura 21. Tarjeta microSDXC SanDisk 64GB



Figura 22. Fuente de alimentación, 5V y 2,4 A



Figura 23. Cable Ethernet y HDMI.



Figura 24. Periféricos conector USB

Con todos los dispositivos y componentes que se exponen en las figuras anteriores, se realizó el montaje final en la Raspberry Pi, el cual se muestra en la Figura 25.

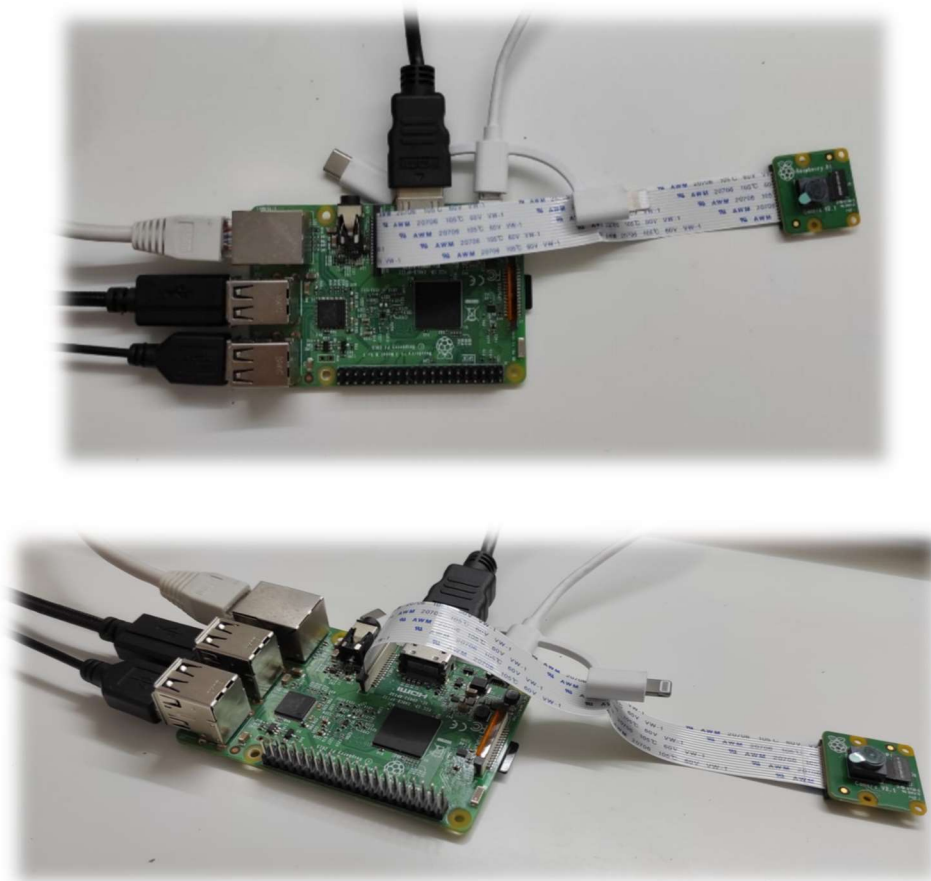


Figura 25. Montaje final con Raspberry Pi

3.2.2 MONTAJE Y CONEXIONADO. ARDUINO

Para la realización del montaje del modelo, así como el conexionado con el Arduino Mega, se han empleado los dispositivos, elementos y componentes que se muestran en las Figuras 26, 27, 28, 29 y 30.

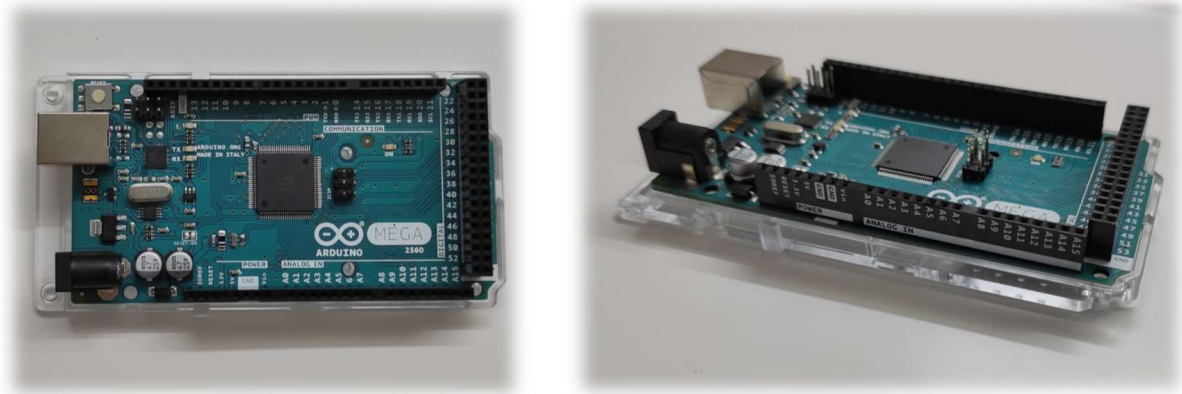


Figura 26. Arduino Mega

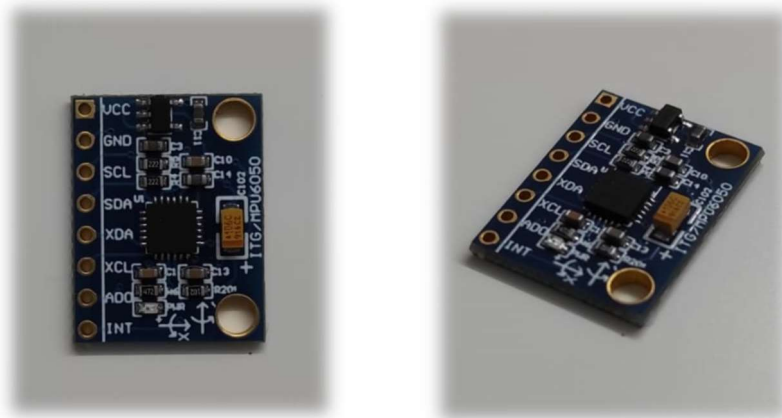


Figura 27. Módulo GY-521, con sensor MPU6050

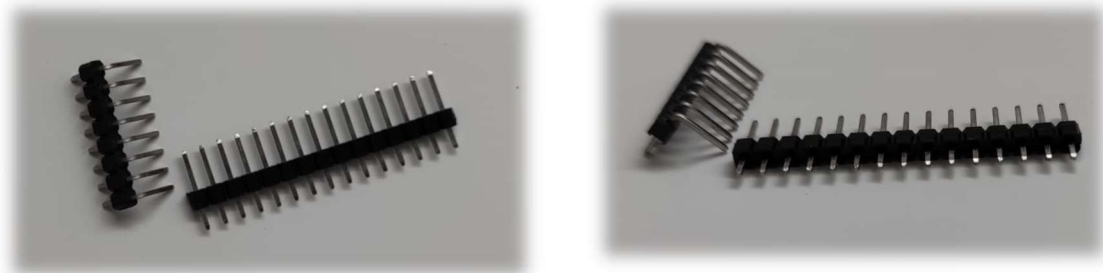


Figura 28. Pines para el conexionado

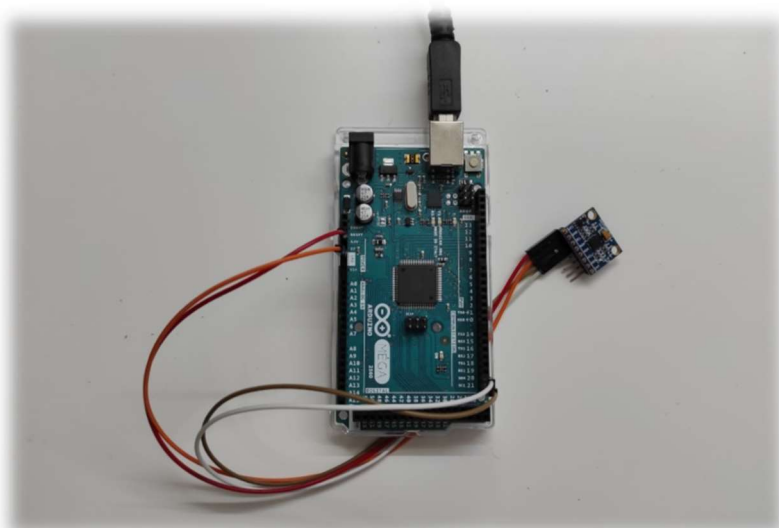


Figura 29. Cables para el conexionado.



Figura 30. Cable alimentación Arduino desde portátil

Con todos los dispositivos y componentes que se exponen en las figuras anteriores, se realizó el montaje final en la Raspberry Pi, el cual se muestra en la Figura 31.



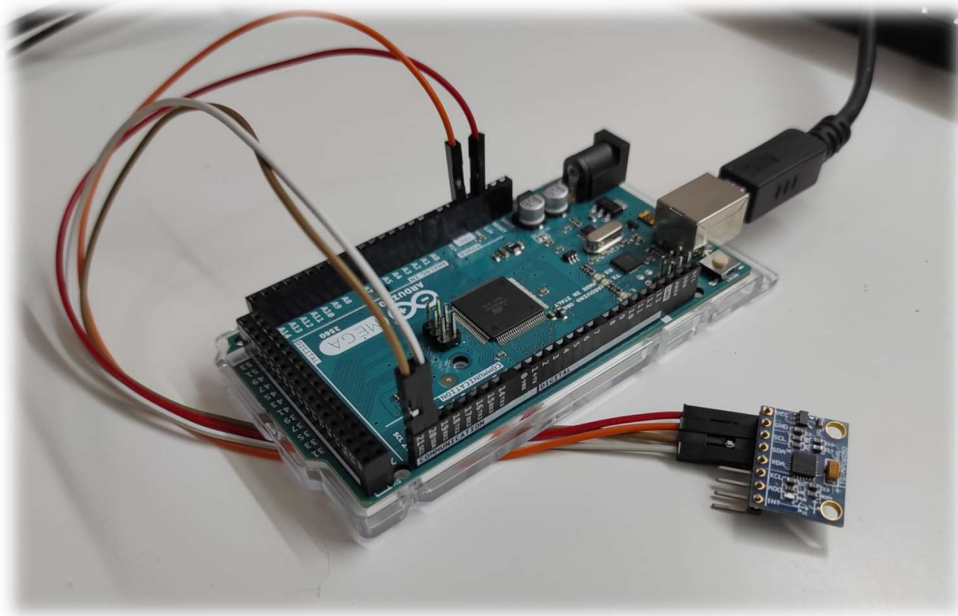


Figura 31. Montaje final con Arduino Mega

Cabe destacar que, para el conexionado entre pines del sensor y el Arduino Mega, este se ha realizado siguiendo el esquemático adjuntado en el Anexo II.

3.3 CÓDIGOS EMPLEADOS

En este apartado se trata de manera detallada y explicativa los códigos empleados para lograr las funciones básicas simplificadas del sistema propuesto: grabación continua de imágenes, detección de impacto, guardado de imágenes correspondientes al rango de tiempo en el que se produce el impacto. Estos códigos han sido realizados por el autor del presente Trabajo de Fin de Grado, salvo el empleado para la calibración del sensor, proporcionado en la documentación de la librería MPU6050.h y de libre uso con el condicionante de referenciar a sus autores, lo cual se realizará en su apartado correspondiente.

Dadas las acciones a ejecutar, el código se ha estructurado de la siguiente manera:

CÓDIGO EN PYTHON IDLE [42]:

- Bucle de grabación y borrado de vídeos condicionado, manteniendo siempre un número determinado de archivos guardados en la memoria de manera que se asegure que no se exceda la capacidad de la misma.
- Conversión y unión de los vídeos que se encuentran guardados en la memoria posterior al impacto.
- Bucle en hilo simultáneo al principal, en el que se realiza la lectura de los ficheros de datos, y se realiza el tratamiento de los mismos para la detección de un impacto.

CÓDIGO EN ARDUINO IDE [43]:

- Bucle de lectura de los valores obtenidos por el sensor, así como su envío por comunicación serial para la escritura de los ficheros de datos.

CÓDIGO EN PYCHARM [44]:

- Bucle para la obtención de los datos por comunicación serial, así como su escritura en los ficheros correspondientes.

En relación al código empleado en la Raspberry Pi, correspondiente al que se engloba dentro de Python IDLE, para lograr la correcta funcionalidad se ha empleado lo que se denomina como programación en hilos, que permite ejecutar de manera simultánea un subproceso y el hilo principal del programa, es decir, permite realizar el bucle de grabación y borrado de imágenes en el hilo principal, mientras se ejecuta un subproceso de manera simultánea en el que se realiza la lectura de datos del fichero y el tratamiento de los mismos para determinar el momento en el que se produce un impacto.

Concluyendo la breve introducción de la estructura del programa, a continuación se entra en detalle en cada uno de los códigos empleados. Estos se encuentran anexados al final del documento, en los Anexos III y IV.

3.3.1 GRABACIÓN EN BUCLE CONDICIONADA

El principal objetivo a lograr en la realización de este código es la ejecución simultánea de dos procesos distintos. En primer lugar, en el proceso principal se ejecutará el bucle de grabación, guardado y borrado de imágenes, de manera repetida hasta que se detecte un impacto. De esta manera, se creará un subproceso en otro hilo en el que se realice la lectura de datos del fichero y su análisis, de manera que si se supera un valor umbral, correspondiente a la simulación de un impacto, se cierre el hilo y se salga del bucle de grabación.

Finalmente, se realizará el tratamiento de los archivos obtenidos para lograr un único vídeo final en formato .mp4.

De esta manera, se realizará la explicación del código separándolo en distintos bloques, de manera que se facilite su comprensión. La estructura a seguir será la siguiente:

- Inicialización de las librerías y utilidad de las mismas.
- Bucle de grabación de imágenes condicionada
- Bucle detector de impacto.
- Tratamiento de imágenes tras el impacto.

- Inicialización de las librerías y utilidad de las mismas.

Las librerías empleadas en el programa se muestran, en su mayoría, en las primeras líneas de código. Esto aparece representado en la Figura 32.

```
import io
import os
import sys
import threading
import picamera
import time
import pandas as pd
from subprocess import call

from moviepy.editor import *
```

Figura 32. Librerías empleadas en el bucle de grabación condicionado

Las utilidades de las librerías empleadas se exponen a continuación:

- **Librería io:** empleada para trabajar con *streams* [45].
- **Librería os:** empleada para acceder a los archivos guardados en la memoria [46].
- **Librería sys:** empleada para cerrar el hilo secundario en el momento de detección de un impacto [47].
- **Librería threading:** empleada para trabajar con subprocesos ejecutados en hilos simultáneos al programa principal. En concreto, empleada en el bloque de lectura del sensor y tratamiento de los datos para lograr la detección de un impacto [48].
- **Librería picamera:** empleada para el control del módulo de cámara y de la toma de imágenes mediante el mismo. En concreto, empleada en el bloque de grabación continua y borrado de imágenes [49].
- **Librería time:** empleada para la adición de determinados retardos en distintos puntos del código [50].
- **Librería subprocess:** empleada para poder ejecutar una línea de código a través de la consola de comandos del Raspbian [51].
- **Librería Pandas:** empleada para la lectura de datos a partir de ficheros .xlsx [52].
- **Librería moviepy.editor:** empleada para realizar la unión de los vídeos tomados una vez detectado el impacto, obteniendo así un único vídeo final [53].

Adicionalmente, aun cuando no se ha empleado en el propio código, ha sido necesario realizar la instalación de la librería **openpyxl** [54] para lograr el correcto funcionamiento de la librería Pandas, pues se requiere de manera adicional. Además,

cabe destacar la necesidad de instalación de la librería **numpy** [55] para poder trabajar con vectores y matrices.

- Bucle de grabación de imágenes condicionadas.

En este bloque se realiza la grabación de manera continua, así como el borrado de archivos guardados en la memoria, de manera que se garantice que no se supere la capacidad de la misma.

Para actuar sobre el módulo de cámara empleado en la implementación, se ha empleado la librería Picamera, tal y como se comentó al comienzo de este apartado.

El código correspondiente a este bloque se muestra en las siguientes líneas de código. Esto aparece representado en la Figura 33.

```
while choque:
    with picamera.PiCamera() as camera:
        camera.resolution = (640, 480)
        camera.start_recording('Video%d.h264' % i)
        camera.wait_recording(5)
        camera.stop_recording
        i += 1
        choque = hilo1.is_alive()
    if i >= 4:
        os.remove('Video%d.h264' % (i - 3))
```

Figura 33. Bucle de grabación condicionada

La toma de imágenes y borrado de las mismas se encuentra dentro de un bucle while, con condicionante la variable choque en alta. La posición en alta de esta variable es debida a la simultánea ejecución del hilo denominado HiloSensor, es decir, indica que el hilo continúa ejecutándose, y por consiguiente aún no se ha detectado impacto.

En las líneas de código mostradas en la imagen anterior se establece la resolución del vídeo tomado, se graba un vídeo de 5 segundos y se incrementa el valor de la variable i, de manera que el siguiente archivo de vídeo que se grabe no sustituya al actual.

Mediante la línea de código posterior al incremento de *i*, se comprueba que el hilo denominado *HiloSensor* continúa ejecutándose, es decir, que aún no se ha producido impacto. Esto se realiza para sobrescribir el valor de la variable *choque*, que es el condicionante principal del bucle de este bloque, de manera que si se detecta un impacto esta variable se actualice y se salga del bucle.

En las siguientes líneas de código se borran los archivos de vídeos anteriores a los tres últimos tomados. De esta manera, durante la ejecución del bucle siempre se guardarán tres vídeos en la memoria, de forma que se garantice que al momento en el que se produzca un impacto este quedará registrado.

Una vez se detecte un impacto dejará de ejecutarse el bucle descrito anteriormente, y se ejecutarán las siguientes líneas de código. Esto aparece representado en la Figura 34.

```
with picamera.PiCamera() as camera:  
    camera.resolution = (640, 480)  
    camera.start_recording('VideoFinal.h264')  
    camera.wait_recording(5)  
    camera.stop_recording()
```

Figura 34. Grabación del clip posterior a la detección del impacto

En estas líneas se ejecuta la grabación de un vídeo final obtenido tras el impacto, de una duración de 5 segundos. Esto se realiza de manera análoga a la comentada previamente.

Con estas líneas de código se da por finalizado el bloque de grabación continua y borrado de imágenes.

- Bucle detector de impacto

Para la realización del bucle detector de impacto ha sido necesario realizar un hilo de ejecución simultánea al hilo principal, tal y como se comentó previamente. De esta

manera, en las líneas de código posteriores a la inicialización de librerías, se define un objeto del tipo *threading*, es decir, un hilo. En su interior, se establece la función *run*, que comprenderá todas las líneas de código que se ejecutarán en el hilo. De esta manera, el hilo no comenzará a ejecutar su función *run* hasta que se le indique, lo cual se comentará posteriormente.

De esta forma, las líneas de código comprendidas en la función *run* del hilo, siendo estas las que se ejecutarán una vez se indique, aparecen representadas en la Figura 35.

```
class HiloSensor(threading.Thread):
    def run(self):
        print("\nSe inicializa el hilo detector de impacto. \n")
        aceleracionFicheroX = pd.read_excel('/home/pi/Descargas/aceleracionx.xlsx')
        aceleracionFicheroY = pd.read_excel('/home/pi/Descargas/aceleraciony.xlsx')
        aceleracionFicheroZ = pd.read_excel('/home/pi/Descargas/aceleracionz.xlsx')

        diferenciaAceleracion = pd.DataFrame({
            'aceleracionx' : abs((aceleracionFicheroX['Aceleracion'])),
            'aceleraciony' : abs((aceleracionFicheroY['Aceleracion'])),
            'aceleracionz' : abs((aceleracionFicheroZ['Aceleracion'])),
        })

        vectorx = []
        vectory = []
        vectorz = []

        valorChoque = 8

        print("Se han abierto los ficheros. \n")
        print("Comienza el retardo \n")
        time.sleep(30)
        print("Finaliza el retardo \n")
```

Figura 35. Código de una sola ejecución del hilo detector de impacto

Las líneas de código expuestas en la figura anterior son de una única ejecución, que se realizará en el momento en que se indique la entrada al hilo. En ellas, se realiza la apertura de los ficheros que se va a proceder a leer, que disponen de los datos de aceleración tomadas por el sensor mediante el Arduino Mega. Además, se crean tres vectores, que se emplearán para guardar los datos correspondientes a las aceleraciones de cada eje.

