Trabajo de Fin de Máster
"Máster Universitario en Microelectónica:
Diseño y Aplicaciones de Sistemas
Micro/Nanométricos"

Sistema de seguimiento tridimensional basado en un array de sensores de flujo óptico implementados con cámaras de baja resolución

Autor: Rafael de la Rosa Vidal
Tutor: Juan Antonio Leñero Bardallo
30 Noviembre 2020

A mi familia
A mis profesores

Agradecimientos

Después de un año de trabajo y mucho esfuerzo, por el hecho de combinar la vida laboral con estudio superiores, me dispongo a cerrar un nuevo capítulo en mi carrera académica. Desde el principio he estado rodeado de personas que me han aportado el conocimiento y el apoyo necesarios para alcanzar todas las metas que me he propuesto. A todos ellos quiero darles las gracias.

En primer lugar, me gustaría agradecerles a todos los profesores y profesionales que han construido las bases de mi formación por su dedicación y tesón. En especial a mi tutor, Juan Antonio Leñero, quién ha puesto a mi disposición todo el conocimiento y las herramientas necesarias para la ejecución de este trabajo así como una total confianza en mí. A José María Guerrero, a quien le tengo mucho aprecio y respeto, que me ha brindado oportunidades de crecimiento profesional por las cuales siempre le estaré inmensamente agradecido y que, indirectamente, también es parte de este proyecto. A Jesús Álvarez, un gran profesional que me ha enseñado una buena parte de lo que en este trabajo se refleja y, que es una gran persona a la que le tengo mucha estima, siempre me ha servido de apoyo y motivación.

A mi familia, un pilar fundamental en mi vida, por apoyarme en todas mis decisiones y por hacer de mí la persona que soy a día de hoy. A mi pareja, por su apoyo incondicional, por soportarme y por ser mi principal fuente de motivación cada vez que algo se tuerce. Siempre os estaré eternamente agradecido.

Resumen

En este Trabajo Fin de Máster se ha diseñado, implementado y testado un sistema de seguimiento tridimensional basado en un array de sensores de *Flujo Óptico* procesando imágenes captadas con cámaras de baja resolución. Se trata de un proyecto multidisciplinar en el que se han abarcado tareas de diseño mecánico, diseño hardware, desarrollo firmware y software.

El objetivo global del proyecto ha sido construir una plataforma móvil que permite posicionar arrays de sensores de imagen para seguir objetos móviles mediante el cómputo del *Flujo Óptico* minimizando el coste computacional. El sistema de seguimiento debe ser versátil y poderse adaptar a otros algoritmos y tipos de sensores.

La elección de cámaras de baja resolución para el diseño de los sensores de *Flujo Óptico* permite aliviar los requisitos computacionales para su procesamiento, siendo la información contenida en las imágenes suficiente para tal fin. Por otro lado, el algoritmo utilizado presenta un menor grado de requisitos computacionales en comparación con el estado del arte. La implementación de todo el firmware desarrollado se ha orientado al aumento del rendimiento del sistema, para ello se ejecutan estrategias que eliminan los tiempos de holgura entre la adquisición de los datos y su procesamiento. Se consigue alcanzar una tasa de adquisición más procesamiento de 5 FPS para cada sensor del sistema, lo que para un total de dos sensores equivale a 10 FPS.

La metodología del proyecto ha sido la siguiente: se parte del diseño de una PCB y puesta en marcha de los algoritmos de *Flujo Óptico* con orientación al rendimiento computacional. Luego se hace el diseño mecánico y modelado 3D de todos los elementos del sistema, dando lugar a un modelo virtual 3D del sistema completo. Se fabrican las piezas necesarias para la fabricación de la plataforma y, finalmente, se llevan a cabo tareas de configuración y control automático.

Con el fin de alcanzar la función de seguimiento deseada, se diseña una interfaz de usuario que permite modificar el comportamiento de la plataforma, así como la monitorización de todos los datos generados por esta. A partir de estos datos, se confecciona una ley de control que vincula el valor de *Flujo Óptico* devuelto por los módulos *EyesOF* con el valor de las señales de control de posición de la plataforma.

El sistema implementado es una plataforma robótica versátil útil para implementar y testar algoritmos de seguimiento que requieran el posicionamiento de cámaras. Puede usarse con sensores y algoritmos de distinta naturaleza a los presentados en el proyecto para resolver problemas diversos que abarquen el posicionamiento y el seguimiento de trayectorias.

Abstract

In this Master's thesis, a tridimensional tracking system based on an *Optical Flow* sensor array was designed and tested. The *Optical Flow* sensors were implemented with low-resolution cameras. The project is multidisciplinary as it includes mechanical design, hardware design, software development and firmware development tasks.

The global project target has been the robotic platform construction which allows to position an image sensor array to track mobile objects employing the *Optical Flow* computation and maximizing the system performance. The system must be versatile and adaptable for other algorithms and image sensor types.

The low-resolution cameras chosen for the *Optical Flow* sensor design allow alleviating the computational requirements for its processing. The information contained in the low-resolution images is enough for the *Optical Flow* computation. On the other side, the used algorithm presents a minor grade of computational requirements in comparison with the state-of-the-art. The firmware implementation was oriented to increase the system performance using code strategies which remove slack times between the data acquisition and their processing. Due to this, it has reached an acquisition and processing rate of about 5 FPS for each *Optical Flow* sensor, as the system has two sensors, the total rate achieved is about 10FPS.

The study starts with the PCB design and performance-oriented *Optical Flow* algorithm setting-up. Then it is done the mechanical design and 3D model of all system elements, giving rise to a 3D virtual model of the complete system. Some parts of the platform were made as the platform construction need them. Finally, a set of tasks about setup and automatic control have been accomplished.

To implement the desired mobile object tracking function, it was developed a desktop application which makes possible to change the platform behaviour as well as the monitoring of all the data generated by the platform. From the gathered data, it was composed a control law which relates the control signals and the platform position.

The developed system is a versatile robotic platform useful for algorithms about tracking function implementation and their testing which requires the camera positioning. The system can be used with algorithms and sensor types different from the ones presented in this study. A large number of functions about positioning or trajectory tracking are suitable for this platform.