Solo con la finalidad de demostrar el funcionamiento del código, y resultando completamente independiente a los parámetros reales a emplear para la detección de un impacto en el vehículo, se ha decidido escoger como valor límite el máximo de la diferencia de los módulos en cada instante obtenidos para los valores empleados. Sabiendo que la máxima diferencia es de 9, se ha establecido un valor límite denominado valorChoque de 8.

Debido a que el funcionamiento real sería realizando la lectura de valores obtenidos por el sensor, siendo estos no limitados, se ha decidido emplear un retardo de 30 segundos para lograr ampliar el tiempo de grabación sin necesidad de ampliar el número de valores guardados en el fichero.

De esta manera, una vez finalice el retardo de 30 segundos establecidos para lograr la grabación de cierta cantidad de vídeos, simulando así una situación más cercana a la real, se entra en el bucle de comparación de valores, el cual se muestra en la Figura 36.

```
for i in aceleracionFicheroX.index[0:]:
    if i < 1:
        vectorx.append(float(diferenciaAceleracion['aceleracionx'][i]))
        vectory.append(float(diferenciaAceleracion['aceleraciony'][i]))
        vectorz.append(float(diferenciaAceleracion['aceleracionz'][i]))
        i += 1
    else:
        restax = abs(vectorx[i-2]) - abs(vectorx[i-1])
        restay = abs(vectory[i-2]) - abs(vectory[i-1])
        restaz = abs(vectorz[i-2]) - abs(vectorz[i-1])
        vectorx.append(float(diferenciaAceleracion['aceleracionx'][i]))
        vectory.append(float(diferenciaAceleracion['aceleraciony'][i]))
        vectorz.append(float(diferenciaAceleracion['aceleracionz'][i]))
        modulo = (restax**2 + restay**2 + restaz**2)**0.5
        i += 1

    if modulo > valorChoque:
        print("Se detectó impacto. Se cierra hilo detector.\n")
        sys.exit()
```

Figura 36. Bucle detector de impacto del hilo secundario

En este bucle, el cual se ejecuta para un rango determinado por la longitud del fichero que contiene los valores correspondientes a la aceleración medida en el eje x se realiza, en primer lugar, únicamente la lectura del primer valor comprendido en cada fichero, para realizar posteriormente la resta del mismo con el siguiente obtenido inmediatamente posterior. Finalmente, se realiza el módulo de estos tres valores y se compara con el valor establecido, en caso de que sea superior, se llama a la salida del hilo y, por tanto, a la finalización del bucle de grabación.

Todo esto se realizaría únicamente cuando se llame a la ejecución de la función run, tal y como se comentó previamente. Esto se realizaría, en parte, con las siguientes líneas de código, expuestas en la Figura 37.

```
hilo1 = HiloSensor()
hilo1.start()
choque = hilo1.is_alive()
i = 1
print("Se comienza a grabar. \n")
```

Figura 37. Líneas de código previas a la entrada al bucle de grabación

En ellas, se asocia el hilo creado a una variable denominada “hilo1” y se ejecuta al comienzo. Además, se muestra cómo se actualiza la variable choque y la variable i, ambas empleadas en el bucle de grabación condicionado.

- Tratamiento de imágenes tras el impacto.

Debido a que la obtención de los vídeos mediante el módulo de cámara de la Raspberry Pi se realiza en formato .h264, es necesario realizar un tratamiento de los archivos para lograr la conversión al formato .mp4.

Para ello, se han empleado las líneas de código que se muestran en la Figura 38.

```
convertidor1 = "MP4Box -add Video" + str(i - 2) + ".h264 Video1.mp4"  
convertidor2 = "MP4Box -add Video" + str(i - 1) + ".h264 Video2.mp4"  
convertidor3 = "MP4Box -add VideoFinal.h264 Video13mp4"  
  
call([convertidor1], shell=True)  
call([convertidor2], shell=True)  
call([convertidor3], shell=True)  
  
borrar1 = "rm Video" + str(i - 2) + ".h264"  
borrar2 = "rm Video" + str(i - 1) + ".h264"  
borrar = "rm VideoFinal.h264"  
  
call([borrar1], shell=True)  
call([borrar2], shell=True)  
call([borrar3], shell=True)
```

Figura 38. Conversión de los clips obtenidos a formato .mp4

En primer lugar, se establecen tres *streams* que comprendan la misma cadena de caracteres en las que se cambia únicamente el nombre de los vídeos guardados en la memoria.

A continuación, mediante la librería *subprocess*, se ejecutan estos comandos mediante la ventana de comandos del Raspbian. De esta manera se logra realizar la conversión del formato .h264 a .mp4.

Finalmente, se realiza el proceso de borrado de archivos de video residuales, correspondiente a los del formato .h264, de manera análoga al anterior.

Una vez convertidos los archivos de vídeo al formato deseado, se continúa con la unión de los mismos empleando la librería *moviepy.editor*, tal y como se muestra en las siguientes líneas de código, expuestas en la Figura 39.

```
from moviepy.editor import *
Clip1 = VideoFileClip("Video1.mp4")
Clip2 = VideoFileClip("Video2.mp4")
Clip3 = VideoFileClip("Video3.mp4")
ClipFinal = concatenate_videoclips([Clip1, Clip2, Clip3])
ClipFinal.write_videofile("Resultado.mp4")

borrar4 = "rm Video1.mp4"
borrar5 = "rm Video2.mp4"
borrar6 = "rm Video3.mp4"

call([borrar4], shell=True)
call([borrar5], shell=True)
call([borrar6], shell=True)
```

Figura 39. Obtención del archivo de video final, compuesto por los clips grabados.

Se observa cómo se generan variables en las que se guardan los clips de vídeo deseados, y a continuación se unen y se guarda el archivo resultante con el nombre de Resultado en formato .mp4.

Finalmente, se procede al borrado de los archivos residuales en formato .mp4, empleando la metodología seguida anteriormente.

De manera justificativa, se adjunta al final de esta memoria un enlace para la visualización del vídeo obtenido tras la ejecución del código. De igual forma, se adjunta un enlace para la visualización de la captura de pantalla obtenida mientras se ejecuta el código, de manera que se aprecien todos aquellos archivos generados, tratados y eliminados. Cabe destacar que para la realización de este vídeo, al ejecutar la grabación de pantalla simultánea a la ejecución del código, el tiempo de tratamiento de los archivos generados se ve aumentado con respecto a la ejecución sin la grabación de pantalla. Esta documentación se encuentra adjunta en el Anexo VII.

3.3.2 LECTURA DEL SENSOR

La obtención de datos a partir de la lectura realizada por el sensor debería realizarse directamente estableciendo conexión con la Raspberry Pi. No obstante, debido a los problemas surgidos detallados anteriormente, se ha tenido que emplear un Arduino Mega para la obtención de los mismos.

Es por ello que, con el fin de lograr una mayor comprensión del código empleado y sus diferencias con respecto a su realización empleando una Raspberry Pi, se procederá a realizar una breve introducción de conceptos básicos de Arduino.

Los códigos de programación de cualquier programa realizado en Arduino mantienen la misma estructura, pues ella tiene predefinida una serie de bloques en las que se realizan las distintas partes del código, de manera ordenada. La estructura de un programa en Arduino se compone por los siguientes puntos:

- Inicialización de librerías.
- Definición de variables y constantes.
- Setup.
- Loop.

Siguiendo los puntos anteriores, se comenzaría inicializando todas aquellas librerías necesarias para la correcta ejecución del código, así como la inicialización de las variables que se emplearán en el mismo. Posteriormente, se ejecutarían aquellas líneas de código comprendidas en el Setup, de manera que únicamente se ejecuten al comienzo de la ejecución del código, de manera previa al bucle Loop. Este último, ejecuta en bucle todas aquellas líneas de código que comprenda, de manera indefinida y cíclica.

Además, como diferencia a considerar con respecto a la propuesta inicial a la hora de emplear únicamente la Raspberry Pi en la implementación, se debe aclarar que el lenguaje de programación que se emplea en Arduino está basado en C++, cuya programación presenta diferencias con respecto a la programación en Python. Además, Arduino dispone de un programa destinado a la creación y ejecución de los códigos denominado Arduino IDE, además de librerías de ejemplo y de mucha documentación.

Tras la breve introducción al programa empleado, lenguaje de programación, y estructura a seguir, en conclusión, tras conocer todos aquellos aspectos relevantes para emplear un Arduino, se procede a la explicación del código desarrollado.

En primer lugar, las librerías empleadas en el programa se muestran en las primeras líneas de código, expuestas en la Figura 40.

```
#include "I2Cdev.h"  
#include "MPU6050.h"  
#include "Wire.h"
```

Figura 40. Librerías empleadas para la lectura del sensor

Las utilidades de las librerías empleadas se exponen a continuación:

- **Librería i2cdev.h:** empleada para proveer interfaces simples a los dispositivos con los que se establezca conexión i2c. [56]
- **Librería Wire.h:** empleada para realizar la comunicación i2c empleando buses de datos entre el Arduino y el dispositivo deseado. [57]
- **Librería MPU6050.h:** empleada para poder obtener datos de lectura del sensor correspondiente. [58]

En las siguientes líneas de código, expuestas en la Figura 41, se realiza la creación de las variables que se emplearán para guardar los datos leídos de las aceleraciones en cada eje, determinados como x, y y z, así como la creación de un objeto denominado sensor correspondiente a la clase MPU6050.

```
MPU6050 sensor;  
int x, y, z;
```

Figura 41. Inicialización de variables para la lectura del sensor

Al emplear las líneas de código de esta manera, se establece como predefinida el registro del sensor a 0x68, pero esta se podría modificar a 0x69 indicándolo de la siguiente manera: MPU6050 sensor(0x69);

A continuación, en el bucle setup, correspondiente a la Figura 42, se inicializan la comunicación i2c, el sensor, y la comunicación serial. Esta última ha sido empleada para establecer comunicación con el ordenador, empleando la consola de pycharm. Esto con el objetivo de realizar la escritura de los valores leídos en ficheros almacenados en la memoria interna del ordenador, de manera que se puedan trabajar en el código implementado en la Raspberry Pi. Esto se contemplará en los siguientes apartados.

```
void setup() {  
  Serial.begin(9600);  
  Wire.begin();  
  sensor.initialize();  
}
```

Figura 42. Acciones ejecutadas previas a la lectura del sensor

Se ha especificado que la comunicación serial se inicialice en 9600 baudios, unidad que representa el número de símbolos por segundo en la transmisión. Esto delimita la cantidad de valores leídos que se muestran por segundo en la consola de simulación del Arduino IDE, y además, la cantidad de valores leídos que se envían por segundo a la consola de pycharm. Por ende, afecta de manera directa a los valores escritos en el fichero de datos que se empleará en el código de la Raspberry Pi.

Finalmente, se han ejecutado las líneas de código correspondientes a la obtención de los valores de aceleración, su tratamiento y su envío a la consola de Pycharm para su posterior escritura en ficheros. Esto se muestra en la Figura 43.

```
void loop() {  
    sensor.getAcceleration(&ax, &ay, &az);  
    float ax_m_s2 = ax * (9.81/16384.0);  
    float ay_m_s2 = ay * (9.81/16384.0);  
    float az_m_s2 = az * (9.81/16384.0);  
    Serial.println(ax_m_s2);  
    Serial.println(ay_m_s2);  
    Serial.println(az_m_s2);  
}
```

Figura 43. Bucle de lectura del sensor

Empleando la librería para el control del sensor MPU6050, se obtienen los valores de aceleración correspondientes a los ejes x, y y z, en función del rango en el que se establezca la obtención de los datos del acelerómetro, siendo los permitidos $\pm 2g$, $\pm 4g$, $\pm 8g$, y $\pm 16g$ para el acelerómetro comprendido en este sensor en concreto, donde g representa el valor de la gravedad. Además, el sensor ofrece un rango de lectura que comprende como valores mínimos, máximos y centrales los siguientes: -32768, 0 y 32768 respectivamente.

De esta manera, en las líneas de código mostradas en la figura anterior, se ha multiplicado por los valores comentados para obtener la aceleración en unidades del sistema internacional, m/s^2 .

Finalmente, se escriben los valores leídos por el puerto serial, y se establece un retardo de 0,1 segundos hasta obtener la siguiente lectura. Estos datos, enviados por comunicación serial, serán recogidos en un fichero almacenado en el portátil, tal y como se comentó previamente.

Cabe destacar que para lograr que las medidas obtenidas correspondan con valores coherentes resulta necesario realizar la calibración del sensor previo a su uso, de manera que se modifiquen sus valores dpm internos y se ajusten a los obtenidos tras la calibración del sensor. La realización de la calibración se comentará en su correspondiente apartado, denominado “3.2.4 Calibración del sensor”.

3.3.3 CREACIÓN DE FICHEROS DE DATOS

Para la creación de los ficheros que comprendan los datos leídos por el sensor, y que serán empleados posteriormente en la Raspberry Pi, se ha establecido comunicación serial entre el Arduino IDE y el ordenador. Para ello, se ha empleado el programa Pycharm, en el que se ha realizado el código de programación empleando Python.

Las librerías empleadas para la realización del código se muestran en la Figura 44, donde la primera se ha empleado para realizar la conexión serial con el Arduino y la segunda únicamente para establecer un retardo a la hora de recoger los datos proporcionados.

```
import serial
import time
```

Figura 44. Librerías empleadas para la creación de los ficheros de datos

A continuación, se procede a la creación de tres ficheros en formato .txt que comprenderán los valores de las aceleraciones leídas en los tres ejes que dispone la unidad de medida inercial. Estos archivos se crean en las rutas que se muestran en la figura expuesta y se determina que se abran para la escritura. Seguidamente, se asimila la variable denominada serialArduino a la comunicación serial que se establecerá, empleado el puerto COM3 y a 9600 baudios. Las líneas de código correspondiente a lo anteriormente descrito se muestran en la Figura 45.

```
aceleracionx = open('C:\\Users\\gibag\\Desktop\\Mitxt\\aceleracionx.txt', 'w')
aceleraciony = open('C:\\Users\\gibag\\Desktop\\Mitxt\\aceleraciony.txt', 'w')
aceleracionz = open('C:\\Users\\gibag\\Desktop\\Mitxt\\aceleracionz.txt', 'w')
serialArduino = serial.Serial("COM3", 9600)
```

Figura 45. Creación de los archivos y establecimiento de la comunicación serial

Finalmente, se procede a ejecutar un bucle para un rango de valores estimados por consideración propia de 0 a 200, obteniendo de esta manera 201 lecturas en cada eje. Esto se muestra en la Figura 46.

```
i = 0
j = 1
for i in range(0, 200):
    if j == 1:
        cad = serialArduino.readline().decode('ascii')
        print(cad)
        aceleracionz.write(cad)
        j = 2
        i += 1
    if j == 2:
        cad = serialArduino.readline().decode('ascii')
        print(cad)
        aceleracionx.write(cad)
        j = 3
        i += 1
    if j == 3:
        cad = serialArduino.readline().decode('ascii')
        print(cad)
        aceleraciony.write(cad)
        j = 1
        i += 1
```

Figura 46. Bucle de escritura de los valores obtenidos por comunicación serial

En este bucle, se escribe cada valor obtenido por comunicación serial en su fichero correspondiente, dependiendo del eje de medición. Para ello, se establecen condicionantes para cada lectura realizada, distribuyéndose de manera continuada y ordenada, según su orden de obtención.

Finalmente, los ficheros obtenidos tras la ejecución simultánea del código implementado en Arduino, en lenguaje C++, y del código implementado en pycharm, en lenguaje Python, contienen los datos de lectura del sensor estructurados tal y como se muestra en la Figura 47.

9.82
9.75
9.76
9.83

Figura 47. Estructura del fichero de datos escrito

La muestra presentada en la imagen anterior corresponde a los datos correspondientes a las lecturas realizadas en el eje de medición z. Se observa que, tal y como era de esperar al encontrarse en situación de reposo, los valores obtenidos son cercanos al de la aceleración de la gravedad ($9,81 \text{ m/s}^2$), al afectar esta fuerza al eje medido. La diferencia presentada resulta mínima, pudiendo variar e incluso aumentar su valor en caso de realizar nuevas mediciones en las que el sensor no se encuentre en la misma posición que disponía en la realización de la calibración. Esto se debería a la diferencia de posición del sensor, que presentaría un ángulo de desfase con respecto a la posición empleada para la calibración. En términos finales de la implementación, se trataría de realizar la soldadura y el correcto posicionamiento del sensor, así como su correspondiente calibración, para minimizar este error y otros que se pudieran obtener en los demás ejes. Esto último no será de aplicación en el presente Trabajo de Fin de Grado, simplemente se aporta como consideración adicional a tener en cuenta.

3.3.4 CALIBRACIÓN DEL SENSOR

Tal y como se comentó anteriormente, antes de proceder a la toma de lecturas del sensor se requiere realizar su calibración, de manera que se reduzcan los *offset* internos y se ajusten a las condiciones reales de medición.

El código para realizar esta calibración ha sido creado por Luis Ródenas, basado en el trabajo previo realizado por Jeff Rowberg en la librería MPU6050 y en la librería I2Cdev. Este código se encuentra disponible en la documentación de la librería empleada, y se dota de libertad de derechos de uso condicionada a realizar la referencia de los autores. De esta manera, se procederá a su empleo y se anexará al final de este trabajo, sin realizar su análisis [59].

No obstante, se comentará de manera escueta su metodología, así como los resultados obtenidos tras la calibración del sensor.

El programa empleado establece unos valores ideales de *offset*, a los que tratará de ajustar contemplando un error, pudiendo ser este modificado por el usuario según la precisión que se desee. Continúa realizando la lectura de los *offset* iniciales que presenta el sensor, y con el empleo de un filtro estabiliza estos valores, haciendo que los mismos

converjan a los ideales establecidos previamente. Una vez finaliza la calibración, el código devuelve por pantalla los valores de *offset* iniciales y los nuevos obtenidos, de manera que se conozca el error que presentarán las futuras lecturas.

Para el caso de la implementación, en vista a que la calibración realizada no finalizaba, se decidió modificar las líneas de código del programa de manera que se aumentara el error permitido, así como el valor de baudios. Las líneas de código modificadas se muestran en la Figura 48.

```
int acel_deadzone=10;  
int gyro_deadzone=3;
```

Figura 48. Valores modificados para la calibración

Los valores predeterminados por el autor del código son, de manera respectiva a lo expuesto en la figura anterior: 8 y 1. Al aumentar estas cifras se consigue ampliar el margen de error permitido en la calibración.

Finalmente, tras la realización de la calibración, el programa devolvió la información expuesta en la Figura 49.

```
FINISHED!  
=====  
RESULTS:  
Sensor data is listed in the format:  accelX  accelY  accelZ  gyroX  gyroY  gyroZ  
Sensor readings INCLUDING offsets:    2      3      16387  0      0      0  
Compare with IDEAL sensor readings:  0      0      16384  0      0      0  
-----  
Your MPU-6050 offsets:  
mpu.setXAccelOffset(-1628);  
mpu.setYAccelOffset(-2388);  
mpu.setZAccelOffset(1004);  
mpu.setXGyroOffset(100);  
mpu.setYGyroOffset(-74);  
mpu.setZGyroOffset(19);  
  
You can copy and paste the above offsets directly into your sketch. :)
```

Figura 49. Offset tras la calibración

Se observa cómo, en comparación a los valores ideales determinados, el error cometido resulta pequeño y por tanto aceptable.

4. LÍNEAS ABIERTAS Y FUTURAS

En este apartado se realiza una autoevaluación del modelo básico implementado del sistema propuesto, y se establecen propuestas de mejora del mismo y proyectos de ampliación que cubran nuevas funcionalidades.

4.1 PROPUESTAS DE MEJORA

En la realización de la implementación del modelo propuesto se ha tratado de obtener las funcionalidades básicas de la manera más óptima posible, evaluando los resultados obtenidos en función de la complejidad de realización. Es por ello que, en busca de la optimización del tiempo invertido en relación a los resultados obtenidos, se han aceptado errores considerados como aceptables debido a su magnitud.

De esta manera, las correcciones a los errores cometidos resultarían en una implementación completa en cuanto a calidad y precisión. Las medidas correctivas se comentarán a modo descriptivo en los siguientes subapartados.

4.1.1 METODOLOGÍA DE TOMA DE IMÁGENES

La principal propuesta de mejora a aplicar al sistema empleado sería realizar un cambio de la manera en que se toman las imágenes.

En el código implementado, se toman videos de una determinada duración, que se van guardando y borrando en la memoria de manera continua. Una vez se detecta el impacto, se graba un último vídeo posterior a este momento y se realiza la conversión de formato y la unión de todos los clips tomados. Esto provoca una pequeña pérdida de información entre grabaciones, correspondiente a una pequeña cantidad de fotogramas. Debido a que esta pérdida de fotogramas es poco relevante y el resultado obtenido resulta suficiente para justificar el funcionamiento del sistema, se ha decidido tomar como aceptable el error cometido.

A modo justificativo se ha realizado una simulación del código simplificado, obteniendo los valores de tiempo al momento de finalización de grabación y al momento del comienzo de grabación del siguiente clip. Realizando la diferencia de estos valores, se han obtenido una serie de valores, cuyos máximo y mínimo son 0,104 y 0,085 segundos, cuyo valor promedio resulta de 0,088 segundos para un número de pruebas de 20 repeticiones. El código empleado para la realización de estas pruebas se adjunta en el Anexo V. Considerando que el tiempo promedio de reacción de una persona ante un estímulo visual es de 0,25 segundos, se puede determinar que el error cometido no afecta de manera grave a la obtención de resultados.

No obstante, se podría solventar esta deficiencia cambiando la metodología de toma de imágenes empleada. La librería utilizada para la captura de imágenes permite realizar la toma independiente de fotogramas, que se borrarían de manera análoga al proceso que realiza el programa actual. Sería necesario cambiar el método de conversión de formato y procesado de fotogramas para obtener un vídeo final.

El principal motivo de no emplear esta metodología en el código actual se debe a la complicación que esta última requiere y a la poca diferencia entre los resultados que se obtienen empleando una u otra metodología. Es por ello que, asumiendo el pequeño error cometido comentado anteriormente, se justifica el método aplicado.

4.1.2 AJUSTE DE PARÁMETROS

El modelo a implementar requiere, además de la grabación de imágenes en bucle, la lectura del sensor y la obtención de datos a partir del mismo, así como su posterior tratamiento para lograr la detección de un impacto.

La metodología de detección del impacto aplicada se basa en la comparación de los valores de aceleración obtenidos en los tres ejes del sensor, por parte de los acelerómetros internos que dispone. Debido a que la trayectoria de un vehículo no resulta lineal en una única dirección y sentido, se ha empleado la comparación del módulo correspondiente a los parámetros de aceleración de cada eje. Se realiza la diferencia entre el valor del módulo de la aceleración al momento de la lectura y su inmediato anterior, de

manera que el resultado de la diferencia en caso de impacto sea significativo en comparación a los valores de diferencia obtenidos en una conducción normal, estableciendo para ello un rango de valores determinados.

En la implementación del modelo se ha omitido el empleo de datos reales referentes a la producción de un impacto de un vehículo, pues la metodología a seguir y el tratamiento de los datos resulta igual en ambos casos. El objetivo de este bloque de código es el de lograr la obtención de valores a partir del código y realizar su tratamiento, independientemente de si estos valores son referentes al impacto de un vehículo o no.

No obstante, a la hora de mejorar la implementación se propone el empleo de datos reales, bien obtenidos de bases de datos o empíricamente, para lograr un ajuste a la detección de un impacto en un vehículo. Para la obtención de manera empírica, se propone realizar la toma de datos de una conducción normal durante un tiempo aceptable, que permita obtener datos suficientes referentes a los cambios producidos en la aceleración del vehículo. Posteriormente, se propone realizar el análisis de estos datos, gráficamente y mediante la obtención de su función correspondiente. De esta manera, se podrían determinar los valores límite en la función correspondiente a la diferencia de los valores de aceleración, correspondientes a los picos de la gráfica, es decir, a los máximos y mínimos de la función. De esta manera, se determina que los valores de diferencia entre aceleraciones correspondientes al momento de un impacto probablemente se encontrarían fuera del rango determinado, comprendido entre los máximos y mínimos anteriores, según la fiabilidad de los datos obtenidos para el estudio.

Debido a la complejidad de la obtención de datos reales y su análisis, así como el tiempo necesario a invertir para ello, se ha omitido en la realización de la implementación del modelo propuesto. Tal y como se comentó anteriormente, esto no resulta relevante, pues la obtención de los datos y su tratamiento se realizaría de igual manera. Simplemente variaría la magnitud del impacto detectado, es decir, la magnitud de las aceleraciones obtenidas así como su frecuencia de variación, parámetros relevantes a la hora de detectar un impacto del vehículo, pero no a la hora de demostrar la obtención de valores y su tratamiento, objeto de la realización de la implementación simplificada.

4.2 PROYECTOS DE AMPLIACIÓN

Con la finalidad de lograr una finalización más completa del trabajo realizado, además de dotar de posibles ampliaciones del proyecto propuesto, en cuanto a funcionalidades a implementar se refiere, se ha decidido realizar la redacción de este subapartado. En este, se tratarán de abarcar posibles y más relevantes mejoras en las funcionalidades del proyecto, que doten de posibles ampliaciones en las líneas abiertas del sistema.

4.2.1 ADICIÓN DE GRABACIÓN INTERNA O EXTERNA TRASERA

Si bien el empleo de una única cámara para la toma de imágenes puede resultar suficiente en la mayoría de casos en los que se presente un impacto, la información que esta proporcione puede llegar a resultar limitada para ocasiones en las que el impacto se produzca en la zona posterior trasera del vehículo, o bien cuando se produzca por imprudencias del conductor o de los ocupantes del vehículo. En estos últimos casos, la aportación de evidencias por parte de la *dash cam* puede llegar a resultar irrelevante al no disponer de documentación gráfica directa de las causas del impacto.

De esta manera, la adición de una cámara adicional a la principal puede ayudar a la obtención de más información tras la detección de un impacto, solventando así esta deficiencia. Se abren dos opciones de ampliación: adición de grabación interna del vehículo o adición de grabación externa trasera.

Para ambos casos se requeriría la adición de una segunda cámara que permita, de manera análoga a la cámara principal, obtener información gráfica de su entorno. En vista de que los modelos de Raspberry Pi únicamente disponen de un conector CSI para la adición de una cámara, se debería emplear otra alternativa de conexión de la misma.

La solución más sencilla y viable sería la adición de una cámara que permita realizar su conexión por puerto USB y que, preferiblemente, sea de un tamaño reducido. Al realizar la adición de una cámara por puerto USB, en vez de un módulo destinado

para la conexión con la Raspberry Pi, se debería garantizar la compatibilidad entre dispositivos y entre las librerías empleadas para su control y programación. De esta forma, y tras consultar la bibliografía correspondiente a la librería empleada en la implementación, siendo esta Picamera [49], se determina la no compatibilidad de esta librería con el tipo de dispositivo requerido, al estar ésta destinada a la programación y control de módulos de cámara cuyo uso es destinado a la Raspberry Pi.

Como solución a esta incompatibilidad se propone el empleo de otras librerías para realizar la programación de esta segunda cámara. Una de estas librerías es fswebcam [60], que permite realizar la toma de imágenes o fotogramas mediante una cámara conectada por puerto USB. En comparación a la librería empleada para la cámara principal, esta dispone de más limitaciones en cuanto a funcionalidades se refiere, pero al emplearla en conjunto a otras librerías se pueden obtener resultados suficientes para la mejora propuesta.

De esta manera, se realizaría la captura de fotogramas de manera análoga a la implementación realizada, y más concretamente a la propuesta de mejora planteada en el apartado “4.1.1 Metodología de Toma de Imágenes”. Estas se realizarían en un hilo independiente al principal, de manera que se ejecuten de forma simultánea. Posterior a la toma de fotogramas, se realizaría su procesamiento para lograr un vídeo realizado con la consecución de dichos fotogramas. Para ello, se propone aplicar la librería mencoder [61], con la que se puede realizar la unión de estos fotogramas para lograr un vídeo final.

Es importante contemplar la posibilidad de aumento en el retardo obtenido entre la grabación de los vídeos tras las pruebas realizadas en el apartado “4.1.1 Metodología de Toma de Imágenes”. Sería necesario realizar nuevamente estas pruebas conjuntamente con la segunda cámara funcionando, de manera que se realice el estudio de la pérdida de información para esas condiciones. En caso de resultar desfavorable, se propone aplicar la medida contemplada en ese mismo apartado, relativa a realizar la toma de imágenes por medio de fotogramas en vez de vídeos consecutivos.

Cabe destacar que, en aspectos generales, las imágenes obtenidas de esta segunda cámara resultarán de calidad inferior a la obtenida con la cámara principal. Las principales causas de esta diferencia se deben al puerto de conexión empleado, siendo el CSI

destinado para la adición de una cámara y, por tanto, estando este mejor diseñado para tal fin; y a la limitación en la programación de la toma de imágenes debida a la no compatibilidad con ciertas librerías, teniendo para este caso un menor número de opciones que, además, dotan de una menor cantidad de funciones. No obstante, y aún con las limitaciones que se presentan para la adición de una segunda cámara, se determina que esta mejora puede resultar de interés a la hora de realizar ampliaciones al proyecto propuesto, pues permite la obtención de información gráfica adicional de la situación que ha provocado un impacto.

4.2.2 ALMACENAMIENTO EN NUBE

Uno de los mayores problemas que pueden surgir al emplear una *dash cam* es la pérdida de las imágenes guardadas tras la detección de un impacto. Esta pérdida puede deberse bien a la corrupción de archivos por algún fallo en el sistema o bien a un error humano, pudiendo ser este el borrado del vídeo por error. Con la finalidad de prevenir estas situaciones, y para garantizar una mayor seguridad de los archivos obtenidos, se propone realizar la ampliación al sistema para lograr el almacenamiento en la nube. De esta manera, se podría establecer un guardado manual por parte del usuario y un guardado periódico que se realice de manera automática.

El principal inconveniente a la hora de realizar esta ampliación se debe a que, al ser un sistema cuyo uso previsto es en vehículos, se debe dotar de una conexión a internet para lograr esta funcionalidad. La Raspberry Pi, tal y como se ha expuesto en apartados anteriores, dispone de un módulo integrado que permite realizar la conexión WiFi, por lo que para establecer esta funcionalidad bastaría con indicar al usuario de la necesidad de conexión de este dispositivo a una red WiFi externa. Además, en caso de querer realizar la actualización de archivos de manera periódica y automatizada, se debería garantizar la conexión a la red de manera fija y estable, como mínimo mientras el dispositivo se encuentre activo.

Por otro lado, la determinación de la nube a emplear podría dejarse a elección del usuario, pudiendo emplear cualquiera de las disponibles más conocidas, como pueden ser Google Drive [62], Amazon Cloud Drive [63], Microsoft OneDrive [64], Box [65], iCloud [66], entre otros muchos.

4.2.3 CONTROL POR VOZ

Debido a que el sistema planteado en este documento es orientado al uso del mismo en vehículos, resulta de especial interés lograr un control por voz que resulte en un sistema de manos libres, que permita al usuario controlar el dispositivo, la toma de imágenes o el borrado de las mismas, mientras conduce sin presentar riesgos a la conducción.

Para lograr esta funcionalidad, en primer lugar sería necesario añadir un micrófono que permita tomar datos externos y transmitirlos a la Raspberry Pi. En el mercado existen distintas opciones, que van desde implementar una tarjeta de sonido hasta realizar la conexión con un micrófono mediante USB, siendo esta última la más sencilla y económica, y resultando suficiente para la funcionalidad que se busca implementar.

Por otro lado, una vez se obtengan los datos tomados desde el micrófono, será necesario realizar un tratamiento de los mismos para poder lograr el reconocimiento por voz.

Para ello, existen distintas librerías de Python que se pueden aplicar. Una de ellas es Spkcat [67], destinada a la conversión de archivos en formato .wav a texto. Algunos de los lenguajes que soporta son inglés, alemán, francés, italiano o catalán, entre otros.

Para solventar la falta del idioma español, la manera más sencilla es establecer unos comandos cortos y básicos en un idioma genérico, como puede ser el inglés. Se podrían emplear las palabras “*Stop*” para dejar de grabar, “*Start*” para comenzar la grabación y “*Remove*” para borrar archivos.

En relación a la codificación del programa, sería necesario realizar la lectura del micrófono en otro hilo que se ejecute de manera simultánea al hilo principal, de funcionamiento análogo a lo que se realiza con la unidad inercial de medida.

En el hilo creado, se realizaría la conversión de los archivos .wav tomados mediante el micrófono a texto, que se guardaría en variables y se usaría como condicionante a la hora de realizar las acciones propuestas.

4.2.4 TRATAMIENTO DE IMÁGENES.

Implementar en el proyecto el tratamiento de imágenes podría añadir funcionalidades adicionales como pueden ser reconocimiento de peatones, reconocimiento de matrículas o reconocimiento de la variación en las luces de frenado, entre otras posibilidades. Todo esto puede servir tanto de ayuda a la conducción dotando de asistencia al conductor, o bien la posibilidad de estudio de conductas delictivas realizadas por otros conductores.

Para lograr esta funcionalidad orientada a cualquiera de las aplicaciones descritas, sería necesario emplear una nueva librería que dotara de métodos y funciones capaces de, como mínimo y siendo lo principal requerido, realizar mejoras de imagen y segmentación de las mismas.

Una de las librerías diseñadas para el tratamiento y procesamiento de imágenes es scikit-image [68]. Con ella se pueden emplear distintas funciones y métodos para realizar el filtrado, suavizado, segmentación, implementación de los distintos espacios de colores para su estudio y separación en gradientes.

De esta manera y aplicando, por ejemplo, esta librería, se abren dos opciones en cuanto a funcionalidad a dotar al sistema: tratamiento y procesamiento de imágenes en tiempo real y simultáneo a la conducción o tratamiento y procesamiento de imágenes tras la detección de un impacto.

En el primer caso, realizar el tratamiento y procesamiento de imágenes en tiempo real de conducción puede dotar de ayuda de conducción al usuario, y aportar de esta manera una mayor seguridad en la misma. Este tratamiento y procesamiento de imágenes sería orientado a objetos y figuras relevantes, como puede ser la detección de personas o animales próximos al vehículo, o la detección de una variación en las luces de frenado, que en conjunto al estudio de la proximidad en la que se encuentren podrían resultar en una colisión trasera. De esta manera, al realizar la detección de algún objeto o figura determinada dentro de unos parámetros límite de seguridad, se emitiría una señal de aviso al conductor.

Por otro lado, en el caso de realizar el tratamiento y procesamiento de imágenes posterior a la detección de un impacto, se podría dotar de información relevante como pueden ser datos sobre la matrícula de los vehículos implicados en el mismo. De esta manera, sería de interés realizar el tratamiento de imágenes tras la detección de un impacto de manera previa al borrado y procesado de vídeos o fotogramas, según el método empleado, pues de esta manera se aseguraría que en caso de disponer de esta información, se pueda realizar su tratamiento y no se elimine. Además, realizar un procesamiento de imágenes más complejo puede dotar al sistema de una mayor precisión en las imágenes comprendidas en el rango de grabación guardado, pues se podría determinar cuáles de las imágenes tomadas son verdaderamente relevantes y contengan información

Cabe destacar que para la realización de este tratamiento y procesamiento de imágenes sería necesario realizar un estudio y análisis de técnicas orientadas a ello. Una de estas técnicas a destacar, que podría no resultar suficiente al emplearla como única, es la segmentación. Esta trata de lograr la identificación de determinada información de interés, como pueden ser los bordes de una figura, para así lograr realizar la separación de esta con respecto al fondo. Este ámbito de estudio es muy amplio y excede el propósito del presente Trabajo de Fin de Grado, pero se ha decidido comentar estos aspectos básicos para recalcar la importancia de realizar un estudio de estas técnicas para obtener unos resultados óptimos y aceptables en caso de realizar la adición de esta funcionalidad.

Finalmente, se deberían considerar las capacidades otorgadas por la Raspberry Pi y las limitaciones que esta pueda presentar frente a una implementación que abarque lo anteriormente comentado, pues según la complejidad del procesamiento de imágenes se podría llegar a sobrepasar la capacidad de análisis en tiempo real del procesador y, en este caso, se debería sustituir por uno de prestaciones superiores. De esta manera, podría llegar a ser necesario cambiar el componente principal empleado en este proyecto, la Raspberry Pi, y proceder a la realización del diseño de una placa de circuito impreso que se ajuste más a las dotaciones deseadas.

5. CONCLUSIONES / CONCLUSIONS

CONCLUSIONES

Tras la realización del proyecto expuesto en esta memoria, se puede concluir en que se ha logrado cumplir con los objetivos propuestos inicialmente, tanto aquellos referentes a aspectos técnicos del diseño como aquellos correspondientes a objetivos personales por parte del autor de este documento.

De esta manera, se determina haber obtenido unos resultados favorables con la implementación simplificada, al haber comprobado el funcionamiento básico del sistema y la comunicación entre todos los dispositivos y componentes implicados en el mismo. Además del estudio realizado de los elementos empleados en el diseño para el fin de las funcionalidades básicas propuestas, se ha logrado realizar un estudio de todas aquellas funcionalidades que, estando relacionadas con el preámbulo en el que se ha basado la realización de este trabajo, se podrían lograr en líneas futuras aplicando distintos proyectos de mejora y ampliación, demostrando así el gran abanico de posibilidades que se abren con el uso de este dispositivo.

CONCLUSIONS

At the end of the realization of the project included in this document, it can be concluded that it has been possible to achieve the objectives initially proposed, those referring to technical aspects of the design and those corresponding to the personal objectives of the author of this document.

In this way, satisfactory results have been obtained with the simplified implementation, having verified the basic operation of the system and the communication between all the devices and components involved in it. In addition to the study carried out of the elements used in the design for the purpose of the proposed basic functionalities, it has been reached to make a study of all those functionalities that, being related to the preamble on which the realization of this work has been based, could be achieved in future lines by applying different improvement and expansion projects, demonstrating the wide range of possibilities that open up with the use of this device.

6. PRESUPUESTO DEL MODELO A IMPLEMENTAR

A continuación se realiza un desglose del presupuesto correspondiente a la realización de un único dispositivo en fase de prototipado. Cabe destacar que, en la fase de producción, se deberían incluir las horas de testeo y validación del dispositivo, así como la programación del mismo, el encapsulamiento, y la producción de un número amplio de unidades para su lanzamiento al mercado, siempre y cuando se cuente con la autorización legal para su instalación en vehículos.