Índice general

1. Introducción y motivación **1**
1.1. Motivaciones y objetivos 2

2. Estado del arte **5**

3. Flujo óptico **9**
3.1. Teoría del cálculo del *Flujo Óptico* 9
3.2. Algoritmos de cálculo de *Flujo Óptico* 12
3.2.1. Algoritmo de Lucas y Kanade 13
3.2.2. Algoritmo de Horn y Schunck 14
3.3. Algoritmo de interpolación de Srinivasan 15

4. Diseño hardware y firmware para la adquisición del frame y el cálculo del *Flujo Óptico* **19**
4.1. Sensor óptico ADNS2610 19
4.2. Descripción de la arquitectura hardware del dispositivo 20
4.2.1. Módulo EyeOF 22
4.2.2. Construcción del dispositivo 24
4.3. Descripción de la arquitectura firmware del dispositivo 26
4.3.1. Configuración del microcontrolador en la herramienta STM32 CubeMX . . 27
4.3.2. Diseño de la máquina de estados finitos que define el funcionamiento del dispositivo 28
4.3.3. Configuración y lectura del sensor ADNS2610 30
4.3.4. Implementación en código del cálculo de *Flujo Óptico* 31
4.3.5. Fusión de los vectores de *Flujo Óptico* 31
4.3.6. Transferencia de datos al PC 33

5. Diseño y control de la plataforma móvil **35**
5.1. Motores DC brushless 35
5.2. Controlador STorM32 BGC 37
5.3. Diseño mecánico de la plataforma 39
5.4. Definición e implementación de la función de control 42
5.4.1. Bucle de control para la posición: Configuración del controlador STorM32 BGC y estabilización de la plataforma 44
5.4.2. Bucle de control para el seguimiento: Configuración de las entradas de control de posición y ajuste de los reguladores PID 47

6. Desarrollo de la interfaz de usuario **51**
6.1. Tecnología WPF y .NET Core 51
6.2. Paradigma de programación MVVM (Model-View-ViewModel) 51

6.3. Diseño gráfico de la aplicación 52
6.3.1. Monitorización del frame 52
6.3.2. Monitorización del *Flujo Óptico* 53
6.4. Diseño funcional de la aplicación 55
6.4.1. Inicialización de la aplicación y comandos disponibles 55
6.4.2. Posición de la plataforma y función seguimiento 56
6.4.3. Función calibración 56

7. Conclusiones y líneas de trabajo futuras **59**

Bibliogafía **60**

Anexos **69**
A. Rutina principal 71
B. Librería para la comunicación con el sensor ADNS2610 75
C. Librería para el cálculo de flujo óptico 83
D. Librería para el desarrollo de la máquina de estados finitos implementada 91
E. Librería para la implementación del control de la plataforma 101
F. Descripción en XAML de la interfaz 107
G. Descripción en XAML de la vista para monitorización de la imagen y el flujo óptico 111
H. 'View Model' de la aplicación 'Eyes OF Gimbal Platform' 115
I. 'Librería para recibir datos desde la plataforma en la aplicación 'Eyes OF Gimbal Platform' 123
J. Modelo de datos para el módulo EyeOF 127

Índice de figuras

2.1. Implementación de un detector EMD para el sistema OCTAVE - Extraído de [26] . 6
2.2. Robot portero desarrollado en [44] - Extraído de [44] 7
2.3. Sistema desarrollado en [46]: (a) Sensor de *Flujo Óptico CURVACE* (b) Sensor y plataforma móvil - Extraído de [46] 7

3.1. Segmentación de objetos en movimiento mediante *Flujo Óptico* - Extraído de [56] . 10
3.2. Ejemplo de problema de la apertura 12
3.3. Regiones resultado de un frame de 8×8 píxeles y $\Delta x_{ref} = \Delta y_{ref} = 1$ 16

4.1. Chip ADNS2610 y su pinout 19
4.2. Descripción de los bits en el registro PixelData 20
4.3. Arquitectura del dispositivo 21
4.4. Módulo EyeOF: Renderizado 3D 22
4.5. Esquema eléctrico del módulo EyeOF 23
4.6. Diseño 3D de cada uno de los componentes del módulo y su ensamblaje 24
4.7. Sección del módulo EyeOF y medida de la distancia focal necesaria para cumplir con el estándar de montura tipo C 25
4.8. Módulos EyeOF conectados a la placa STM32L476RG 25
4.9. Adaptación de la interfaz SPI a la interfaz serie del chip ADNS2610 26
4.10. Pinout del microcontrolador STM32L465RGTx 27
4.11. Diagrama de estados de la FSM del módulo EyeOF 29
4.12. Mapa de índices del array unidimensional donde se almacena el frame: En azul los índices en los que algún coeficiente de la operación de *Flujo Óptico* puede ser calculado; en rojo los índices en los que no 32
4.13. Índices tomados en el cálculo de cada coeficiente. Avance del cálculo de coeficientes. 32
4.14. Disposición espacial de los módulos EyeOF en el sistema 33

5.1. Dron DJI Phantom 4 Pro v2.0 de la marca DJI que contiene un gimbal o cardán para estabilizar la cámara 36
5.2. Construcción básica de motores DC brushed (a) y motores DC brushless (b) 37
5.3. Planta de la tarjeta STorM32 BGC 38
5.4. GUI para configuración del controlador STorM32 BGC 39
5.5. Diseño mecánico de la plataforma: modelo 3D del soporte de los módulos 40
5.6. Modelo 3D del sistema y vistas ortogonales con movimientos a $\pm 45°$ 41
5.7. Vista de la plataforma construida 43
5.8. Bucles de control implementados en el sistema 43
5.9. Calibración de la IMU contenida en el módulo MPU 6050 44
5.10. Configuración resultado de seguir el proceso guiado de configuración asistida . . . 45
5.11. Diagrama básico de un controlador PID 46
5.12. Valores de los parámetros de los PIDs ajustados 47

5.13. Configuración de las señales de Radio Control 48
5.14. Respuestas dinámica de la actitud del sistema a una entrada de tipo escalón unitario 49

6.1. MVVM - Bloques funcionales y su relación 52
6.2. Eyes OF Gimbal Platform - vista inicial 53
6.3. Flujo de datos en la clase *WriteableBitmap* de C# 54
6.4. Frame de un rostro capturado por el módulo EyeOF y representado mediante la clase *WriteableBitmap* 54
6.5. Eyes OF Gimbal Platform - vista que contiene todas las funcionalidades posibles de la aplicación 56

Lista de abreviaturas

AC Alternating Current.