Código	Tipo	Modelo	Descripción	Unidad	Coste unitario	Coste	
1. Componentes							
101	Raspberry Pi	Raspberry Pi 3 Model B	Placa de circuito impreso prediseñada Raspberry Pi	1 (ud.)	32,55 €	32,55 €	
102	Módulo de Cámara	Módulo de Cámara Raspberry Pi V2	Módulo compatible con todos los modelos de Raspberry pi, 8 Megapíxeles, 1080p a 30 fps, 720p a 60 fps, 480p a 90 fps, conexión por puerto CSI	1 (ud.)	28,97 €	28,97 €	
103	Módulo IMU	GY-521	Módulo con unidad de medida inercial de 6 ejes integrada, compatible con Raspberry Pi 3 Model B	1 (ud.)	3,96 €	3,96 €	
104	Tarjeta microSD	Sandisk 64 GbB	Tarjeta microSD de 64GB, 100 MB/s de lectura y 60 MB/s de escritura	1 (ud.)	14,83 €	14,83 €	
105	Adaptador Toma Mechero	Qilive 3.0	Adaptador para la toma de mechero de vehículos, con voltaje de salida de 5V y amperaje de salida de 3ª	1 (ud.)	4,64 €	4,64 €	
106	Cable adaptador USB - microUSB	Qilive	Cable adaptador con entrada USB y salida microUSB, con amperaje de salida de 2,4 A	1 (ud.)	12,07 €	12,07 €	
2. Honorarios técnicos							
201	Mano de obra		Diseño e Implementación realizada por Ingeniero Técnico Industrial	300 (horas) [equivalente 12 ECTS]	28,00€	8.400,00€	
						Subtotal	8.497,02 €
						I.G.I.C. (7%)	594,79 €
						TOTAL	9.091,81 €

7. BIBLIOGRAFÍA

- [1] Anuario Estadístico de Accidentes 2020. Disponible en:
<https://www.dgt.es/menusecundario/dgt-en-cifras/dgt-en-cifras-resultados/dgt-en-cifras-detalle/?id=00819> Último acceso: agosto de 2022
- [2] Fechas de obligatoriedad de principales sistemas de seguridad en vehículos. Disponible en: <https://www.eurotaller.com/noticia/sabias-que-el-abs-el-cinturon-de-seguridad-y-el-airbag-son-obligatorios-en-los-vehiculos> Último acceso: agosto de 2022
- [3] Uso de Dash Cam en distintas localizaciones. Disponible en:
<https://es.wikipedia.org/wiki/Dashcam> Último acceso: diciembre de 2021
- [4] Descripción Raspberry Pi. Disponible en:
<https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/> Último acceso: diciembre de 2021
- [5] Halfacree, G. (2020). *La Guía Oficial de Raspberry Pi para Principiantes. Cómo usar tu nuevo ordenador*. Raspberry Pi Press.
- [6] Introducción a Raspberry Pi. Disponible en:
<https://www.digikey.es/es/articles/10-things-to-know-before-starting-a-raspberry-pi-project> Último acceso: diciembre de 2021
- [7] Modelos de Raspberry Pi en el mercado. Disponible en:
<https://www.raspberrypi.com/products/> Último acceso: marzo de 2022
- [8] Datos Hardware modelos Raspberry Pi. Disponible en:
<https://www.raspberrypi.com/documentation/computers/compute-module.html>
Último acceso: marzo de 2022
- [9] Comparativa modelos Raspberry Pi. Disponible en:
<https://www.comohacer.eu/comparativa-y-analisis-raspberry-pi-vs-competencia/>
Último acceso: marzo de 2022
- [10] Comparativa modelos Raspberry Pi. Disponible en:
<https://raspberryparatorpes.net/raspberry-pi-tabla-tecnica-completa/> Último acceso: marzo de 2022
- [11] Pinout según modelo Raspberry Pi. Disponible en:
<https://www.hwlibre.com/gpio-raspberry-pi/> Último acceso: marzo de 2022
- [12] Esquemático Raspberry Pi 3 Model B. Disponible en:
<https://datasheets.raspberrypi.com/rpi3/raspberry-pi-3-b-reduced-schematics.pdf>
Último acceso: marzo de 2022

- [13] Guía de asignación de pines Raspberry Pi. Disponible en:
<https://es.pinout.xyz/> Último acceso: marzo de 2022
- [14] Comunicación UART. Disponible en:
https://www.rohde-schwarz.com/es/productos/test-y-medida/osciloscopios/educational-content/que-es-uart_254524.html Último acceso:
abril de 2022
- [15] Comunicación I2C. Disponible en:
<https://hetpro-store.com/TUTORIALES/i2c/> Último acceso: abril de 2022
- [16] Comunicación I2C. Disponible en:
<https://www.digikey.es/es/articles/why-the-inter-integrated-circuit-bus-makes-connecting-ics-so-easy/> Último acceso: abril de 2022
- [17] Comunicación I2C. Disponible en:
<https://programarfacil.com/blog/arduino-blog/comunicacion-i2c-con-arduino/> Último
acceso: abril de 2022
- [18] Raspberry Pi Imager. Disponible en:
<https://www.raspberrypi.com/software/> Último acceso: abril de 2022
- [19] Sistemas Operativos soportados por Raspberry Pi. Disponible en:
<https://www.profesionalreview.com/2021/09/10/sistema-operativo-raspberry-pi/>
Último acceso: mayo de 2022
- [20] Raspberry Pi Os. Disponible en:
https://es.wikipedia.org/wiki/Raspberry_Pi_OS/ Último acceso: mayo de 2022
- [21] Protocolo Secure Shell. Disponible en:
https://es.wikipedia.org/wiki/Secure_Shell Último acceso: junio de 2022
- [22] Software VNC. Disponible en:
<https://www.geeknetic.es/VNC/que-es-y-para-que-sirve> Último acceso: junio de 2022
- [23] Lenguaje Python. Disponible en:
<https://www.python.org/> Último acceso: mayo de 2022
- [24] Resolución de imagen. Disponible en:
https://es.wikipedia.org/wiki/Resoluci%C3%B3n_de_imagen Último acceso: mayo de
2022
- [25] Distancia focal y ángulo de visión. Disponible en:
<https://www.nikon.com.mx/learn-and-explore/a/tips-and-techniques/entendiendo-la-distancia-focal.html> Último acceso: mayo de 2022

- [26] Distancia focal y ángulo de visión. Disponible en:
<https://captureheatlas.com/es/que-es-la-distancia-focal/> Último acceso: mayo de 2022
- [26] Resolución de vídeo. Disponible en:
<https://www.definicionabc.com/tecnologia/resolucion-video.php> Último acceso: mayo de 2022
- [27] Resolución de vídeo. Disponible en:
<https://www.reneelab.es/resolucion-4k-2k-1080p-720p.html> Último acceso: mayo de 2022
- [28] Módulo de Cámara de Raspberry Pi V2. Disponible en:
<https://www.raspberrypi.com/products/camera-module-v2/> Último acceso: abril de 2022
- [29] Unidad de medida inercial. Disponible en:
<https://unisialia.com/que-es-imu-funcionamiento-construccion-aplicaciones/> Último acceso: mayo de 2022
- [30] Movimientos de cabeceo, alabeo y guiñada. Disponible en:
https://es.wikipedia.org/wiki/Ejes_del_avi%C3%B3n Último acceso: julio de 2022
- [31] Movimiento de precesión de un giróscopo. Disponible en:
<https://mriquestions.com/energy-for-precession.html> Último acceso: julio de 2022
- [32] Datasheet MPU6050. Disponible en:
<https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf>
Último acceso: julio de 2022
- [33] Esquemático del conexionado interno del GY-521. Disponible en:
<https://www.hotmcu.com/gy521-mpu6050-3axis-acceleration-gyroscope-6dof-module-p-83.html> Último acceso: julio de 2022
- [34] Valores de alimentación para Raspberry Pi. Disponible en:
<https://www.raspberrypi.com/documentation/computers/raspberry-pi.html#power-supply> Último acceso: abril de 2022
- [35] Medios de alimentación para Raspberry Pi. Disponible en:
<https://thepihut.com/blogs/raspberry-pi-tutorials/how-do-i-power-my-raspberry-pi>
Último acceso: abril de 2022
- [36] Alimentación desde vehículo. Disponible en:
<https://www.xataka.com/vehiculos/esta-es-la-razon-por-la-que-los-puertos-usb-de-tu-coche-son-tan-lentos-al-recargar-tu-movil> Último acceso: abril de 2022

- [37] Tarjeta microSD para Raspberry Pi. Disponible en:
<https://www.raspberrypi.com/documentation/computers/getting-started.html#sd-cards>
Último acceso: abril de 2022
- [38] Tarjeta microSD Sandisk 64GB. Disponible en:
<https://www.westerndigital.com/es-es/products/memory-cards/sandisk-nintendo-switch-microsd#SDSQXAT-064G-GNCZN> Último acceso: abril de 2022
- [39] Instalación de Python-smbus. Disponible en:
<https://packages.debian.org/buster/armhf/python-smbus/download> Último acceso:
agosto de 2022
- [40] Error de claves públicas. Disponible en:
https://hijosdeinit.gitlab.io/howto_solucionar_error_gpg_clave_publica_repositorio_Debian_y_derivados
- [41] Configuración para I2C. Disponible en:
<https://www.abelectronics.co.uk/kb/article/1/i2c-part-2---enabling-i-c-on-the-raspberry-pi> Último acceso: agosto 2022
- [42] Python IDLE. Disponible en:
<https://docs.python.org/es/3/library/idle.html> Consultado en agosto de 2022 Último
acceso: agosto de 2022
- [43] Arduino IDE. Disponible en:
<https://www.arduino.cc/en/software> Último acceso: agosto de 2022
- [44] Pycharm. IDE de Python. Disponible en:
<https://www.jetbrains.com/es-es/pycharm/> Último acceso: agosto de 2022
- [45] Librería io. Disponible en:
<https://docs.python.org/es/3.9/library/io.html> Último acceso: agosto de 2022
- [46] Librería os. Disponible en:
<https://docs.python.org/es/3.10/library/os.html> Consultado en agosto de 2022 Último
acceso: agosto de 2022
- [47] Librería sys. Disponible en:
<https://docs.python.org/es/3/library/sys.html> Último acceso: agosto de 2022
- [48] Librería threading. Disponible en:
<https://docs.python.org/es/3.8/library/threading.html> Último acceso: agosto de 2022
- [49] Librería picamera. Disponible en:
<https://picamera.readthedocs.io/en/release-1.13/> Último acceso: agosto de 2022

- [50] Librería time. Disponible en:
<https://docs.python.org/es/3/library/time.html> Último acceso: agosto de 2022
- [51] Librería subprocess. Disponible en:
<https://docs.python.org/3/library/subprocess.html> Último acceso: agosto de 2022
- [52] Librería Pandas. Disponible en:
<https://pandas.pydata.org/> Último acceso: agosto de 2022
- [53] Librería moviepy.editor. Disponible en:
<https://pypi.org/project/moviepy/> Último acceso: agosto de 2022
- [54] Librería openpyxl. Disponible en:
<https://openpyxl.readthedocs.io/en/stable> Último acceso: agosto de 2022
- [55] Librería numpy. Disponible en:
<https://numpy.org/> Último acceso: agosto de 2022
- [56] Librería i2cdev.h. Disponible en:
<https://www.i2cdevlib.com/usage> Último acceso: agosto de 2022
- [57] Librería wire.h. Disponible en:
<https://www.arduino.cc/reference/en/language/functions/communication/wire/> Último
acceso: agosto de 2022
- [58] Librería MPU6050.h. Disponible en:
<https://github.com/electroniccats/mpu6050> Último acceso: agosto de 2022
- [59] Código de calibración de MPU6050. Disponible en:
[https://github.com/blinkmaker/Improved-MPU6050-
calibration/blob/master/MPU6050_offset_calibration_UPDATED.ino](https://github.com/blinkmaker/Improved-MPU6050-calibration/blob/master/MPU6050_offset_calibration_UPDATED.ino) Último acceso:
agosto de 2022
- [60] Librería fswebcam. Disponible en:
<http://www.sanslogic.co.uk/fswebcam/>. Último acceso: septiembre de 2022
- [61] Librería mencoder. Disponible en:
<http://www.mplayerhq.hu/design7/news.html> Último acceso: septiembre de 2022
- [62] Google Drive. Disponible en:
https://www.google.com/intl/es_es/drive/ Último acceso: septiembre de 2022
- [63] Amazon Cloud Drive. Disponible en:
<https://aws.amazon.com/es/products/storage/> Último acceso: septiembre de 2022
- [64] Microsoft OneDrive. Disponible en:

<https://www.microsoft.com/es-es/microsoft-365/onedrive/online-cloud-storage> Último

acceso: septiembre de 2022

[65] Box. Disponible en:

<https://www.box.com/es-419> Último acceso: septiembre de 2022

[66] iCloud. Disponible en:

<https://www.icloud.com/iclouddrive> Último acceso: septiembre de 2022

[67] Librería Spchcat. Disponible en:

<https://github.com/petewarden/spchcat#spchcat> Último acceso: agosto de 2022

[68] Librería Scikit-image. Disponible en:

<https://scikit-image.org/> Último acceso: septiembre de 2022

[69] Imagen Raspberry Pi. Disponible en:

<https://ardubasic.wordpress.com/2014/05/04/arduino-mega-2560-revision-3> Último

acceso: septiembre de 2022

[70] Imagen de Arduino Mega. Disponible en:

<https://arduproject.es/mpu6050-y-su-programacion/> Último acceso: septiembre de 2022

[71] Imagen de GY-521 (MPU6050). Disponible en:

<https://www.aranacorp.com/es/tag/raspberry-pi-es/> Último acceso: septiembre de 2022

ANEXO I

ANEXO I.

ESTUDIO DE MERCADO DEL PRODUCTO EN LA ACTUALIDAD

TABLA COMPARATIVA MODELOS MARCA GARMIN					
	Garmin Dash Cam Mini 2	Garmin Dash Cam 47	Garmin Dash Cam 57	Garmin Dash Cam 67W	Garmin Dash Cam Tandem
PRECIO (€)	129,99	179,99	199,99	249,99	349,99
TAMAÑO (cm) ancho-largo-profundidad	3,13 x 5,33 x 2,91	5,62 x 4,05 x 2,19	5,62 x 4,05 x 2,19	5,62 x 4,05 x 2,19	5,50 x 4,10 x 2,35
PESO (gr)	32,1	60,1	60,5	57,0	64,4
SOPORTE	Adhesivo de bajo perfil	Adhesivo/magnético de bajo perfil	Adhesivo/magnético de bajo perfil	Adhesivo/magnético de bajo perfil	Magnético de bajo perfil
ALIMENTACIÓN	USB-microUSB (supercondensador)	Batería Ión-Litio recargable (duración hasta 30 min)	Batería Ión-Litio recargable (duración hasta 30 min)	Batería Ión-Litio recargable (duración hasta 30min)	(supercondensador)
RESOLUCIÓN CÁMARA	1080p HD	1080p HD	1440 HD	1440 HD	1440 HD (exterior) 720p (interior)
VELOCIDAD FOTOGRAMAS	Hasta 30 FPS	Hasta 30 FPS	Hasta 60 FPS	Hasta 60 FPS	Hasta 30 FPS
ÁNGULO DE VISIÓN (°)	140 (diagonal)	140 (diagonal)	140 (diagonal)	180 (diagonal)	180 (exterior e interior)
ORIENTACIÓN	Grabación exterior	Grabación exterior	Grabación exterior	Grabación exterior	Grabación exterior Grabación interior
PANTALLA INTEGRADA	No	Sí	Sí	Sí	No
TAMAÑO PANTALLA (Pulgadas)	-----	2,0 (5,1 cm diagonal)	2,0 (5,1 cm diagonal)	2,0 (5,1 cm diagonal)	-----

RESOLUCIÓN PANTALLA	-----	320 x 240 píxeles	320 x 240 píxeles	320 x 240 píxeles	-----
TIPO PANTALLA	-----	LCD TFT a color QVGA	LCD TFT a color QVGA	LCD TFT a color QVGA	-----
SENSORES	-G-Sensor	-G-Sensor -GPS -Galileo	-G-Sensor -GPS -Galileo	- G-Sensor -GPS -Galileo	-GPS -Galileo
ALMACENAMIENTO	-microSDHC 8Gb - 512Gb, clase 10 o superior (no incluido) -Nube Vault durante 24h sin límite de tamaño	-microSDHC hasta 512Gb, clase 10 o superior (no incluido) -Nube Vault durante 24h sin límite de tamaño	-microSDHC hasta 512Gb, clase 10 o superior (no incluido) -Nube Vault durante 24h sin límite de tamaño	-microSDHC hasta 512 Gb, clase 10 o superior (no incluido) -Nube Vault durante 24h sin límite de tamaño	-microSD 16 Gb (incluida). Compatible con microSD mínimo 8 Gb clase 10 o superior
FUNCIONAMIENTO	-Al conectarla a la fuente de alimentación del vehículo graba continuamente -Registra y guarda vídeos automáticamente cuando hay una incidencia (G-Sensor)	-Al conectarla a la fuente de alimentación del vehículo graba continuamente -Registra y guarda vídeos automáticamente cuando hay una incidencia (G-Sensor) -Permite añadir ubicación (GPS)	-Al conectarla a la fuente de alimentación del vehículo graba continuamente -Registra y guarda vídeos automáticamente cuando hay una incidencia (G-Sensor) -Permite añadir ubicación (GPS)	-Al conectarla a la fuente de alimentación del vehículo graba continuamente -Registra y guarda vídeos automáticamente cuando hay una incidencia (G-Sensor) -Permite añadir ubicación (GPS)	-La cámara guarda automáticamente el vídeo cuando detecta una incidencia. -Dota de pruebas exactas del momento y el lugar en el que ocurre
EXTRAS	-conexión WiFi y Bluetooth -App Garmin Drive	-conexión WiFi y Bluetooth -App Garmin Drive	- conexión WiFi y Bluetooth -App Garmin Drive	-conexión WiFi y Bluetooth - App Garmin Drive	-conexión WiFi -App Garmin Drive

	<ul style="list-style-type: none"> -Grabación coche aparcado (cable extra comprar) -Cargador USB dual -Control por voz -Compatible con Vault (nube) (pago extra para mejorar duración del almacenamiento) 	<ul style="list-style-type: none"> -Grabación coche aparcado (cable extra comprar) -Cargador USB dual -Control por voz -Compatible con Vault (nube) (pago extra para mejorar duración del almacenamiento) -Recibe alertas de conducción -Parking Guard (requiere WiFi) 	<ul style="list-style-type: none"> -Grabación coche aparcado (cable extra comprar) -Cargador USB dual -Control por voz -Compatible con Vault (nube) (pago extra para mejorar duración del almacenamiento) -Recibe alertas de conducción -Parking Guard (requiere WiFi) 	<ul style="list-style-type: none"> - Grabación coche aparcado (cable extra comprar) - Cargador USB dual - Control por voz - Compatible con Vault (nube) (pago extra para mejorar duración del almacenamiento) -Recibe alertas de conducción -Parking Guard (requiere WiFi) 	<ul style="list-style-type: none"> -Grabación coche aparcado (cable extra comprar) -Cargador USB dual -Control por voz - Compatible con Vault (nube) (pago extra para mejorar duración del almacenamiento) -Parking Guard (requiere WiFi) -NightGlo: Visión nocturna interior y exterior
ENLACE	<u>Garmin Dash Cam Mini 2</u>	<u>Garmin Dash Cam 47</u>	<u>Garmin Dash Cam 57</u>	<u>Garmin Dash Cam 67W</u>	<u>Garmin Dash Cam Tandem</u>

TABLA COMPARATIVA MODELOS MARCA VANTRUE					
	E1 MINI WIFI Dashcam	S1 Dual Dashcam	X4S Dual WIFI Dashcam	S2 Dual WIFI Dashcam	S2 3 Lens Dashcam
PRECIO (€)	149,99	199,99	219,99	229,99	259,99
TAMAÑO (cm) ancho-largo	12,2 x 13,0	-----	12,2 x 13,0	-----	-----
PESO (gr)	70	-----	80	-----	-----
SOPORTE	Soporte magnético	Montaje adhesivo	Montaje de succion	Montaje adhesivo	Montaje adhesivo
ALIMENTACIÓN	VANTRUE Hardwire Kit (desde alimentación del vehiculo)	Mini USB	VANTRUE Hardwire Kit (desde alimentación del vehiculo), USB-C	USB-C	USB-C
RESOLUCIÓN CÁMARA	1944p HD	2160p HD (Frontal) 1080p (Trasera)	2160p HD	1080p, 1440p 960p 2K HD	1440p (Exterior) 1080p (Interior)
VELOCIDAD FOTOGRAMAS	Hasta 30 FPS	Hasta 60 FPS	Hasta 120 FPS	Hasta 30 FPS	Hasta 30 FPS
ÁNGULO DE VISIÓN (°)	160 (diagonal)	170 (Frontal) 160 (Trasera)	155 (diagonal)	160 (exterior) 165 (interior)	160 (exterior) 165 (interior)
ORIENTACIÓN	Grabación exterior	Grabación exterior	Grabación exterior	Grabación exterior Grabación interior	Grabación exterior Grabación interior
PANTALLA INTEGRADA	Si	Si	Si	Si	Si
TAMAÑO PANTALLA (Pulgadas)	1.54	2	3	3	2.45
RESOLUCIÓN PANTALLA	-----	-----	-----	-----	-----
TIPO PANTALLA	LCD IPS	LCD	LCD	LCD (táctil)	LCD
SENSORES	-G-Sensor	-G-Sensor	-G-Sensor	-G-Sensor	-G-Sensor

ALMACENAMIENTO	-Capacidad de tarjeta SD de hasta 512 GB	-Capacidad de tarjeta SD de hasta 256 GB	-Capacidad de tarjeta SD de hasta 512 GB	-Capacidad de tarjeta SD de hasta 512 GB	-Capacidad de tarjeta SD de hasta 512 GB
FUNCIONAMIENTO	-Registra y guarda vídeos automáticamente cuando hay una incidencia (G-Sensor)	-Grabación continua. -Registra y guarda vídeos automáticamente cuando hay una incidencia (G-Sensor)	-Configuración inicial desde menú. -Grabación continua. -Registra y guarda vídeos automáticamente cuando hay una incidencia (G-Sensor) -Opción de vigilancia en 24 estacionamiento.	-Configuración inicial desde menú. - Detección de colisiones: las cámaras delantera y de cabina pueden detectar colisiones. -Grabación continua. -Registra y guarda vídeos automáticamente cuando hay una incidencia (G-Sensor)	-Grabación continua. -Registra y guarda vídeos automáticamente cuando hay una incidencia (G-Sensor)
EXTRAS	-Conexión WiFi. -App dash-cam -Posicionamiento GPS dual preciso. -Función de lapso de tiempo que toma fotos automáticamente a intervalos específicos.	-Posicionamiento GPS -Modo de asistencia en el estacionamiento. -Control de colisión 24h.	-Conexión WiFi. -App dash-cam -Posicionamiento GPS dual preciso. -Función de lapso de tiempo que toma fotos automáticamente a intervalos específicos.	-Conexión WiFi. -App dash-cam -Posicionamiento GPS dual preciso. -Micrófono. -Opción de cámara extra trasera de grabación simultánea. - Detección de	-Posicionamiento GPS (opcional). - Detección de movimiento y colisión 24h. -Opción adicional de grabación trasera.

	-Control por voz con 9 comandos distintos.		-Micrófono y altavoz integrado. -Opción de cámara extra trasera de grabación simultánea.	movimiento frente la cabina.	
ENLACE	<u>E1 MINI WIFI Dashcam</u>	<u>S1 Dual Dashcam</u>	<u>X4S Dual WIFI Dashcam</u>	<u>S2 Dual WIFI Dashcam</u>	<u>S2 3 Lens Dashcam</u>

TABLA COMPARATIVA MODELOS MARCA VIOFO				
	A119 V3	A129 Plus DUO	T130	A139 Pro Duo
PRECIO (€)	119,99	179,99	239,99	269,99
TAMAÑO (cm) ancho-largo-profundidad	2 x 2 x 3	8,2 x 5 x 4,1 (Frontal) 4,9 x 5,2 x 2,6 (Trasera)	-----	-----
PESO (gr)	-----	-----	-----	-----
SOPORTE	Montaje adhesivo	Montaje adhesivo	Montaje adhesivo	Montaje adhesivo
ALIMENTACIÓN	Mini USB	USB	USB-C	USB-C
RESOLUCIÓN CÁMARA	1080p, 1440p, 1600p HD	2160p HD (Frontal) 1080p (Trasera)	2160p HD (Frontal) 1080p (Trasera)	1440p (Frontal) 1080p (Interior) 1080p (Trasera)
VELOCIDAD FOTOGRAMAS	Hasta 60 FPS	Hasta 30 FPS	Hasta 30 FPS	Hasta 30 FPS
ÁNGULO DE VISIÓN (°)	140 (diagonal)	130 (Frontal) 140 (Trasera)	140 (Frontal) 165 (Trasera) 165 (Interior)	140 (Frontal) 170 (Trasera) 170 (Interior)
ORIENTACIÓN	Grabación exterior	Grabación exterior	Grabación exterior Grabación interior	Grabación exterior Grabación interior
PANTALLA INTEGRADA	Si	Si	No	No
TAMAÑO PANTALLA (Pulgadas)	2	2	-----	-----
RESOLUCIÓN PANTALLA	-----	-----	-----	-----
TIPO PANTALLA	LCD IPS	LCD	-----	-----
SENSORES	-G-Sensor	-G-Sensor	-G-Sensor	-G-Sensor
ALMACENAMIENTO	-Capacidad de tarjeta microSD de hasta 256 GB	-Capacidad de tarjeta microSD de hasta 256 GB	-Capacidad de tarjeta microSD de hasta 256 GB	-Capacidad de tarjeta microSD de hasta 256 GB

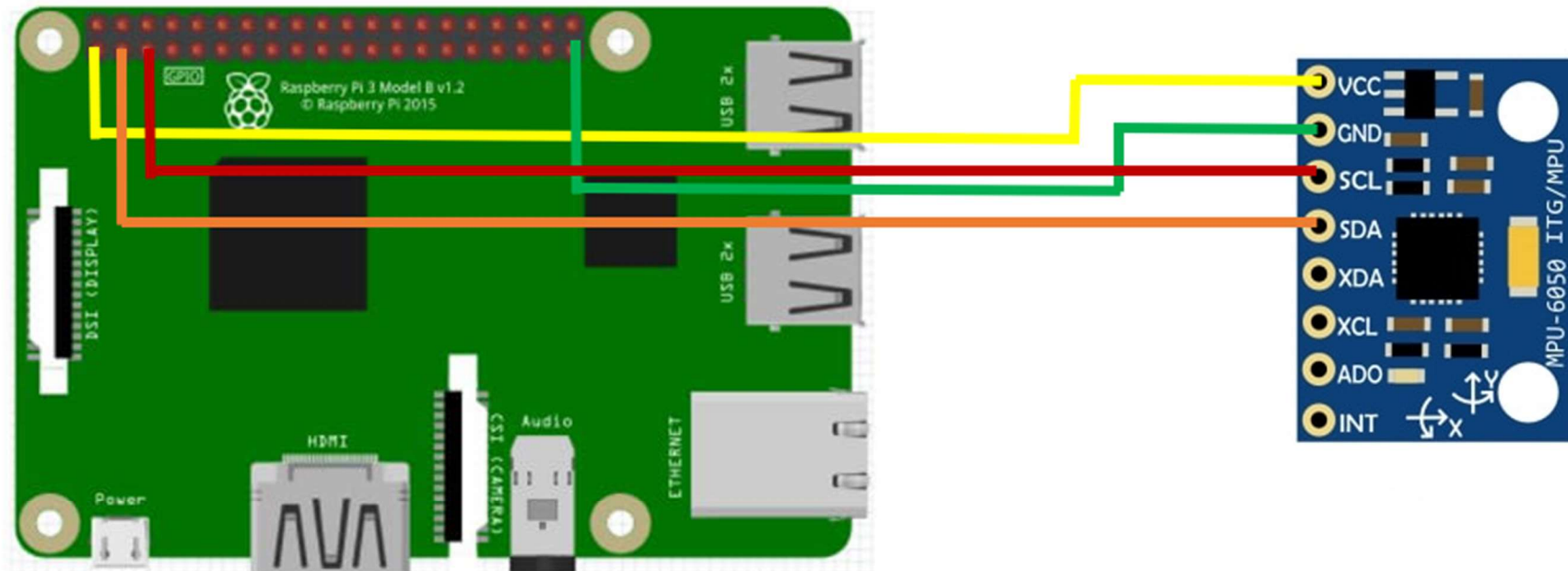
FUNCIONAMIENTO	-Configuración inicial desde menú.	-Configuración inicial desde menú.	-Configuración inicial desde aplicación (VIOFO APP).	-Configuración inicial desde aplicación (VIOFO APP).
EXTRAS	-Monitor de aparcamiento con detección de movimiento configurable.	-Posicionamiento GPS. -Conexión Bluetooth y WiFi. -Modo de asistencia en el estacionamiento. -Control de colisión 24h.	-Posicionamiento GPS. -Conexión WiFi. -Modo de asistencia en el estacionamiento. -Control de colisión 24h. -Notificación por voz. -Grabación en bucle.	-Posicionamiento GPS. -Conexión Bluetooth y WiFi. -Modo de asistencia en el estacionamiento. -Control de colisión 24h. -Notificación por voz. -Grabación en bucle.
ENLACE	A119 V3	A129 Plus DUO	T130	A139 Pro Duo

ANEXO II

ANEXO II.

ESQUEMÁTICO DEL CONEXIONADO ELÉCTRICO DEL MODELO.

CONEXIONADO DE RASPBERRY PI – GY-521



CONEXIONADO DE ARDUINO MEGA – GY-521

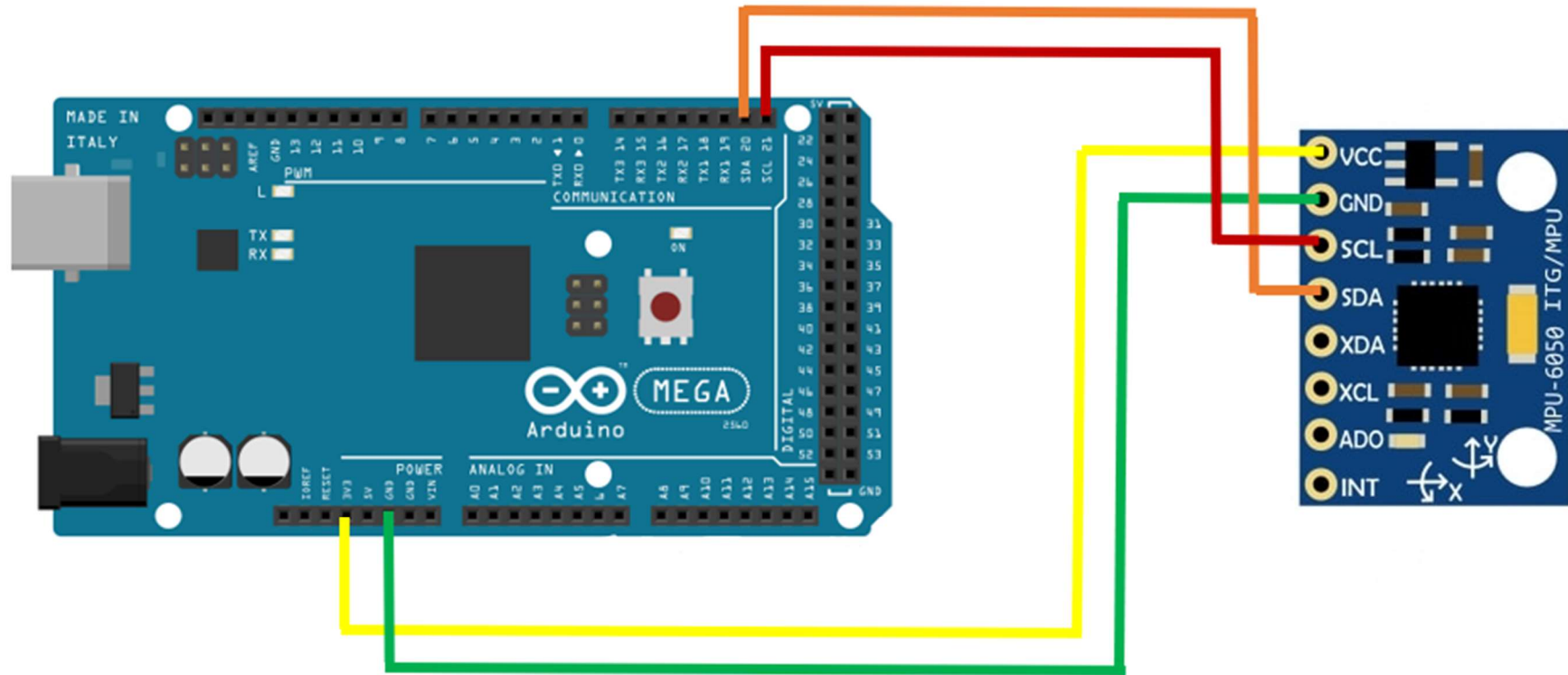


Imagen Raspberry Pi 3 Model B obtenida de [69]

Imagen Arduino Mega obtenida de [70]

Imagen GY-521 (MPU6050) obtenida de [71]

ANEXO III

ANEXO III.

CÓDIGO DE PROGRAMACIÓN DE LAS FUNCIONES BÁSICAS DEL SISTEMA PROPUESTO.

BUCLE DE GRABACIÓN CONDICIONADO

```
import io
import os
import sys
import threading
import picamera
import time
import pandas as pd
from subprocess import call

class HiloSensor(threading.Thread):
    def run(self):
        print("\nSe inicializa el hilo detector de impacto. \n")
        aceleracionFicheroX =
pd.read_excel('/home/pi/Descargas/aceleracionx.xlsx')
        aceleracionFicheroY =
pd.read_excel('/home/pi/Descargas/aceleraciony.xlsx')
        aceleracionFicheroZ =
pd.read_excel('/home/pi/Descargas/aceleracionz.xlsx')

        diferenciaAceleracion = pd.DataFrame({
            'aceleracionx' :
abs((aceleracionFicheroX['Aceleracion'])),
            'aceleraciony' :
abs((aceleracionFicheroY['Aceleracion'])),
            'aceleracionz' :
abs((aceleracionFicheroZ['Aceleracion'])),
        })

        vectorx = []
        vectory = []
        vectorz = []

        valorChoque = 8
        comparacionModulo = []

        print("Se han leído los ficheros. \n")
        print("Comienza el retardo \n")
        time.sleep(30)
        print("Finaliza el retardo \n")
```

```
for i in aceleracionFicheroX.index[0:]:
    if i < 1:

vectorx.append(float(diferenciaAceleracion['aceleracionx'][i]))

vectory.append(float(diferenciaAceleracion['aceleraciony'][i]))

vectorz.append(float(diferenciaAceleracion['aceleracionz'][i]))
    i += 1
    else:
        restax = abs(vectorx[i-2]) - abs(vectorx[i-1])
        restay = abs(vectory[i-2]) - abs(vectory[i-1])
        restaz = abs(vectorz[i-2]) - abs(vectorz[i-1])

vectorx.append(float(diferenciaAceleracion['aceleracionx'][i]))

vectory.append(float(diferenciaAceleracion['aceleraciony'][i]))

vectorz.append(float(diferenciaAceleracion['aceleracionz'][i]))
        modulo = (restax**2 + restay**2 + restaz**2)**0.5
        comparacionModulo.append(float(modulo))
        i += 1

        if modulo > valorChoque:
            print("Se detectó impacto. Se cierra hilo
detector.\n")
            sys.exit()
```

```
hilo1 = HiloSensor()
hilo1.start()
choque = hilo1.is_alive()
i = 1
print("Se comienza a grabar. \n")

while choque:
    with picamera.PiCamera() as camera:
        camera.resolution = (640, 480)
        camera.start_recording('Video%d.h264' % i)
        camera.wait_recording(5)
        camera.stop_recording
        i += 1
        choque = hilo1.is_alive()
    if i >= 4:
        os.remove('Video%d.h264' % (i - 3))

with picamera.PiCamera() as camera:
    camera.resolution = (640, 480)
```

```
camera.start_recording('VideoFinal.h264')
camera.wait_recording(5)
camera.stop_recording()
print("Convirtiendo vídeos a formato .mp4. \n")
convertidor1 = "MP4Box -add " + "Video" + str(i - 2) + ".h264"
+ " Video1.mp4"
convertidor2 = "MP4Box -add " + "Video" + str(i - 1) + ".h264"
+ " Video2.mp4"
convertidor3 = "MP4Box -add VideoFinal.h264 Video3.mp4"
call([convertidor1], shell=True)
call([convertidor2], shell=True)
call([convertidor3], shell=True)
print("\n Borrando ficheros residuos de formato .h264 \n ")
borrar1 = "rm " + "Video" + str(i - 2) + ".h264"
borrar2 = "rm " + "Video" + str(i - 1) + ".h264"
borrar3 = "rm " + "VideoFinal.h264"
call([borrar1], shell=True)
call([borrar2], shell=True)
call([borrar3], shell=True)

from moviepy.editor import *

clip1 = VideoFileClip("Video1.mp4")
clip2 = VideoFileClip("Video2.mp4")
clip3 = VideoFileClip("Video3.mp4")
final_clip = concatenate_videoclips([clip1,clip2,clip3])
print("\n Obteniendo el vídeo Final. \n")
final_clip.write_videofile("Resultado.mp4")
borrar4 = "rm " + "Video1.mp4"
borrar5 = "rm " + "Video2.mp4"
borrar6 = "rm " + "Video3.mp4"
call([borrar4], shell=True)
call([borrar5], shell=True)
call([borrar6], shell=True)
```


LECTURA DEL SENSOR

```
#include "I2Cdev.h"
#include "MPU6050.h"
#include "Wire.h"

MPU6050 sensor;
int x, y, z;

void setup() {
  Serial.begin(9600);
  Wire.begin();
  sensor.initialize();
}

void loop() {
  sensor.getAcceleration(&x, &y, &z);
  float xReal = x * 2 * (9.81/32768.0);
  float yReal = y * 2 * (9.81/32768.0);
  float zReal = z * 2 * (9.81/32768.0);
  Serial.println(xReal);
  Serial.println(yReal);
  Serial.println(zReal);
}
```

CREACIÓN DE FICHEROS

```
import serial
import time

aceleracionx =
open('C:\\Users\\gibag\\Desktop\\Mitxt\\aceleracionx.txt', 'w')
aceleraciony =
open('C:\\Users\\gibag\\Desktop\\Mitxt\\aceleraciony.txt', 'w')
aceleracionz =
open('C:\\Users\\gibag\\Desktop\\Mitxt\\aceleracionz.txt', 'w')
serialArduino = serial.Serial("COM3", 9600)
time.sleep(0.1)

i = 0
j = 1
for i in range(0, 200):
    if j == 1:
        cad = serialArduino.readline().decode('ascii')
        print(cad)
        aceleracionz.write(cad)
        j = 2
        i += 1
    if j == 2:
        cad = serialArduino.readline().decode('ascii')
        print(cad)
        aceleracionx.write(cad)
        j = 3
        i += 1
    if j == 3:
        cad = serialArduino.readline().decode('ascii')
        print(cad)
        aceleraciony.write(cad)
        j = 1
        i += 1
```

ANEXO IV

ANEXO IV.

CÓDIGO EMPLEADO PARA LA CALIBRACIÓN DEL MPU6050.

```
// Arduino sketch that returns calibration offsets for MPU6050
// Version 1.1 (31th January 2014)
// Version 1.2 (25th August 2019)
// Done by Luis Ródenas <luisrodenaslorda@gmail.com> and improved
// by Shakeel <blinkmaker.com>
// Based on the I2Cdev library and previous work by Jeff Rowberg
// <jeff@rowberg.net>
// Updates (of the library) should (hopefully) always be available
// at https://github.com/jrowberg/i2cdevlib
```

```
// These offsets were meant to calibrate MPU6050's internal DMP,
// but can be also useful for reading sensors.
// The effect of temperature has not been taken into account so I
// can't promise that it will work if you
// calibrate indoors and then use it outdoors. Best is to calibrate
// and use at the same room temperature.
```

```
/* ===== LICENSE =====
I2Cdev device library code is placed under the MIT license
Copyright (c) 2011 Jeff Rowberg
```

```
Permission is hereby granted, free of charge, to any person
obtaining a copy
of this software and associated documentation files (the
"Software"), to deal
in the Software without restriction, including without limitation
the rights
to use, copy, modify, merge, publish, distribute, sublicense,
and/or sell
copies of the Software, and to permit persons to whom the Software
is
furnished to do so, subject to the following conditions:
```

```
The above copyright notice and this permission notice shall be
included in
all copies or substantial portions of the Software.
```

```
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
SHALL THE
```

AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN

THE SOFTWARE.