API Application Programming Interface.

bps bits per second.

CMOS Complementary Metal-Oxide-Semiconductor.

COM Communication Port.

CPU Central Processing Unit.

CURVACE CURVed Artificial Compound Eyes.

DC Direct Current.

DMA Direct Memory Access.

DSP Digital Signal Processor.

DVS Dynamic Vision Sensor.

EMD Elementary Motion Detector.

FPS Frame Per Second.

FSM Finite State Machine.

GUI Graphical User Interface.

HAL High Abstraction Level.

HAT Hardware Attached on Top.

IDE Integrated Development Enviroment.

IMSE Instituto de Microelectrónica de Sevilla.

IMU Inertial Measurement Unit.

LDO Low Drop-Out.

LL Low Level.

MB MegaByte.

MEMS Micro-ElectroMechanical Systems.

MIPS Millions of Instructions per Processor Second.

MISO Master Input Slave Output.

MOSFET Metal Oxide Semiconductor Field Effect Transistor.

MOSI Master Output Slave Input.

MSB Most Significant Bit.

MVVM Model-View-ViewModel.

OCTAVE Optic flow based Control sysTem for Aerial VEhicles.

PC Personal Computer.

PCB Printed Circuit Board.

PID Proportional-Integral-Derivative (controller).

PWM Pulse Width Modulation.

RC Radio Control.

RPM Revoluciones Por Minuto.

SCK Serial Clock.

SDIO Serial Data Input Output.

SOF Start Of Frame.

SPI Serial Peripheral Interface.

STL Standard Triangle Language.

TPI Threads Per Inch.

UAV Unmanned Aerial Vehicle.

USART Universal Synchronous/Asynchronous Receiver Transmitter.

USB Universal Serial Bus.

WPF Windows Presentation Foundation.

XAML Extensible Application Markup Language.

Capítulo 1

Introducción y motivación

En la naturaleza existen multitud de organismos relativamente simples que realizan tareas rutinarias complejas de manera eficaz y eficiente (volar, nadar, evitar objetos en vuelo, etc.). A lo largo de la historia muchos de los avances que han desembocado en el desarrollo de tecnologías actuales se han basado en la observación de comportamientos biológicos existentes en la naturaleza. Se pueden enumerar distintos ejemplos como la capacidad de volar de las aves y la aeronáutica, el sistema de orientación de los murciélagos y el sistema radar por ultrasonidos, entre otros. Hoy en día, se continúa estudiando los sistemas biológicos como referencia para nuevos diseños, se intenta imitar el comportamiento de los seres vivos mediante su traducción a tecnologías ya desarrolladas, dando lugar a sistemas *"bioinspirados"* [1, 2, 3, 4]. Existen varios ejemplos de esta tendencia como la empresa *Festo®*[5], que en su rama *"Bionics Learning Network"* lleva desarrollando sistemas robóticos imitando el comportamiento de multitud de seres vivos desde 2006; o la iniciativa europea *Human Brain Project* cuyo objetivo es poner en marcha una infraestructura de investigación que permita a los investigadores científicos e industriales avanzar en el conocimiento en los campos de la neurociencia, la informática y la medicina relacionada con el cerebro [6].

La "Inteligencia Artificial", una materia muy popular en la actualidad, trata precisamente de desarrollar un comportamiento basado en el funcionamiento del sistema nervioso humano. Tecnologías como el reconocimiento de voz o el reconocimiento de imágenes tienen como objetivo imitar las funciones que realiza un humano en su vida diaria y poder dotar a una máquina de dichas funciones. El "Machine Learning" [7] o el "Deep Learning" [8] tratan de implementar el mecanismo que posee el cerebro humano para modificar su plasticidad sinápitca en función de los estímulos que procesa, es decir, emular el aprendizaje. Son tecnologías que derivan de la observación y el estudio del comportamiento del cerebro humano.

Uno de los campos de mayor interés es la percepción visual que tienen diferentes seres vivos del mundo real y la interacción entre ellos [9]. Insectos como las hormigas o las abejas realizan tareas de manera coordinada cuya ejecución no les resultaría posible si no existiera dicha coordinación. Por otro lado, sus habilidades de navegación aérea están muy desarrolladas y poseen una alta capacidad de evasión de obstáculos a pesar de que su organismo no es especialmente complejo comparado con otros seres vivos. Esto último, hace que los insectos sean sujetos muy adecuados para el estudio sus capacidades, sus estructuras fisiológicas son más simples y encontrar las relaciones estímulo-acción es más sencillo.

En este trabajo se va a desarrollar el concepto de *"Flujo óptico"* cuyo origen se encuentra precisamente en el estudio de la percepción del mundo real de los seres vivos [10]. El *Flujo óptico* permite a los seres vivos navegar por el espacio tridimensional, tener consciencia de los movimientos que tienen los objetos que les rodea respecto a su punto de vista.

1.1. Motivaciones y objetivos

Muchos autores, que se revisarán en el Capítulo 2, han trabajado en el cómputo del *Flujo Óptico*, desarrollando sistemas que consiguen extraer el *Flujo óptico* del espacio que rodea al sistema. La mayoría de ellos requieren de una capacidad computacional muy elevada lo que limita su implementación a aplicaciones que posean un suministro de energía no limitado. Otros, aunque no necesitan de tal capacidad computacional, sí que hacen uso de un hardware muy específico, y, se centran en caracterizar dicho hardware, no en hacer uso de la información generada. Por otro lado, ambos enfoques aumentan el coste económico del sistema, los sistemas con mayor capacidad computacional son más caros, al igual que la fabricación de hardware muy específico.