=====

*/

// I2Cdev and MPU6050 must be installed as libraries

#include "I2Cdev.h"

#include "MPU6050.h"

#include "Wire.h"

////////////////////////////////////// CONFIGURATION
//////////////////////////////////////

//Change this 3 variables if you want to fine tune the sketch to your needs.

int buffersize=1000; //Amount of readings used to average, make it higher to get more precision but sketch will be slower (default:1000)

int acel_deadzone=8; //Acelerometer error allowed, make it lower to get more precision, but sketch may not converge (default:8)

int gyro_deadzone=1; //Gyro error allowed, make it lower to get more precision, but sketch may not converge (default:1)

// default I2C address is 0x68

// specific I2C addresses may be passed as a parameter here

// AD0 low = 0x68 (default for InvenSense evaluation board)

// AD0 high = 0x69

//MPU6050 accelgyro;

MPU6050 accelgyro(0x68); // default <-- use for AD0 high

int16_t ax, ay, az, gx, gy, gz;

int mean_ax, mean_ay, mean_az, mean_gx, mean_gy, mean_gz, state=0;

int ax_offset, ay_offset, az_offset, gx_offset, gy_offset, gz_offset;

int mpuConnection;

////////////////////////////////////// SETUP
//////////////////////////////////////

void setup() {

 // join I2C bus (I2Cdev library doesn't do this automatically)

 Wire.begin();

 // COMMENT NEXT LINE IF YOU ARE USING ARDUINO DUE

```
TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz). Leonardo
measured 250kHz.

// initialize serial communication
Serial.begin(9600);

// initialize MPU-6050
accelgyro.initialize();

// wait for ready
while (Serial.available() && Serial.read()); // empty buffer

Serial.println("Place the MPU-6050 breakout board in a flat or
horizontal position, with SMD components facing up.\n");
Serial.println(F("Type in any character and press Enter/Send to
start MPU-6050 calibration..."));

while (Serial.available() == 0){ } //wait for character to be
entered
while (Serial.available() && Serial.read()); // empty buffer
again

// start message
Serial.println("\n-----
-----");
Serial.println("\nStarting MPU-6050 Calibration Sketch.");
delay(1000);
Serial.println("\nDon't touch the MPU-6050 until you see a
\"FINISHED!\" message.");
delay(2000);

// verify connection
//Serial.println(accelgyro.testConnection() ? "MPU-6050
connection SUCCESSFUL!" : "MPU-6050 connection FAILED.");
//delay(1000);
Serial.println("\n-----
-----");
Serial.println("\nVerifying MPU-6050 connection...");
delay(500);

mpuConnection = accelgyro.testConnection();

if (mpuConnection == 0) {
    Serial.println("\nMPU-6050 connection FAILED.");
    Serial.println("\nCheck your boards and connections. Reset
the Arduino board to run the calibration sketch again.");
}
```

```
while (mpuConnection == 0) { } //wait for MPU-6050 connection
to be established.

Serial.println("\nMPU-6050 connection SUCCESSFUL!");
delay(500);
Serial.println("\n-----
-----");
Serial.println("\nResetting MPU-6050 offsets.");

// reset offsets
accelgyro.setXAccelOffset(0);
accelgyro.setYAccelOffset(0);
accelgyro.setZAccelOffset(0);
accelgyro.setXGyroOffset(0);
accelgyro.setYGyroOffset(0);
accelgyro.setZGyroOffset(0);

delay(500);
}

//////////////////////////////////// LOOP
////////////////////////////////////
void loop() {
  if (state==0){
    Serial.println("\nReading accelerometer and gyroscope sensors
for first time.");
    meansensors();
    state++;
    delay(1000);
  }

  if (state==1) {
    Serial.println("\nCalculating offsets");
    calibration();
    state++;
    delay(1000);
  }

  if (state==2) {
    meansensors();
    Serial.println("\nFINISHED!");

Serial.println("\n=====
=====");
    Serial.println("\nRESULTS:");
    Serial.println("\nSensor data is listed in the
format:\taccelX\taccelY\taccelZ\tgyroX\tgyroY\tgyroZ");
    Serial.print("\nSensor readings INCLUDING offsets:\t");
    Serial.print(mean_ax);
```

```
Serial.print("\t");
Serial.print(mean_ay);
Serial.print("\t");
Serial.print(mean_az);
Serial.print("\t");
Serial.print(mean_gx);
Serial.print("\t");
Serial.print(mean_gy);
Serial.print("\t");
Serial.println(mean_gz);

Serial.println("\nCompare with IDEAL sensor
readings:\t0\t0\t16384\t0\t0\t0");
Serial.println("\n-----
-----");

Serial.println("\nYour MPU-6050 offsets:\n");
Serial.print("mpu.setXAccelOffset(");
Serial.print(ax_offset);
Serial.println(");");
Serial.print("mpu.setYAccelOffset(");
Serial.print(ay_offset);
Serial.println(");");
Serial.print("mpu.setZAccelOffset(");
Serial.print(az_offset);
Serial.println(");");
Serial.print("mpu.setXGyroOffset(");
Serial.print(gx_offset);
Serial.println(");");
Serial.print("mpu.setYGyroOffset(");
Serial.print(gy_offset);
Serial.println(");");
Serial.print("mpu.setZGyroOffset(");
Serial.print(gz_offset);
Serial.println(");");

Serial.println("\nYou can copy and paste the above offsets
directly into your sketch. :)");

while (1);
}
}

////////////////////////////////////// FUNCTIONS
//////////////////////////////////////
void meansensors(){
long
i=0,buff_ax=0,buff_ay=0,buff_az=0,buff_gx=0,buff_gy=0,buff_gz=0;
```



```
while (i<(buffersize+101)){
    // read raw accel/gyro measurements from device
    accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

    if (i>100 && i<=(buffersize+100)){ //First 100 measures are
discarded
        buff_ax=buff_ax+ax;
        buff_ay=buff_ay+ay;
        buff_az=buff_az+az;
        buff_gx=buff_gx+gx;
        buff_gy=buff_gy+gy;
        buff_gz=buff_gz+gz;
    }
    if (i==(buffersize+100)){
        mean_ax=buff_ax/buffersize;
        mean_ay=buff_ay/buffersize;
        mean_az=buff_az/buffersize;
        mean_gx=buff_gx/buffersize;
        mean_gy=buff_gy/buffersize;
        mean_gz=buff_gz/buffersize;
    }
    i++;
    delay(2); //Needed so we don't get repeated measures
}
}

void calibration(){
    ax_offset=-mean_ax/8;
    ay_offset=-mean_ay/8;
    az_offset=(16384-mean_az)/8;

    gx_offset=-mean_gx/4;
    gy_offset=-mean_gy/4;
    gz_offset=-mean_gz/4;
    while (1){
        int ready=0;
        accelgyro.setXAccelOffset(ax_offset);
        accelgyro.setYAccelOffset(ay_offset);
        accelgyro.setZAccelOffset(az_offset);

        accelgyro.setXGyroOffset(gx_offset);
        accelgyro.setYGyroOffset(gy_offset);
        accelgyro.setZGyroOffset(gz_offset);

        meansensors();
        Serial.println("...");

        if (abs(mean_ax)<=acel_deadzone) ready++;
        else ax_offset=ax_offset-mean_ax/acel_deadzone;
```

```
    if (abs(mean_ay)<=acel_deadzone) ready++;  
    else ay_offset=ay_offset-mean_ay/acel_deadzone;  
  
    if (abs(16384-mean_az)<=acel_deadzone) ready++;  
    else az_offset=az_offset+(16384-mean_az)/acel_deadzone;  
  
    if (abs(mean_gx)<=gyro_deadzone) ready++;  
    else gx_offset=gx_offset-mean_gx/(gyro_deadzone+1);  
  
    if (abs(mean_gy)<=gyro_deadzone) ready++;  
    else gy_offset=gy_offset-mean_gy/(gyro_deadzone+1);  
  
    if (abs(mean_gz)<=gyro_deadzone) ready++;  
    else gz_offset=gz_offset-mean_gz/(gyro_deadzone+1);  
  
    if (ready==6) break;  
  }  
}
```

ANEXO V

ANEXO V.

CÓDIGO EMPLEADO PARA JUSTIFICACIÓN DE ERROR COMETIDO CON LA PÉRDIDA DE FOTOGRAMAS.

```
import io
import picamera
import time

errorAcumulado = 0

for i in range(0,20):
    with picamera.PiCamera() as camera:
        camera.resolution = (640, 480)
        camera.start_recording('Video1.h264')
        camera.wait_recording(1)
        camera.stop_recording()
        inicial = time.time()
        camera.resolution = (640, 480)
        final = time.time()
        camera.start_recording('Video1.h264')
        camera.wait_recording(1)
        camera.stop_recording()
        error = final - inicial
        errorAcumulado = errorAcumulado + error
        print(error)
        i += 1

print("El valor de error promedio para las muestras realizadas
es de: \n")
print(errorAcumulado/(i + 1))
```

ANEXO VI

ANEXO VI.

DATASHEET DE MPU-6050.

	MPU-6000/MPU-6050 Product Specification	Document Number: PS-MPU-6000A-00 Revision: 3.4 Release Date: 08/19/2013
---	--	---

3 Product Overview

3.1 MPU-60X0 Overview

MotionInterface™ is becoming a “must-have” function being adopted by smartphone and tablet manufacturers due to the enormous value it adds to the end user experience. In smartphones, it finds use in applications such as gesture commands for applications and phone control, enhanced gaming, augmented reality, panoramic photo capture and viewing, and pedestrian and vehicle navigation. With its ability to precisely and accurately track user motions, MotionTracking technology can convert handsets and tablets into powerful 3D intelligent devices that can be used in applications ranging from health and fitness monitoring to location-based services. Key requirements for MotionInterface enabled devices are small package size, low power consumption, high accuracy and repeatability, high shock tolerance, and application specific performance programmability – all at a low consumer price point.

The MPU-60X0 is the world’s first integrated 6-axis MotionTracking device that combines a 3-axis gyroscope, 3-axis accelerometer, and a Digital Motion Processor™ (DMP) all in a small 4x4x0.9mm package. With its dedicated I²C sensor bus, it directly accepts inputs from an external 3-axis compass to provide a complete 9-axis MotionFusion™ output. The MPU-60X0 MotionTracking device, with its 6-axis integration, on-board MotionFusion™, and run-time calibration firmware, enables manufacturers to eliminate the costly and complex selection, qualification, and system level integration of discrete devices, guaranteeing optimal motion performance for consumers. The MPU-60X0 is also designed to interface with multiple non-inertial digital sensors, such as pressure sensors, on its auxiliary I²C port. The MPU-60X0 is footprint compatible with the MPU-30X0 family.

The MPU-60X0 features three 16-bit analog-to-digital converters (ADCs) for digitizing the gyroscope outputs and three 16-bit ADCs for digitizing the accelerometer outputs. For precision tracking of both fast and slow motions, the parts feature a user-programmable gyroscope full-scale range of ± 250 , ± 500 , ± 1000 , and $\pm 2000^\circ/\text{sec}$ (dps) and a user-programmable accelerometer full-scale range of $\pm 2g$, $\pm 4g$, $\pm 8g$, and $\pm 16g$.

An on-chip 1024 Byte FIFO buffer helps lower system power consumption by allowing the system processor to read the sensor data in bursts and then enter a low-power mode as the MPU collects more data. With all the necessary on-chip processing and sensor components required to support many motion-based use cases, the MPU-60X0 uniquely enables low-power MotionInterface applications in portable applications with reduced processing requirements for the system processor. By providing an integrated MotionFusion output, the DMP in the MPU-60X0 offloads the intensive MotionProcessing computation requirements from the system processor, minimizing the need for frequent polling of the motion sensor output.

Communication with all registers of the device is performed using either I²C at 400kHz or SPI at 1MHz (MPU-6000 only). For applications requiring faster communications, the sensor and interrupt registers may be read using SPI at 20MHz (MPU-6000 only). Additional features include an embedded temperature sensor and an on-chip oscillator with $\pm 1\%$ variation over the operating temperature range.

By leveraging its patented and volume-proven Nasiri-Fabrication platform, which integrates MEMS wafers with companion CMOS electronics through wafer-level bonding, InvenSense has driven the MPU-60X0 package size down to a revolutionary footprint of 4x4x0.9mm (QFN), while providing the highest performance, lowest noise, and the lowest cost semiconductor packaging required for handheld consumer electronic devices. The part features a robust 10,000g shock tolerance, and has programmable low-pass filters for the gyroscopes, accelerometers, and the on-chip temperature sensor.


For power supply flexibility, the MPU-60X0 operates from VDD power supply voltage range of 2.375V-3.46V. Additionally, the MPU-6050 provides a VLOGIC reference pin (in addition to its analog supply pin: VDD), which sets the logic levels of its I²C interface. The VLOGIC voltage may be 1.8V $\pm 5\%$ or VDD.

The MPU-6000 and MPU-6050 are identical, except that the MPU-6050 supports the I²C serial interface only, and has a separate VLOGIC reference pin. The MPU-6000 supports both I²C and SPI interfaces and has a single supply pin, VDD, which is both the device’s logic reference supply and the analog supply for the part. The table below outlines these differences:

	MPU-6000/MPU-6050 Product Specification	Document Number: PS-MPU-6000A-00 Revision: 3.4 Release Date: 08/19/2013
---	--	---

Primary Differences between MPU-6000 and MPU-6050

Part / Item	MPU-6000	MPU-6050
VDD	2.375V-3.46V	2,375V-3,46V
VLOGIC	n/a	1.71V to VDD
Serial Interfaces Supported	I ² C, SPI	I ² C
Pin 8	/CS	VLOGIC
Pin 9	AD0/SDO	AD0
Pin 23	SCL/SCLK	SCL
Pin 24	SDA/SDI	SDA

	MPU-6000/MPU-6050 Product Specification	Document Number: PS-MPU-6000A-00 Revision: 3.4 Release Date: 08/19/2013
---	--	---

4 Applications

- *BlurFree™* technology (for Video/Still Image Stabilization)
- *AirSign™* technology (for Security/Authentication)
- *TouchAnywhere™* technology (for "no touch" UI Application Control/Navigation)
- *MotionCommand™* technology (for Gesture Short-cuts)
- Motion-enabled game and application framework
- InstantGesture™ iG™ gesture recognition
- Location based services, points of interest, and dead reckoning
- Handset and portable gaming
- Motion-based game controllers
- 3D remote controls for Internet connected DTVs and set top boxes, 3D mice
- Wearable sensors for health, fitness and sports
- Toys

	MPU-6000/MPU-6050 Product Specification	Document Number: PS-MPU-6000A-00 Revision: 3,4 Release Date: 08/19/2013
---	--	---

5 Features

5.1 Gyroscope Features

The triple-axis MEMS gyroscope in the MPU-60X0 includes a wide range of features:

- Digital-output X-, Y-, and Z-Axis angular rate sensors (gyroscopes) with a user-programmable full-scale range of ± 250 , ± 500 , ± 1000 , and $\pm 2000^\circ/\text{sec}$
- External sync signal connected to the FSYNC pin supports image, video and GPS synchronization
- Integrated 16-bit ADCs enable simultaneous sampling of gyros
- Enhanced bias and sensitivity temperature stability reduces the need for user calibration
- Improved low-frequency noise performance
- Digitally-programmable low-pass filter
- Gyroscope operating current: 3.6mA
- Standby current: 5 μ A
- Factory calibrated sensitivity scale factor
- User self-test

5.2 Accelerometer Features

The triple-axis MEMS accelerometer in MPU-60X0 includes a wide range of features:

- Digital-output triple-axis accelerometer with a programmable full scale range of $\pm 2g$, $\pm 4g$, $\pm 8g$ and $\pm 16g$
- Integrated 16-bit ADCs enable simultaneous sampling of accelerometers while requiring no external multiplexer
- Accelerometer normal operating current: 500 μ A
- Low power accelerometer mode current: 10 μ A at 1.25Hz, 20 μ A at 5Hz, 60 μ A at 20Hz, 110 μ A at 40Hz
- Orientation detection and signaling
- Tap detection
- User-programmable interrupts
- High-G interrupt
- User self-test

5.3 Additional Features

The MPU-60X0 includes the following additional features:

- 9-Axis MotionFusion by the on-chip Digital Motion Processor (DMP)
- Auxiliary master I²C bus for reading data from external sensors (e.g., magnetometer)
- 3.9mA operating current when all 6 motion sensing axes and the DMP are enabled
- VDD supply voltage range of 2.375V-3.46V
- Flexible VLOGIC reference voltage supports multiple I²C interface voltages (MPU-6050 only)
- Smallest and thinnest QFN package for portable devices: 4x4x0,9mm
- Minimal cross-axis sensitivity between the accelerometer and gyroscope axes
- 1024 byte FIFO buffer reduces power consumption by allowing host processor to read the data in bursts and then go into a low-power mode as the MPU collects more data
- Digital-output temperature sensor
- User-programmable digital filters for gyroscope, accelerometer, and temp sensor
- 10,000 g shock tolerant
- 400kHz Fast Mode I²C for communicating with all registers
- 1MHz SPI serial interface for communicating with all registers (MPU-6000 only)
- 20MHz SPI serial interface for reading sensor and interrupt registers (MPU-6000 only)

	MPU-6000/MPU-6050 Product Specification	Document Number: PS-MPU-6000A-00 Revision: 3.4 Release Date: 08/19/2013
---	--	---


- MEMS structure hermetically sealed and bonded at wafer level
- RoHS and Green compliant

5.4 MotionProcessing

- Internal Digital Motion Processing™ (DMP™) engine supports 3D MotionProcessing and gesture recognition algorithms
- The MPU-60X0 collects gyroscope and accelerometer data while synchronizing data sampling at a user defined rate. The total dataset obtained by the MPU-60X0 includes 3-Axis gyroscope data, 3-Axis accelerometer data, and temperature data. The MPU's calculated output to the system processor can also include heading data from a digital 3-axis third party magnetometer.
- The FIFO buffers the complete data set, reducing timing requirements on the system processor by allowing the processor burst read the FIFO data. After burst reading the FIFO data, the system processor can save power by entering a low-power sleep mode while the MPU collects more data.
- Programmable interrupt supports features such as gesture recognition, panning, zooming, scrolling, tap detection, and shake detection
- Digitally-programmable low-pass filters
- Low-power pedometer functionality allows the host processor to sleep while the DMP maintains the step count.