En este trabajo se pretende desarrollar un sensor de *Flujo óptico* que no presente requisitos computacionales elevados para su implementación y que se encuentre constituido por componentes comerciales. Una vez desarrollado, utilizar la información de dos de estos sensores acoplados a una plataforma móvil para implementar el seguimiento de objetivos. De esta manera, con tal fin, los objetivos de este Trabajo Fin de Máster fueron los siguientes:

- Estudiar el estado del arte de sistemas relacionados, así como de las distintas opciones disponibles para la implementación de la estimación de *Flujo Óptico*.
- Analizar algunos de los algoritmos extraídos del estado del arte para comprobar cuál es el más adecuado para la aplicación.
- Diseñar, desarrollar y justificar el hardware y el firmware necesarios para obtener un sistema de dos sensores de *Flujo Óptico* operativo.
- Diseñar, modelar y construir la plataforma mecánica que permita el movimiento de los sensores en el espacio tridimensional.
- Estudiar la dinámica de la plataforma desarrollada e implementar el control necesario para su estabilización.
- Diseñar una aplicación de escritorio que permita monitorizar los datos generados por la plataforma construida, así como su ajuste, operación y control.
- Analizar la respuesta dinámica del sistema y, en base a esta información, ajustar los parámetros de control precisos para la aplicación de seguimiento de objetivos.

A tal efecto, este estudio se organiza de la siguiente forma: el Capítulo 2 contiene el estado del arte de sistemas similares, así como de los modelos e implementaciones para la estimación del *Flujo Óptico*. El Capítulo 3 introduce aspectos teóricos del *Flujo Óptico*, su formulación, así como el análisis de algunos de los algoritmos más relevantes. El Capítulo 4 desarrolla el proceso de diseño del hardware y el firmware que forma el sistema constituido por dos sensores de *Flujo Óptico*. Se manda a fabricar el diseño, se sueldan todos sus componentes y se realizan diferentes test para su puesta en marcha. Basados en el diseño de estos módulos se confeccionará una plataforma robótica donde será acoplados los sensores de *Flujo Óptico*. El Capítulo 5 contiene el estudio de los componentes usados en la plataforma, su diseño mecánico y el ajuste de los parámetros de control relacionados a partir del estudio de su dinámica. Se realiza un modelo 3D detallado y a escala de cada elemento del sistema y son dispuestos en el espacio de forma coherente dando lugar a una construcción virtual de la plataforma al completo. Esto permite evitar problemas de colisiones entre elementos en fases de construcción posteriores y ajustar las dimensiones de algunos elementos relevantes para cumplir con algunas restricciones mecánicas necesarias para el correcto funcionamiento del sistema. Varios elementos del sistema precisan de su fabricación para la posterior construcción de la plataforma, para ello se han exportado los modelos 3D correspondientes a un formato compatible

con métodos de fabricación aditiva, como la impresión 3D. Se ha tenido en cuenta la tolerancia del proceso en el diseño de dichos elementos y han sido fabricados en las impresoras 3D disponibles en el IMSE. Tras la construcción del sistema se han realizado tareas de configuración y control automático (estudios de la dinámica del sistema, ajustes de reguladores PID, etc.) con el objetivo de estabilizar la plataforma. Además, se implementan señales de control que permitan modificar la posición de la plataforma según se requiera. Por último, el Capítulo 6 recoge todo el proceso de diseño de la aplicación de escritorio para la monitorización, ajuste y control de la plataforma. Se explica la tecnología utilizada así como las herramientas usadas para la representación de los datos.

En el Capítulo 7 se hace una revisión de los objetivos alcanzados así como de diferentes líneas de trabajo futuras originadas durante el desarrollo del estudio.

Capítulo 2

Estado del arte

La detección de movimiento a través del estudio de una secuencia de imágenes es un problema que se lleva estudiando desde hace décadas. Ya en el siglo XIX el científico von Helmholtz [11] comenzó a investigar sobre la idea de la detección de movimiento a través de la información contenida en una imagen, aunque no aportaba un contenido teórico. No fue hasta la década de 1940, cuando el psicólogo norteamericano Gibson [10] en su trabajo sobre la *"Percepción Visual del mundo real en seres vivos"* planteó este problema acuñando el concepto de *"Flujo Óptico"*. Aunque Gibson [10] no poseía las herramientas necesarias para resolver el problema, dio lugar a muchos estudios que buscaban dar al concepto un sentido matemático y teórico [12, 13, 14, 15, 16]. Otros investigadores se centraron en el sentido biológico, es decir, en el estudio de las estructuras que permitían a los seres vivos detectar el *Flujo Óptico* [17, 18, 19, 20, 21].

El conocimiento aportado por estos trabajos derivó en estudios donde el objetivo era la estimación del *Flujo Óptico* mediante la implementación de estructuras que simularan el comportamiento biológico de las estructuras utilizadas por los seres vivos para este fin (sistemas *"bioinspirados"*) o, por otro lado, la implementación de los modelos teóricos en sistemas computacionales. El trabajo de Srinivasan [22] estudia la detección de *Flujo Óptico* en los insectos, cómo utilizan esta información para navegar por el espacio del mundo real, donde coexisten una serie de factores como obstáculos y depredadores, y desarrolla una serie de aplicaciones robóticas basándose en el sistema de percepción de los insectos. En Srinivasan [23] y Baird et al. [24] se estudia un modelo de robot basado en el comportamiento de las abejas en tareas de navegación, en particular de la especie *Apis mellifera L.* Franceschini et al. [25] expusieron el papel de la percepción de *Flujo Óptico* en las tareas de despegue, estabilización de vuelo y aterrizaje en insectos voladores. Estas tareas para los insectos son cotidianas y las realizan sistemas de consumo muy bajo y con muy baja latencia, de ahí el atractivo de intentar basar la operación de sistemas robóticos en los mismos principios. En Ruffier and Franceschini [26] se desarrolla *OCTAVE ("Optic flow based Control sysTem for Aerial VEhicles")*, un piloto automático para aeronaves basado en el *Flujo Óptico* detectado mediante un circuito bioinspirado denominado *EMD (Elementary Motion Detector)* [27] (Figura 2.1) resultado del estudio de grabaciones electrofisiológicas al aplicar microestímulos a un único fotorreceptor del ojo de un insecto.