5.5 Clocking

- On-chip timing generator $\pm 1\%$ frequency variation over full temperature range
- Optional external clock inputs of 32.768kHz or 19.2MHz

	MPU-6000/MPU-6050 Product Specification	Document Number: PS-MPU-6000A-00 Revision: 3.4 Release Date: 08/19/2013
---	--	---


6 Electrical Characteristics

6.1 Gyroscope Specifications

VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V±5% or VDD, T_A = 25°C

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS	NOTES
GYROSCOPE SENSITIVITY						
Full-Scale Range	FS_SEL=0		±250		°/s	
	FS_SEL=1		±500		°/s	
	FS_SEL=2		±1000		°/s	
	FS_SEL=3		±2000		°/s	
Gyroscope ADC Word Length			16		bits	
Sensitivity Scale Factor	FS_SEL=0		131		LSB/(°/s)	
	FS_SEL=1		65,5		LSB/(°/s)	
	FS_SEL=2		32,8		LSB/(°/s)	
	FS_SEL=3		16,4		LSB/(°/s)	
Sensitivity Scale Factor Tolerance	25°C	-3		+3	%	
Sensitivity Scale Factor Variation Over Temperature			±2		%	
Nonlinearity	Best fit straight line; 25°C		0,2		%	
Cross-Axis Sensitivity			±2		%	
GYROSCOPE ZERO-RATE OUTPUT (ZRO)						
Initial ZRO Tolerance	25°C		±20		°/s	
ZRO Variation Over Temperature	-40°C to +85°C		±20		°/s	
Power-Supply Sensitivity (1-10Hz)	Sine wave, 100mVpp; VDD=2.5V		0,2		°/s	
Power-Supply Sensitivity (10 - 250Hz)	Sine wave, 100mVpp; VDD=2.5V		0,2		°/s	
Power-Supply Sensitivity (250Hz - 100kHz)	Sine wave, 100mVpp; VDD=2.5V		4		°/s	
Linear Acceleration Sensitivity	Static		0,1		°/s/g	
SELF-TEST RESPONSE						
Relative	Change from factory trim	-14		14	%	1
GYROSCOPE NOISE PERFORMANCE						
Total RMS Noise	FS_SEL=0 DLPFCFG=2 (100Hz)		0,05		°/s-rms	
Low-frequency RMS noise	Bandwidth 1Hz to 10Hz		0,033		°/s-rms	
Rate Noise Spectral Density	At 10Hz		0,005		°/s/√Hz	
GYROSCOPE MECHANICAL FREQUENCIES						
X-Axis		30	33	36	kHz	
Y-Axis		27	30	33	kHz	
Z-Axis		24	27	30	kHz	
LOW PASS FILTER RESPONSE						
	Programmable Range	5		256	Hz	
OUTPUT DATA RATE						
	Programmable	4		8,000	Hz	
GYROSCOPE START-UP TIME						
ZRO Settling (from power-on)	DLPFCFG=0 to ±1% of Final		30		ms	

1. Please refer to the following document for further information on Self-Test: *MPU-6000/MPU-6050 Register Map and Descriptions*


	MPU-6000/MPU-6050 Product Specification	Document Number: PS-MPU-6000A-00 Revision: 3.4 Release Date: 08/19/2013
---	--	---

6.2 Accelerometer Specifications

VDD = 2,375V-3,46V, VLOGIC (MPU-6050 only) = 1,8V±5% or VDD, T_A = 25°C

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS	NOTES
ACCELEROMETER SENSITIVITY						
Full-Scale Range	AFS_SEL=0 AFS_SEL=1 AFS_SEL=2 AFS_SEL=3		±2 ±4 ±8 ±16		g g g g	
ADC Word Length	Output in two's complement format		16		bits	
Sensitivity Scale Factor	AFS_SEL=0 AFS_SEL=1 AFS_SEL=2 AFS_SEL=3		16,384 8,192 4,096 2,048		LSB/g LSB/g LSB/g LSB/g	
Initial Calibration Tolerance			±3		%	
Sensitivity Change vs. Temperature	AFS_SEL=0, -40°C to +85°C		±0.02		%/°C	
Nonlinearity	Best Fit Straight Line		0.5		%	
Cross-Axis Sensitivity			±2		%	
ZERO-G OUTPUT						
Initial Calibration Tolerance	X and Y axes Z axis		±50 ±80		mg mg	1
Zero-G Level Change vs. Temperature	X and Y axes, 0°C to +70°C Z axis, 0°C to +70°C		±35 ±60		mg	
SELF TEST RESPONSE						
Relative	Change from factory trim	-14		14	%	2
NOISE PERFORMANCE						
Power Spectral Density	@10Hz, AFS_SEL=0 & ODR=1kHz		400		μg/√Hz	
LOW PASS FILTER RESPONSE						
	Programmable Range	5		260	Hz	
OUTPUT DATA RATE						
	Programmable Range	4		1,000	Hz	
INTELLIGENCE FUNCTION INCREMENT			32		mg/LSB	


1. Typical zero-g initial calibration tolerance value after MSL3 preconditioning
2. Please refer to the following document for further information on Self-Test: *MPU-6000/MPU-6050 Register Map and Descriptions*

	MPU-6000/MPU-6050 Product Specification	Document Number: PS-MPU-6000A-00 Revision: 3.4 Release Date: 08/19/2013
---	--	---

6.3 Electrical and Other Common Specifications

VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V±5% or VDD, T_A = 25°C


PARAMETER	CONDITIONS	MIN	TYP	MAX	Units	Notes
TEMPERATURE SENSOR						
Range			-40 to +85		°C	
Sensitivity	Untrimmed		340		LSB/°C	
Temperature Offset	35°C		-521		LSB	
Linearity	Best fit straight line (-40°C to +85°C)		±1		°C	
VDD POWER SUPPLY						
Operating Voltages		2.375		3.46	V	
Normal Operating Current	Gyroscope + Accelerometer + DMP		3.9		mA	
	Gyroscope + Accelerometer (DMP disabled)		3.8		mA	
	Gyroscope + DMP (Accelerometer disabled)		3.7		mA	
	Gyroscope only (DMP & Accelerometer disabled)		3.6		mA	
	Accelerometer only (DMP & Gyroscope disabled)		500		µA	
Accelerometer Low Power Mode Current	1.25 Hz update rate		10		µA	
	5 Hz update rate		20		µA	
	20 Hz update rate		70		µA	
	40 Hz update rate		140		µA	
Full-Chip Idle Mode Supply Current			5		µA	
Power Supply Ramp Rate	Monotonic ramp. Ramp rate is 10% to 90% of the final value			100	ms	
VLOGIC REFERENCE VOLTAGE						
Voltage Range	MPU-6050 only VLOGIC must be ≤VDD at all times	1.71		VDD	V	
Power Supply Ramp Rate	Monotonic ramp. Ramp rate is 10% to 90% of the final value			3	ms	
Normal Operating Current			100		µA	
TEMPERATURE RANGE						
Specified Temperature Range	Performance parameters are not applicable beyond Specified Temperature Range	-40		+85	°C	

	MPU-6000/MPU-6050 Product Specification	Document Number: PS-MPU-6000A-00 Revision: 3.4 Release Date: 08/19/2013
---	--	---

6.4 Electrical Specifications, Continued

VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V±5% or VDD, TA = 25°C

PARAMETER	CONDITIONS	MIN	TYP	MAX	Units	Notes
SERIAL INTERFACE						
SPI Operating Frequency, All Registers Read/Write	MPU-6000 only, Low Speed Characterization		100 ±10%		kHz	
	MPU-6000 only, High Speed Characterization		1 ±10%		MHz	
SPI Operating Frequency, Sensor and Interrupt Registers Read Only	MPU-6000 only		20 ±10%		MHz	
I ² C Operating Frequency	All registers, Fast-mode			400	kHz	
	All registers, Standard-mode			100	kHz	
I²C ADDRESS						
	AD0 = 0		1101000			
	AD0 = 1		1101001			
DIGITAL INPUTS (SDI/SDA, AD0, SCLK/SCL, FSYNC, /CS, CLKIN)						
V _{IH} , High Level Input Voltage	MPU-6000	0.7*VDD			V	
	MPU-6050	0.7*VLOGIC			V	
V _{IL} , Low Level Input Voltage	MPU-6000			0.3*VDD	V	
	MPU-6050			0.3*VLOGIC	V	
C _i , Input Capacitance			< 5		pF	
DIGITAL OUTPUT (SDO, INT)						
V _{OH} , High Level Output Voltage	R _{LOAD} =1MΩ; MPU-6000	0.9*VDD			V	
	R _{LOAD} =1MΩ; MPU-6050	0.9*VLOGIC			V	
V _{OL1} , LOW-Level Output Voltage	R _{LOAD} =1MΩ; MPU-6000			0.1*VDD	V	
	R _{LOAD} =1MΩ; MPU-6050			0.1*VLOGIC	V	
V _{OLINT1} , INT Low-Level Output Voltage	OPEN=1, 0.3mA sink Current			0.1	V	
Output Leakage Current	OPEN=1		100		nA	
t _{INT} , INT Pulse Width	LATCH_INT_EN=0		50		μs	

	MPU-6000/MPU-6050 Product Specification	Document Number: PS-MPU-6000A-00 Revision: 3.4 Release Date: 08/19/2013
---	--	---

6.5 Electrical Specifications, Continued

Typical Operating Circuit of Section 7.2, VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V±5% or VDD, T_A = 25°C

Parameters	Conditions	Typical	Units	Notes
Primary I²C I/O (SCL, SDA)				
V _{IL} , LOW-Level Input Voltage	MPU-6000	-0.5 to 0.3*VDD	V	
V _{IH} , HIGH-Level Input Voltage	MPU-6000	0.7*VDD to VDD + 0.5V	V	
V _{hys} , Hysteresis	MPU-6000	0.1*VDD	V	
V _{IL} , LOW Level Input Voltage	MPU-6050	-0.5V to 0.3*VLOGIC	V	
V _{IH} , HIGH-Level Input Voltage	MPU-6050	0.7*VLOGIC to VLOGIC + 0.5V	V	
V _{hys} , Hysteresis	MPU-6050	0.1*VLOGIC	V	
V _{OL1} , LOW-Level Output Voltage	3mA sink current	0 to 0.4	V	
I _{OL} , LOW-Level Output Current	V _{OL} = 0.4V	3	mA	
	V _{OL} = 0.6V	5	mA	
Output Leakage Current		100	nA	
t _{of} , Output Fall Time from V _{IHmax} to V _{ILmax}	C _b bus capacitance in pF	20+0.1C _b to 250	ns	
C _I , Capacitance for Each I/O pin		< 10	pF	
Auxiliary I²C I/O (AUX_CL, AUX_DA)				
MPU-6050: AUX_VDDIO=0				
V _{IL} , LOW-Level Input Voltage		-0.5V to 0.3*VLOGIC	V	
V _{IH} , HIGH-Level Input Voltage		0.7*VLOGIC to VLOGIC + 0.5V	V	
V _{hys} , Hysteresis		0.1*VLOGIC	V	
V _{OL1} , LOW-Level Output Voltage	VLOGIC > 2V; 1mA sink current	0 to 0.4	V	
V _{OL3} , LOW-Level Output Voltage	VLOGIC < 2V; 1mA sink current	0 to 0.2*VLOGIC	V	
I _{OL} , LOW-Level Output Current	V _{OL} = 0.4V	1	mA	
	V _{OL} = 0.6V	1	mA	
Output Leakage Current		100	nA	
t _{of} , Output Fall Time from V _{IHmax} to V _{ILmax}	C _b bus capacitance in pF	20+0.1C _b to 250	ns	
C _I , Capacitance for Each I/O pin		< 10	pF	

	MPU-6000/MPU-6050 Product Specification	Document Number: PS-MPU-6000A-00 Revision: 3.4 Release Date: 08/19/2013
---	--	---

6.6 Electrical Specifications, Continued

Typical Operating Circuit of Section 7.2, VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V±5% or VDD, T_A = 25°C

Parameters	Conditions	Min	Typical	Max	Units	Notes
INTERNAL CLOCK SOURCE						
Gyroscope Sample Rate, Fast	CLK_SEL=0,1,2,3 DLPFCFG=0 SAMPLERATEDIV = 0		8		kHz	
Gyroscope Sample Rate, Slow	DLPFCFG=1,2,3,4,5, or 6 SAMPLERATEDIV = 0		1		kHz	
Accelerometer Sample Rate			1		kHz	
Clock Frequency Initial Tolerance	CLK_SEL=0, 25°C	-5		+5	%	
Frequency Variation over Temperature	CLK_SEL=1,2,3; 25°C	-1		+1	%	
	CLK_SEL=0		-15 to +10		%	
PLL Settling Time	CLK_SEL=1,2,3		±1		%	
	CLK_SEL=1,2,3		1	10	ms	
EXTERNAL 32,768kHz CLOCK						
External Clock Frequency	CLK_SEL=4		32,768		kHz	
External Clock Allowable Jitter	Cycle-to-cycle rms		1 to 2		µs	
Gyroscope Sample Rate, Fast	DLPFCFG=0 SAMPLERATEDIV = 0		8,192		kHz	
Gyroscope Sample Rate, Slow	DLPFCFG=1,2,3,4,5, or 6 SAMPLERATEDIV = 0		1,024		kHz	
Accelerometer Sample Rate			1,024		kHz	
PLL Settling Time			1	10	ms	
EXTERNAL 19,2MHz CLOCK						
External Clock Frequency	CLK_SEL=5		19.2		MHz	
Gyroscope Sample Rate	Full programmable range	3.9		8000	Hz	
Gyroscope Sample Rate, Fast Mode	DLPFCFG=0 SAMPLERATEDIV = 0		8		kHz	
Gyroscope Sample Rate, Slow Mode	DLPFCFG=1,2,3,4,5, or 6 SAMPLERATEDIV = 0		1		kHz	
Accelerometer Sample Rate			1		kHz	
PLL Settling Time			1	10	ms	

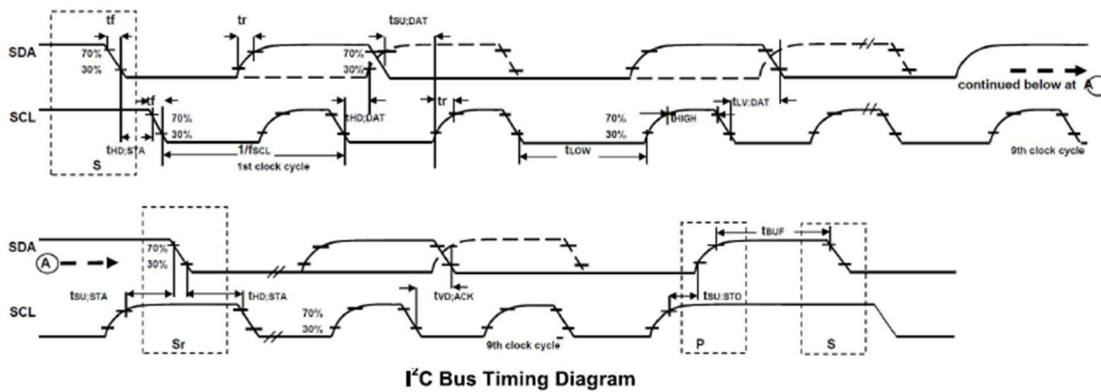
	MPU-6000/MPU-6050 Product Specification	Document Number: PS-MPU-6000A-00 Revision: 3.4 Release Date: 08/19/2013
---	--	---

6.7 I²C Timing Characterization

Typical Operating Circuit of Section 7.2, VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V±5% or VDD, T_A = 25°C

Parameters	Conditions	Min	Typical	Max	Units	Notes
I²C TIMING						
f _{SCL} , SCL Clock Frequency	I ² C FAST-MODE			400	kHz	
t _{HD,STA} , (Repeated) START Condition Hold Time		0.6			µs	
t _{LOW} , SCL Low Period		1.3			µs	
t _{HIGH} , SCL High Period		0.6			µs	
t _{SU,STA} , Repeated START Condition Setup Time		0.6			µs	
t _{HD,DAT} , SDA Data Hold Time		0			µs	
t _{SU,DAT} , SDA Data Setup Time		100			ns	
t _r , SDA and SCL Rise Time	C _b bus cap. from 10 to 400pF	20+0.1C _b		300	ns	
t _f , SDA and SCL Fall Time	C _b bus cap. from 10 to 400pF	20+0.1C _b		300	ns	
t _{SU,STO} , STOP Condition Setup Time		0.6			µs	
t _{BUF} , Bus Free Time Between STOP and START Condition		1.3			µs	
C _b , Capacitive Load for each Bus Line			< 400		pF	
t _{VD,DAT} , Data Valid Time				0.9	µs	
t _{VD,ACK} , Data Valid Acknowledge Time				0.9	µs	

Note: Timing Characteristics apply to both Primary and Auxiliary I²C Bus



	MPU-6000/MPU-6050 Product Specification	Document Number: PS-MPU-6000A-00 Revision: 3.4 Release Date: 08/19/2013
---	--	---

6.9 Absolute Maximum Ratings

Stress above those listed as "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only and functional operation of the device at these conditions is not implied. Exposure to the absolute maximum ratings conditions for extended periods may affect device reliability.

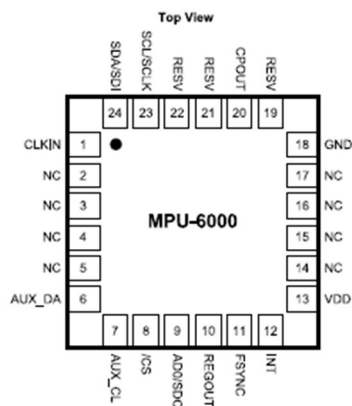
Parameter	Rating
Supply Voltage, VDD	-0.5V to +6V
VLOGIC Input Voltage Level (MPU-6050)	-0.5V to VDD + 0.5V
REGOUT	-0.5V to 2V
Input Voltage Level (CLKIN, AUX_DA, AD0, FSYNC, INT, SCL, SDA)	-0.5V to VDD + 0.5V
CPOUT (2.5V ≤ VDD ≤ 3.6V)	-0.5V to 30V
Acceleration (Any Axis, unpowered)	10,000g for 0.2ms
Operating Temperature Range	-40°C to +105°C
Storage Temperature Range	-40°C to +125°C
Electrostatic Discharge (ESD) Protection	2kV (HBM); 250V (MM)
Latch-up	JEDEC Class II (2), 125°C ±100mA

	MPU-6000/MPU-6050 Product Specification	Document Number: PS-MPU-6000A-00 Revision: 3.4 Release Date: 08/19/2013
---	--	---

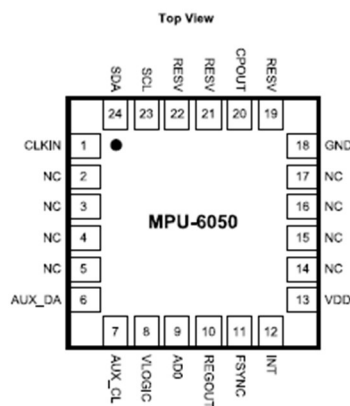
7 Applications Information

7.1 Pin Out and Signal Description

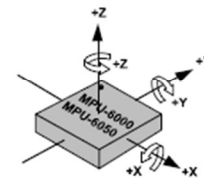
Pin Number	MPU-6000	MPU-6050	Pin Name	Pin Description
1	Y	Y	CLKIN	Optional external reference clock input. Connect to GND if unused.
6	Y	Y	AUX_DA	I ² C master serial data, for connecting to external sensors
7	Y	Y	AUX_CL	I ² C Master serial clock, for connecting to external sensors
8	Y		/CS	SPI chip select (0=SPI mode)
8		Y	VLOGIC	Digital I/O supply voltage
9	Y		AD0 / SDO	I ² C Slave Address LSB (AD0); SPI serial data output (SDO)
9		Y	AD0	I ² C Slave Address LSB (AD0)
10	Y	Y	REGOUT	Regulator filter capacitor connection
11	Y	Y	FSYNC	Frame synchronization digital input. Connect to GND if unused.
12	Y	Y	INT	Interrupt digital output (totem pole or open-drain)
13	Y	Y	VDD	Power supply voltage and Digital I/O supply voltage
18	Y	Y	GND	Power supply ground
19, 21	Y	Y	RESV	Reserved. Do not connect.
20	Y	Y	CPOUT	Charge pump capacitor connection
22	Y	Y	RESV	Reserved. Do not connect.
23	Y		SCL / SCLK	I ² C serial clock (SCL); SPI serial clock (SCLK)
23		Y	SCL	I ² C serial clock (SCL)
24	Y		SDA / SDI	I ² C serial data (SDA); SPI serial data input (SDI)
24		Y	SDA	I ² C serial data (SDA)
2, 3, 4, 5, 14, 15, 16, 17	Y	Y	NC	Not internally connected. May be used for PCB trace routing.



QFN Package
24-pin, 4mm x 4mm x 0,9mm



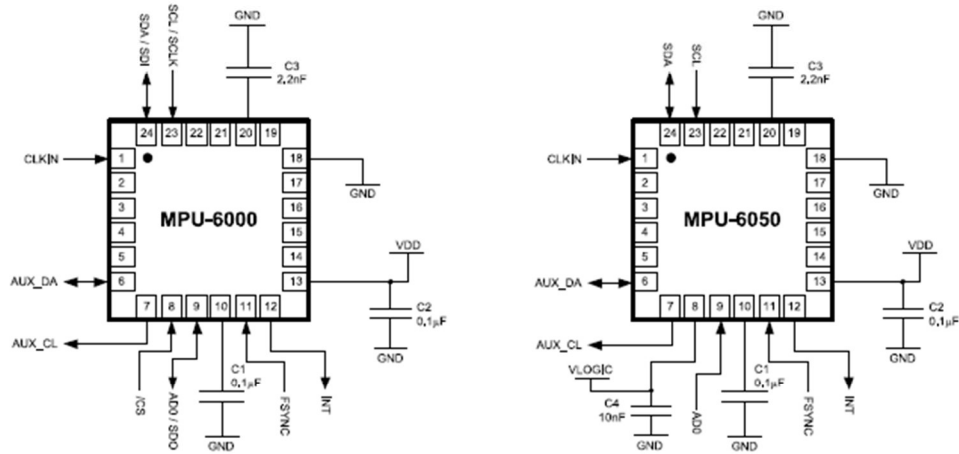
QFN Package
24-pin, 4mm x 4mm x 0,9mm



**Orientation of Axes of Sensitivity and
Polarity of Rotation**

	MPU-6000/MPU-6050 Product Specification	Document Number: PS-MPU-6000A-00 Revision: 3.4 Release Date: 08/19/2013
---	--	---

7.2 Typical Operating Circuit



Typical Operating Circuits

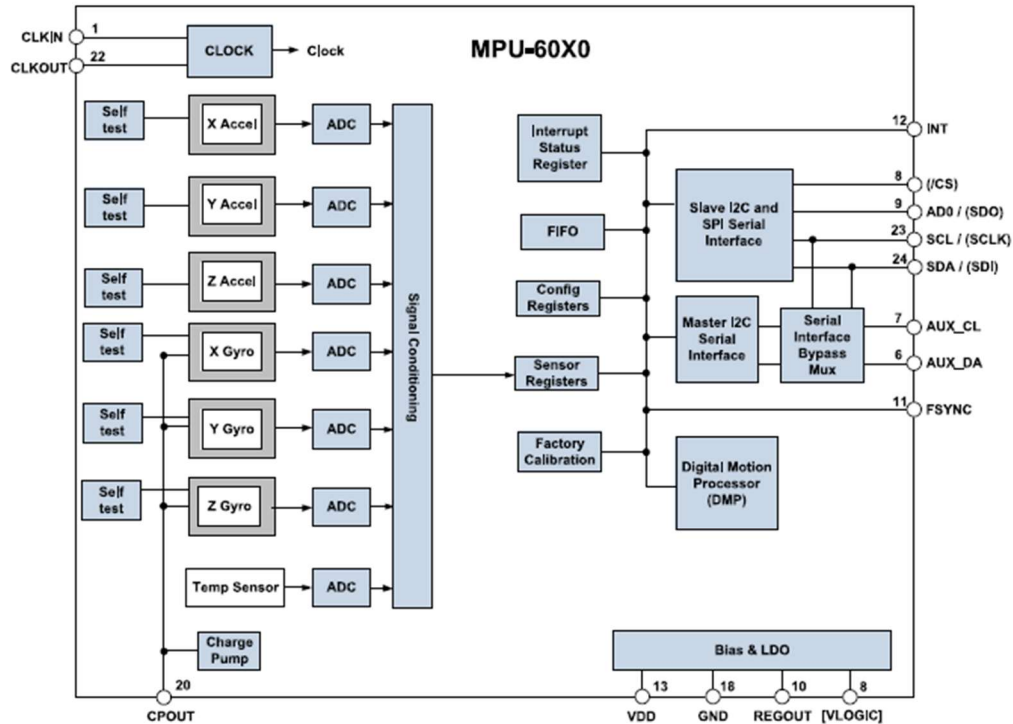
7.3 Bill of Materials for External Components

Component	Label	Specification	Quantity
Regulator Filter Capacitor (Pin 10)	C1	Ceramic, X7R, 0.1µF ±10%, 2V	1
VDD Bypass Capacitor (Pin 13)	C2	Ceramic, X7R, 0.1µF ±10%, 4V	1
Charge Pump Capacitor (Pin 20)	C3	Ceramic, X7R, 2.2nF ±10%, 50V	1
VLOGIC Bypass Capacitor (Pin 8)	C4*	Ceramic, X7R, 10nF ±10%, 4V	1

* MPU-6050 Only.

	MPU-6000/MPU-6050 Product Specification	Document Number: PS-MPU-6000A-00 Revision: 3.4 Release Date: 08/19/2013
---	--	---

7.5 Block Diagram



Note: Pin names in round brackets () apply only to MPU-6000
Pin names in square brackets [] apply only to MPU-6050

7.6 Overview

The MPU-60X0 is comprised of the following key blocks and functions:

- Three-axis MEMS rate gyroscope sensor with 16-bit ADCs and signal conditioning
- Three-axis MEMS accelerometer sensor with 16-bit ADCs and signal conditioning
- Digital Motion Processor (DMP) engine
- Primary I²C and SPI (MPU-6000 only) serial communications interfaces
- Auxiliary I²C serial interface for 3rd party magnetometer & other sensors
- Clocking
- Sensor Data Registers
- FIFO
- Interrupts
- Digital-Output Temperature Sensor
- Gyroscope & Accelerometer Self-test
- Bias and LDO
- Charge Pump

	MPU-6000/MPU-6050 Product Specification	Document Number: PS-MPU-6000A-00 Revision: 3.4 Release Date: 08/19/2013
---	--	---

7.7 Three-Axis MEMS Gyroscope with 16-bit ADCs and Signal Conditioning

The MPU-60X0 consists of three independent vibratory MEMS rate gyroscopes, which detect rotation about the X-, Y-, and Z- Axes. When the gyros are rotated about any of the sense axes, the Coriolis Effect causes a vibration that is detected by a capacitive pickoff. The resulting signal is amplified, demodulated, and filtered to produce a voltage that is proportional to the angular rate. This voltage is digitized using individual on-chip 16-bit Analog-to-Digital Converters (ADCs) to sample each axis. The full-scale range of the gyro sensors may be digitally programmed to ± 250 , ± 500 , ± 1000 , or ± 2000 degrees per second (dps). The ADC sample rate is programmable from 8,000 samples per second, down to 3.9 samples per second, and user-selectable low-pass filters enable a wide range of cut-off frequencies.

7.8 Three-Axis MEMS Accelerometer with 16-bit ADCs and Signal Conditioning

The MPU-60X0's 3-Axis accelerometer uses separate proof masses for each axis. Acceleration along a particular axis induces displacement on the corresponding proof mass, and capacitive sensors detect the displacement differentially. The MPU-60X0's architecture reduces the accelerometers' susceptibility to fabrication variations as well as to thermal drift. When the device is placed on a flat surface, it will measure 0g on the X- and Y-axes and +1g on the Z-axis. The accelerometers' scale factor is calibrated at the factory and is nominally independent of supply voltage. Each sensor has a dedicated sigma-delta ADC for providing digital outputs. The full scale range of the digital output can be adjusted to $\pm 2g$, $\pm 4g$, $\pm 8g$, or $\pm 16g$.

7.9 Digital Motion Processor

The embedded Digital Motion Processor (DMP) is located within the MPU-60X0 and offloads computation of motion processing algorithms from the host processor. The DMP acquires data from accelerometers, gyroscopes, and additional 3rd party sensors such as magnetometers, and processes the data. The resulting data can be read from the DMP's registers, or can be buffered in a FIFO. The DMP has access to one of the MPU's external pins, which can be used for generating interrupts.

The purpose of the DMP is to offload both timing requirements and processing power from the host processor. Typically, motion processing algorithms should be run at a high rate, often around 200Hz, in order to provide accurate results with low latency. This is required even if the application updates at a much lower rate; for example, a low power user interface may update as slowly as 5Hz, but the motion processing should still run at 200Hz. The DMP can be used as a tool in order to minimize power, simplify timing, simplify the software architecture, and save valuable MIPS on the host processor for use in the application.


7.10 Primary I²C and SPI Serial Communications Interfaces

The MPU-60X0 communicates to a system processor using either a SPI (MPU-6000 only) or an I²C serial interface. The MPU-60X0 always acts as a slave when communicating to the system processor. The LSB of the I²C slave address is set by pin 9 (AD0).

The logic levels for communications between the MPU-60X0 and its master are as follows:

- MPU-6000: The logic level for communications with the master is set by the voltage on VDD
- MPU-6050: The logic level for communications with the master is set by the voltage on VLOGIC

For further information regarding the logic levels of the MPU-6050, please refer to Section 10.

	MPU-6000/MPU-6050 Product Specification	Document Number: PS-MPU-6000A-00 Revision: 3.4 Release Date: 08/19/2013
---	--	---

9 Digital Interface

9.1 I²C and SPI (MPU-6000 only) Serial Interfaces

The internal registers and memory of the MPU-6000/MPU-6050 can be accessed using either I²C at 400 kHz or SPI at 1MHz (MPU-6000 only). SPI operates in four-wire mode.

Serial Interface

Pin Number	MPU-6000	MPU-6050	Pin Name	Pin Description
8	Y		/CS	SPI chip select (0=SPI enable)
8		Y	VLOGIC	Digital I/O supply voltage. VLOGIC must be ≤ VDD at all times.
9	Y		AD0 / SDO	I ² C Slave Address LSB (AD0); SPI serial data output (SDO)
9		Y	AD0	I ² C Slave Address LSB
23	Y		SCL / SCLK	I ² C serial clock (SCL); SPI serial clock (SCLK)
23		Y	SCL	I ² C serial clock
24	Y		SDA / SDI	I ² C serial data (SDA); SPI serial data input (SDI)
24		Y	SDA	I ² C serial data

Note:

To prevent switching into I²C mode when using SPI (MPU-6000), the I²C interface should be disabled by setting the *I2C_IF_DIS* configuration bit. Setting this bit should be performed immediately after waiting for the time specified by the "Start-Up Time for Register Read/Write" in Section 6.3.

For further information regarding the *I2C_IF_DIS* bit, please refer to the MPU-6000/MPU-6050 Register Map and Register Descriptions document.

9.2 I²C Interface

I²C is a two-wire interface comprised of the signals serial data (SDA) and serial clock (SCL). In general, the lines are open-drain and bi-directional. In a generalized I²C interface implementation, attached devices can be a master or a slave. The master device puts the slave address on the bus, and the slave device with the matching address acknowledges the master.

The MPU-60X0 always operates as a slave device when communicating to the system processor, which thus acts as the master. SDA and SCL lines typically need pull-up resistors to VDD. The maximum bus speed is 400 kHz.

The slave address of the MPU-60X0 is b110100X which is 7 bits long. The LSB bit of the 7 bit address is determined by the logic level on pin AD0. This allows two MPU-60X0s to be connected to the same I²C bus. When used in this configuration, the address of the one of the devices should be b1101000 (pin AD0 is logic low) and the address of the other should be b1101001 (pin AD0 is logic high).

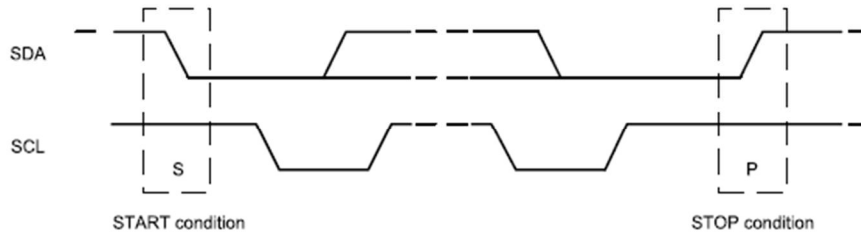
9.3 I²C Communications Protocol

START (S) and STOP (P) Conditions

Communication on the I²C bus starts when the master puts the START condition (S) on the bus, which is defined as a HIGH-to-LOW transition of the SDA line while SCL line is HIGH (see figure below). The bus is considered to be busy until the master puts a STOP condition (P) on the bus, which is defined as a LOW to HIGH transition on the SDA line while SCL is HIGH (see figure below).

	MPU-6000/MPU-6050 Product Specification	Document Number: PS-MPU-6000A-00 Revision: 3.4 Release Date: 08/19/2013
---	--	---

Additionally, the bus remains busy if a repeated START (Sr) is generated instead of a STOP condition,

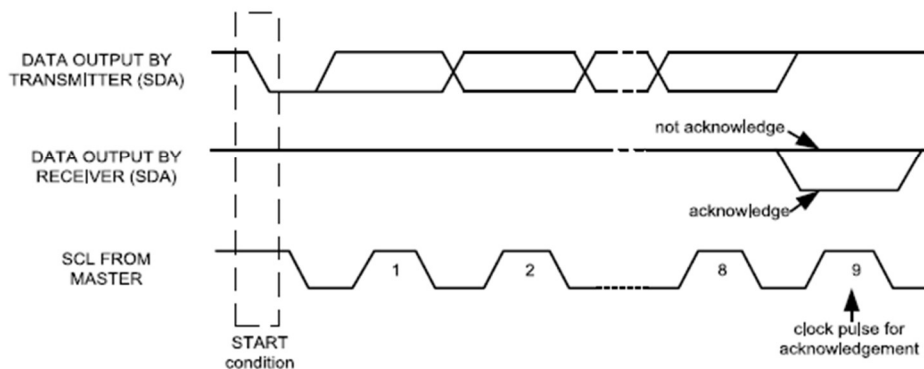


START and STOP Conditions


Data Format / Acknowledge

I²C data bytes are defined to be 8-bits long. There is no restriction to the number of bytes transmitted per data transfer. Each byte transferred must be followed by an acknowledge (ACK) signal. The clock for the acknowledge signal is generated by the master, while the receiver generates the actual acknowledge signal by pulling down SDA and holding it low during the HIGH portion of the acknowledge clock pulse.

If a slave is busy and cannot transmit or receive another byte of data until some other task has been performed, it can hold SCL LOW, thus forcing the master into a wait state. Normal data transfer resumes when the slave is ready, and releases the clock line (refer to the following figure).

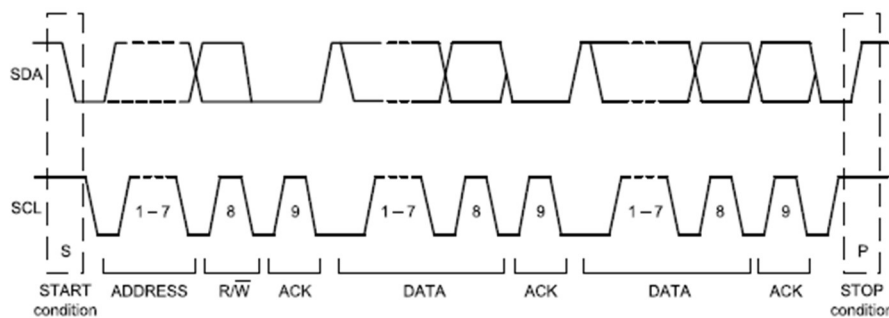


Acknowledge on the I²C Bus

	MPU-6000/MPU-6050 Product Specification	Document Number: PS-MPU-6000A-00 Revision: 3,4 Release Date: 08/19/2013
---	--	---

Communications

After beginning communications with the START condition (S), the master sends a 7-bit slave address followed by an 8th bit, the read/write bit. The read/write bit indicates whether the master is receiving data from or is writing to the slave device. Then, the master releases the SDA line and waits for the acknowledge signal (ACK) from the slave device. Each byte transferred must be followed by an acknowledge bit. To acknowledge, the slave device pulls the SDA line LOW and keeps it LOW for the high period of the SCL line. Data transmission is always terminated by the master with a STOP condition (P), thus freeing the communications line. However, the master can generate a repeated START condition (Sr), and address another slave without first generating a STOP condition (P). A LOW to HIGH transition on the SDA line while SCL is HIGH defines the stop condition. All SDA changes should take place when SCL is low, with the exception of start and stop conditions.



Complete I²C Data Transfer

To write the internal MPU-60X0 registers, the master transmits the start condition (S), followed by the I²C address and the write bit (0). At the 9th clock cycle (when the clock is high), the MPU-60X0 acknowledges the transfer. Then the master puts the register address (RA) on the bus. After the MPU-60X0 acknowledges the reception of the register address, the master puts the register data onto the bus. This is followed by the ACK signal, and data transfer may be concluded by the stop condition (P). To write multiple bytes after the last ACK signal, the master can continue outputting data rather than transmitting a stop signal. In this case, the MPU-60X0 automatically increments the register address and loads the data to the appropriate register. The following figures show single and two-byte write sequences.

Single-Byte Write Sequence

Master	S	AD+W		RA		DATA		P
Slave			ACK		ACK		ACK	

Burst Write Sequence

Master	S	AD+W		RA		DATA		DATA		P
Slave			ACK		ACK		ACK		ACK	

	MPU-6000/MPU-6050 Product Specification	Document Number: PS-MPU-6000A-00 Revision: 3.4 Release Date: 08/19/2013
---	--	---

To read the internal MPU-60X0 registers, the master sends a start condition, followed by the I²C address and a write bit, and then the register address that is going to be read. Upon receiving the ACK signal from the MPU-60X0, the master transmits a start signal followed by the slave address and read bit. As a result, the MPU-60X0 sends an ACK signal and the data. The communication ends with a not acknowledge (NACK) signal and a stop bit from master. The NACK condition is defined such that the SDA line remains high at the 9th clock cycle. The following figures show single and two-byte read sequences.

Single-Byte Read Sequence

Master	S	AD+W		RA		S	AD+R			NACK	P
Slave			ACK		ACK			ACK	DATA		

Burst Read Sequence

Master	S	AD+W		RA		S	AD+R			ACK		NACK	P
Slave			ACK		ACK			ACK	DATA		DATA		

9.4 I²C Terms

Signal	Description
S	Start Condition: SDA goes from high to low while SCL is high
AD	Slave I ² C address
W	Write bit (0)
R	Read bit (1)
ACK	Acknowledge: SDA line is low while the SCL line is high at the 9 th clock cycle
NACK	Not-Acknowledge: SDA line stays high at the 9 th clock cycle
RA	MPU-60X0 internal register address
DATA	Transmit or received data
P	Stop condition: SDA going from low to high while SCL is high

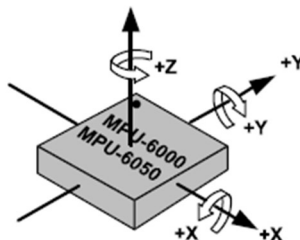
	MPU-6000/MPU-6050 Product Specification	Document Number: PS-MPU-6000A-00 Revision: 3.4 Release Date: 08/19/2013
---	--	---

11 Assembly

This section provides general guidelines for assembling InvenSense Micro Electro-Mechanical Systems (MEMS) gyros packaged in Quad Flat No leads package (QFN) surface mount integrated circuits.

11.1 Orientation of Axes

The diagram below shows the orientation of the axes of sensitivity and the polarity of rotation, Note the pin 1 identifier (•) in the figure.



**Orientation of Axes of Sensitivity and
Polarity of Rotation**

ANEXO VII

ANEXO VII.

DOCUMENTACIÓN GRÁFICA

VÍDEO OBTENIDO DEL MODELO

- <https://youtu.be/QHm-K6O5aI8>

En el vídeo adjuntado previamente se muestra el resultado obtenido tras la ejecución del código con el modelo. Con él se decidió realizar la grabación de un cronómetro de manera que se aprecie, además de la correcta obtención y tratamiento de imágenes realizada, que la pérdida de información derivada de la metodología de toma de imágenes no resulta apreciable a simple vista, siendo el único factor apreciable la atenuación de las imágenes por un muy limitado periodo de tiempo.

FUNCIONAMIENTO DEL CÓDIGO

- <https://youtu.be/Y14b3YMs2zA>

En el vídeo adjuntado previamente se muestra el procedimiento que se sigue internamente con la obtención de ficheros, su borrado y su tratamiento posterior a la detección de un impacto para lograr finalmente la obtención del resultado final. En este vídeo se muestra por un lado la consola de comandos, en la que se muestra por pantalla la fase en la que se encuentra la ejecución del código, y por otro lado el directorio en el que se guardan los archivos obtenidos.

VÍDEO JUSTIFICATIVO DE LA PÉRDIDA DE FOTOGRAMAS

- https://youtu.be/or6icOB59_U

En el vídeo adjuntado previamente se muestra la ejecución del código de prueba realizado para lograr la justificación de los fotogramas perdidos debido a la metodología para la toma de vídeo empleada. En él, se muestra cómo, tras la ejecución del código, se muestran por pantalla todos aquellos valores obtenidos, así como su promedio.