Con la llegada de los sistemas computacionales modernos, se comienza a estudiar la posibilidad de implementar mediante algoritmos computacionales modelos teóricos de estimación de *Flujo Óptico*. En Barron et al. [28] y Baker et al. [29] se muestra una comparación de los resultados obtenidos por diferentes implementaciones de modelos teóricos de *Flujo Óptico*. La base de la mayoría de los algoritmos son los modelos de Horn and Schunck [13] y Lucas and Kanade [15], aunque cada autor ha realizado diferentes optimizaciones que hacen que el modelo presente un rendimiento optimizado así como resultados más precisos. Por ejemplo, algoritmos propuestos por Murray and Buxton [30], Black and Anandan [31] y Brox et al. [32] implementan el modelo de Horn

Figura 2.1: Implementación de un detector EMD para el sistema OCTAVE- Extraído de [26]

and Schunck [13] añadiéndole filtros espacio-temporales, entre otras optimizaciones, que mejoran la precisión del *Flujo Óptico* resultante. Otros algoritmos como los propuestos por Jung et al. [33] y Lempitsky et al. [34] utilizan la fusión de los dos modelos anteriores. A pesar de las múltiples optimizaciones realizadas en el tiempo, aunque la precisión de los resultados ha mejorado sustancialmente, los algoritmos de *Flujo Óptico* son muy demandantes computacionalmente y necesitan de máquinas potentes para ejecutarlos en imágenes con una cadencia propias del tiempo real.

Estos algoritmos han conducido a la aplicación del *Flujo Óptico* en diferentes ámbitos como son el diagnóstico médico [35, 36, 37], la vigilancia [38, 39], la producción de contenido televisivo [40], el seguimiento de objetivos [41, 42], prevención de incendios [43], entre otros. Cada aplicación utiliza el algoritmo que mejor se comporte frente a las características de las imágenes capturadas: ruido, rango dinámico, etc. Muchas de las aplicaciones necesitan de un procesamiento previo de las imágenes antes de aplicar el algoritmo en cuestión, lo que aumenta los requisitos de potencia computacional.

En este trabajo el *Flujo Óptico* es utilizado para el seguimiento de objetivos en el espacio tridimensional abordando no sólo el diseño del sensor de *Flujo Óptico* sino que también se diseña la plataforma mecánica que permitirá posicionar correctamente los sensores así como la implementación de su control. Este tipo de sistemas han sido implementados por otros autores como, por ejemplo, en Delbruck and Lang [44] se desarrolla un robot portero, mostrado en la Figura 2.2, que detecta las bolas lanzadas hacia él y las intercepta. Aunque en este caso no se utilizan algoritmos de *Flujo Óptico*, puesto que se usa un sensor DVS [45] que devuelve los píxeles de la imagen que presentan movimiento, el concepto de controlar la plataforma a partir de la información visual es parecido al que se acoge en este trabajo. La respuesta de la plataforma es muy rápida y el consumo de CPU se encuentra por debajo del 4 % pero, ha de anotarse la existencia de un dispositivo de alta capacidad computacional como es un PC. Además, la plataforma solo presenta un grado de libertad en el eje horizontal.

En Pericet-Camara et al. [46] se construye una plataforma que permite caracterizar el sensor de *Flujo Óptico* bioinspirado desarrollado en el estudio denominado *CURVed Artificial Compound Eyes (CURVACE)* [47]. El sensor *CURVACE* contiene dos microcontroladores de 16-bits de la familia *dsPIC33F*[48] con una capacidad de 40 MIPS. La plataforma permite un grado de libertad en el eje horizontal para realizar mediciones de *Flujo Óptico* en condiciones de rotación conocidas y, así, poder caracterizar el dispositivo. En la referencia se estudia el comportamiento de dos algoritmos: una versión extendida de Lucas and Kanade [15] y el algoritmo de Srinivasan [49]. Este último algoritmo es el que se va a desarrollar e implementar en este trabajo.

La presencia de elementos basados en la estimación de *Flujo Óptico* en dispositivos robóticos, satélites, etc. actuales es cada vez más frecuente debido al aumento de sus prestaciones, la fiabilidad que ofrecen y la optimización de los recursos computacionales requeridos.

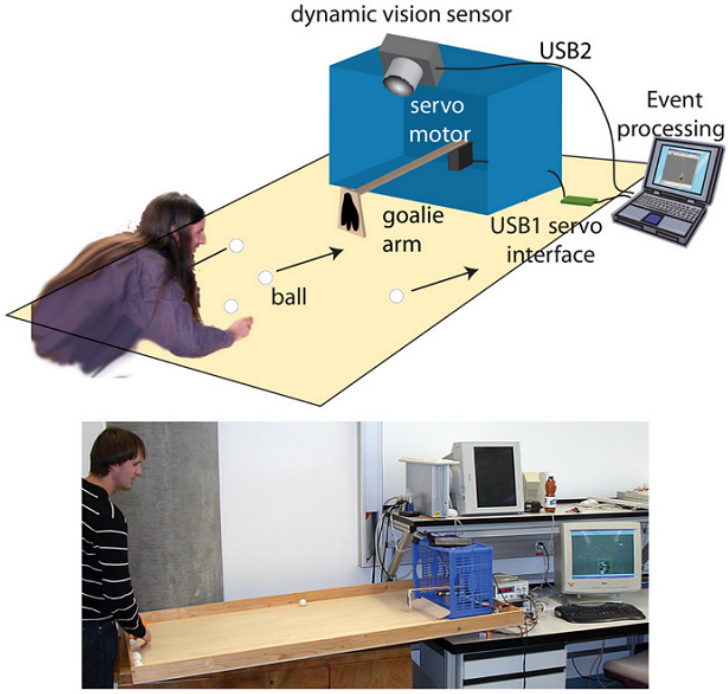

Figura 2.2: Robot portero desarrollado en [44] - Extraído de [44]

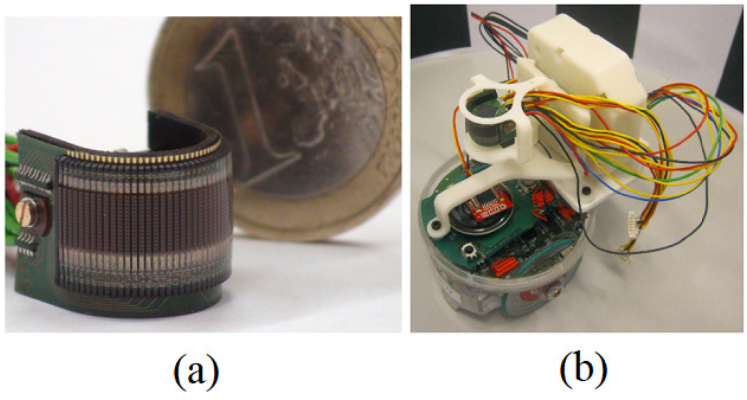

Figura 2.3: Sistema desarrollado en [46]: (a) Sensor de *Flujo Óptico CURVACE* (b) Sensor y plataforma móvil - Extraído de [46]

Capítulo 3

Flujo óptico

El *Flujo Óptico* es la distribución del movimiento aparente de patrones de luminosidad en un frame[1] [13], suponiendo que los cambios en la luminosidad del frame se deben únicamente al movimiento de elementos en el frame [50]. Se trata de un movimiento relativo entre distintos objetos y un observador [50, 10, 51], por tanto, esta magnitud puede aportar información muy valiosa a la hora de localizar y monitorizar la situación de los objetos en un espacio a través de una secuencia de frames separados en el tiempo de dicho espacio [52].

Por otro lado, las discontinuidades en el *Flujo Óptico* permiten realizar tareas de segmentación del frame [53, 54, 55]. La segmentación se basa en la detección de discontinuidades en el *Flujo Óptico* de forma que las zonas donde el *Flujo Óptico* es continuo se relacionan con un mismo objeto en la misma. En la Figura 3.1 se muestra un ejemplo de segmentación mediante *Flujo Óptico*, el bateador es detectado debido a que es la única zona donde existe *Flujo Óptico* puesto que el fondo del frame permanece quieto [56].

La estimación del *Flujo Óptico* es un problema ampliamente analizado por la comunidad científica. A pesar del gran número de modelos y algoritmos diferentes, ninguno de estos cubre algunos problemas que aparecen en implementaciones de procesamiento en tiempo real para condiciones reales [57]. El cálculo de *Flujo Óptico* se basa en dos pasos fundamentalmente [50]:

- calcular las derivadas espacio-temporales, o lo que es lo mismo, medir las velocidades normales a estructuras locales de intensidad luminosa.
- combinar las velocidades normales para llegar a las velocidades totales, mediante, por ejemplo, minimización de mínimos cuadrados [15, 49] o regularización [13].

En los apartados que siguen se va a desarrollar la teoría que existe detrás del *Flujo Óptico*, algunos algoritmos que implementan el cálculo de *Flujo Óptico* y sus ventajas e inconvenientes, así como la elección del algoritmo que se ha hecho en este trabajo y su justificación.

3.1. Teoría del cálculo del *Flujo Óptico*

La base para el cálculo de *Flujo Óptico* es la ecuación de restricción de movimiento [50]. Dada la intensidad de un pixel en el centro de un grupo de $n \times n$ pixeles por la función $I(x, y, t)$, si éste se mueve una cantidad δx y δy en el plano cartesiano en un tiempo δt, pasa a venir definido por

[1] Un frame es una matriz bidimensional cuyos elementos (píxeles) representan el valor de la luminancia en una región proporcional a su extensión dentro de la escena visual captada por una cámara.

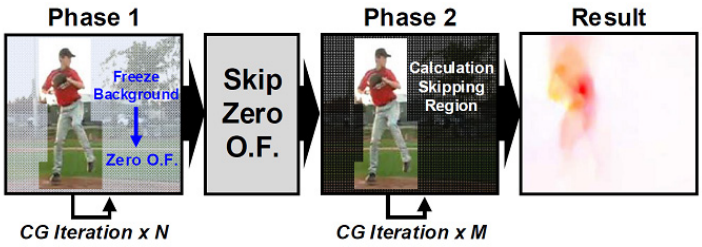

Figura 3.1: Segmentación de objetos en movimiento mediante *Flujo Óptico* - Extraído de [56]

la función $I(x+\delta x, y+\delta y, t+\delta t)$. Puesto que $I(x,y,t)$ y $I(x+\delta x, y+\delta y, t+\delta t)$ definen el mismo punto en el frame, la Ecuación 3.1 debe cumplirse. Esta suposición se cumple para aproximaciones de primer orden, es decir, el movimiento del pixel es local a su vecindad (δx, δy y δt son valores pequeños), lo que hace posible calcular $I(x+\delta x, y+\delta y, t+\delta t)$ a partir de $I(x,y,t)$ según los términos de menor orden de la serie de Taylor de $I(x,y,t)$ como se muestra en la Ecuación 3.2, donde $\nabla I(x,y,t)$ es el gradiente dado por la Ecuación 3.3.

$$I(x,y,t) = I(x+\delta x, y+\delta y, t+\delta t) \tag{3.1}$$

$$I(x+\delta x, y+\delta y, t+\delta t) = I(x,y,t) + \nabla I(x,y,t) \cdot (\delta x, \delta y, \delta t)^T + ... \tag{3.2}$$

$$\nabla I(x,y,t) = \left(\frac{\partial I(x,y,t)}{\partial x}, \frac{\partial I(x,y,t)}{\partial y}, \frac{\partial I(x,y,t)}{\partial t} \right) \tag{3.3}$$

La Ecuación 3.1 y los términos de primer orden de la Ecuación 3.2 permiten obtener la expresión desarrollada en 3.4. Es posible identificar que $\frac{\delta x}{\delta t}$ y $\frac{\delta y}{\delta t}$ son términos relacionados con el movimiento en las componentes x e y del plano cartesiano. Estos términos son el valor de *Flujo Óptico* para el pixel estudiado.

$$\begin{aligned}
\nabla I(x,y,t) \cdot (\delta x, \delta y, \delta t)^T &= 0 \\
\left(\frac{\partial I(x,y,t)}{\partial x}, \frac{\partial I(x,y,t)}{\partial y}, \frac{\partial I(x,y,t)}{\partial t} \right) \cdot (\delta x, \delta y, \delta t)^T &= 0 \\
\frac{\partial I(x,y,t)}{\partial x} \cdot \frac{\delta x}{\delta t} + \frac{\partial I(x,y,t)}{\partial y} \cdot \frac{\delta y}{\delta t} + \frac{\partial I(x,y,t)}{\partial t} \cdot \frac{\delta t}{\delta t} &= 0 \\
\frac{\partial I(x,y,t)}{\partial x} \cdot \frac{\delta x}{\delta t} + \frac{\partial I(x,y,t)}{\partial y} \cdot \frac{\delta y}{\delta t} + \frac{\partial I(x,y,t)}{\partial t} &= 0
\end{aligned} \tag{3.4}$$

Para llegar a una expresión simplificada, se denotan distintos términos según las Ecuaciones 3.5 y 3.6. El resultado es la Ecuación 3.7 y la Ecuación 3.8 [2] donde $\nabla I_{x,y}$ es el gradiente espacial de intensidad y $\vec{v}$ el vector de *Flujo Óptico* para la coordenada (x,y) del plano cartesiano del frame en el instante t.

[2] Siempre que se indique un punto entre vectores se refiere al producto escalar de los vectores

$$I_x = \frac{\partial I(x,y,t)}{\partial x}, \quad I_y = \frac{\partial I(x,y,t)}{\partial y}, \quad I_t = \frac{\partial I(x,y,t)}{\partial t} \tag{3.5}$$

$$v_x = \frac{\partial x}{\partial t}, \quad v_y = \frac{\partial y}{\partial t} \tag{3.6}$$

$$(I_x, I_y) \cdot (v_x, v_y)^T + I_t = 0 \tag{3.7}$$

$$\nabla \vec{I}_{x,y} \cdot \vec{v} = -I_t \tag{3.8}$$

La Ecuación 3.8 es denominada la ecuación de restricción del movimiento en 2D. En esta ecuación se tienen 2 incógnitas (v_x, v_y) que no pueden ser determinadas debido a que solo se tiene la Ecuación 3.8. La causa de esta indeterminación se define como el problema de la apertura.

El Problema de la Apertura

El problema de la apertura define que normalmente una estructura de pixeles local en un frame no es suficiente para conocer el movimiento global de todo el frame (v_x, v_y). Sin embargo, sí que es suficiente para calcular la componente normal del movimiento v_n de la estructura de pixeles local, mientras que la componente tangencial v_t no se puede obtener. En la Figura 3.2 se muestra un ejemplo que muestra el problema gráficamente:

- Se tiene una región cuyos pixeles tienen el mismo valor en toda la región (rectángulo verde). La región se está moviendo como indica el vector de su centro.
- Si se estudia el frame completo según el contenido de una región local, señalada con un círculo, es trivial observar que solo es posible determinar el movimiento normal al borde de la región, mientras que el movimiento tangencial no.

En estas circunstancias, solo es posible obtener la componente normal de la velocidad $\vec{v_n}$. Esta componente vendrá dada por la dirección del gradiente de intensidades $\nabla I_{x,y}$. Si se tiene en cuenta que todos los valores posibles del vector velocidad $\vec{v}$ se encuentran en la recta definida por la Ecuación 3.8, la dirección de la componente normal y su módulo se pueden obtener mediante la dirección normal a dicha recta y el módulo del vector al punto de intersección con la recta.

En la Ecuación 3.9 se desarrolla el producto escalar de la Ecuación 3.8. Para un valor de θ nulo se obtiene el módulo de la componente normal v_n (Ecuación 3.10). El vector de dirección unitario viene dado por $\nabla I_{x,y}$ como se comenta anteriormente (Ecuación 3.11). Finalmente, la componente normal queda definida en la Ecuación 3.12

$$\|\vec{v}\| = \frac{-I_t}{\left\|\nabla \vec{I}_{x,y}\right\| \cdot \cos\theta} \tag{3.9}$$

$$\begin{gathered} \text{mín}\left(\|\vec{v}\|\right) \quad \rightarrow \quad \theta = 0 \quad \rightarrow \quad \text{mín}\left(\|\vec{v}\|\right) = \frac{-I_t}{\left\|\nabla \vec{I}_{x,y}\right\|} \\ v_n = \frac{-I_t}{\|\nabla \vec{I}_{x,y}\|} \end{gathered} \tag{3.10}$$

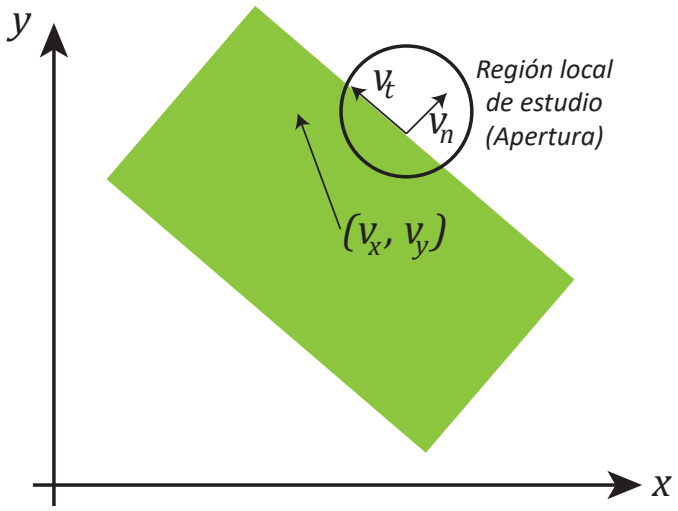

Figura 3.2: Ejemplo de problema de la apertura

$$\vec{n} = \frac{(I_x, I_y)}{\left\|\nabla \vec{I}_{x,y}\right\|} \tag{3.11}$$

$$\vec{v_n} = v_n \cdot \vec{n} \tag{3.12}$$

Asumiendo esta restricción se puede volver a definir la Ecuación 3.8 en función de $\vec{v_n}$ y obtener la Ecuación 3.13. La expresión resultante queda definida en función de I_x, I_y e I_t, que son valores conocidos.

$$\vec{v} \cdot \vec{n} = v_n \tag{3.13}$$

3.2. Algoritmos de cálculo de *Flujo Óptico*

Basados en la teoría anterior, varios autores han desarrollado algoritmos que implementan el cálculo de *Flujo Óptico* siguiendo procedimientos distintos. Principalmente se pueden dividir en cuatro enfoques distintos para abordar el problema [58]:

- **Técnicas basadas en la detección de características:** Se detectan distintas características del frame como bordes, objetos, etc. mediante técnicas de segmentación y se estudian su movimiento en la secuencia de frames para conocer el movimiento global del frame. El resultado presenta precisión y exactitud, pero la carga computacional es alta debido a las tareas de segmentación. [59]
- **Técnicas diferenciales:** Se basan en el cálculo de derivadas espacio-temporales de las intensidades del frame o de versiones filtradas del frame. Estos gradientes suelen ser calculados localmente y combinados globalmente para obtener el *Flujo Óptico*. Este técnica se utiliza mucho en implementaciones de cálculo de *Flujo Óptico* mediante redes neuronales. [15, 60, 61]

- **Técnicas basadas en la energía del frame:** Se basan en la comparación de las energías [62] de la intensidad en el frame en varias zonas del espectro de frecuencias electromagnético mediante el uso de un conjunto de filtros espacio-temporales que abarcan todo el espectro. Otros algoritmos no utilizan la respuesta en frecuencias, utilizan la respuesta en fase. [63, 64, 65]
- **Técnicas de correspondencia regional:** El cálculo de *Flujo Óptico* se basa en encontrar un vector desplazamiento entre dos frames consecutivos que minimice el error entre los frames. Este tipo de técnicas se comportan mejor que las anteriores con frames reales puesto que son más robustas al ruido y al aliasing en el frame. [49, 66]

Entre los algoritmos más destacados se tienen a autores como Lucas and Kanade [15], Horn and Schunck [60], Anandan [67] o Srinivasan [49]. Se van a presentar algunos de ellos, así como sus principales características y diferencias según Barron et al. [28]. Finalmente se desarrollará en detalle el algoritmo que se va a utilizar en este trabajo para implementar el cálculo de *Flujo Óptico* (Srinivasan [49]).

3.2.1. Algoritmo de Lucas y Kanade

El algoritmo de Lucas and Kanade [15] calcula el *Flujo Óptico* mediante la minimización dada en la Ecuación 3.14, donde se implementa el modelo de restricción de la Ecuación 3.8 y se realiza un ajuste en mínimos cuadrados sobre una región local del frame dada por Φ.

$$\sum_{x,y\in\Phi}\left\{W^2(x,y)\cdot\left[\nabla I(x,y,t)\cdot(v_x,v_y)^T+I_t(x,y,t)\right]^2\right\} \tag{3.14}$$

$W(x,y)$ es una función bidimensional que permite asignar grados de relevancia a los píxeles de la región Φ. Suele ser implementada mediante una función Gaussiana 2D. El valor de $\vec{v}$ viene dado según la Ecuación 3.14 en la Ecuación 3.15.

$$\vec{v}=\left[A^TW_m^2A\right]^{-1}A^TW_m^2\vec{r} \tag{3.15}$$

donde, para una región de $n\times n$ píxeles ($Z=n^2$) , $(x_j,y_j)\in\Phi$ en un instante de tiempo determinado se tiene que:

$$A=[\nabla I(x_1,y_1),\cdots,\nabla I(x_Z,y_Z)]$$

$$W_m=\begin{bmatrix}W(x_1,y_1) & & \\ & \ddots & \\ & & W(x_Z,y_Z)\end{bmatrix} \tag{3.16}$$

$$\vec{r}=-\left[I_t(x_1,y_1),\cdots,I_t(x_Z,y_Z)\right]$$

Para abordar el problema de la apertura (Sección 3.1) se obtienen los autovalores (α_1,α_2) y sus correspondientes autovectores $(\vec{b_1},\vec{b_2})$ de la matriz $\left[A^TW^2A\right]$, siendo $\alpha_1\geq\alpha_2\geq 0$, se estudian los siguientes casos:

- Si α_1 y α_2 es mayor a un umbral ajustable ρ (normalmente igual a la unidad), $\vec{v}$ se calcula mediante la Ecuación 3.15.

- Si α_1 es menor a ρ pero α_2 es mayor a ρ, $\vec{v_n}$ se calcula proyectando $\vec{v}$ en la dirección $\vec{b_2}$ (Ecuación 3.17).

$$\vec{v_n} = (\vec{v} \cdot \vec{b_2}) \cdot \vec{b_2} \tag{3.17}$$

Las características principales de este algoritmo son las que siguen:

- Se necesita del almacenamiento de un conjunto de frames (normalmente mayor a 5 frames) que aumentan los requisitos de almacenamiento para su implementación. Además, el cálculo de *Flujo Óptico* se encuentra retrasado dicho número de frames.
- Utiliza filtros paso bajo y paso alto para implementar derivadas espacio-temporales de forma que su resultado sea correcto. Este tipo de cálculos son muy costosos computacionalmente.
- El valor de *Flujo Óptico* calculado tiene un grado de fiabilidad bastante alto.

3.2.2. Algoritmo de Horn y Schunck

El algoritmo de Horn and Schunck [60] combina la ecuación de restricción de movimiento (Ecuación 3.8) con un término de suavizado del frame para estimar el *Flujo Óptico* mediante la minimización de la Ecuación 3.18. El proceso de minimización se realiza mediante un método iterativo que resulta en valores para $\vec{v}$ dados por la Ecuación 3.19 y la Ecuación 3.20.

$$\sum_{x,y \in \Theta} \left\{ (\vec{\nabla} I \cdot \vec{v} + I_t)^2 + \lambda^2 \left[\left(\frac{\partial v_x}{\partial x} \right)^2 + \left(\frac{\partial v_x}{\partial y} \right)^2 + \left(\frac{\partial v_y}{\partial x} \right)^2 + \left(\frac{\partial v_y}{\partial y} \right)^2 \right] \right\} \tag{3.18}$$

$$v_x^{i+1} = (\overline{v_x})^i - \frac{I_x \left[I_x (\overline{v_x})^i + I_y (\overline{v_y})^i + I_t \right]}{\lambda^2 + I_x^2 + I_y^2} \tag{3.19}$$

$$v_y^{i+1} = (\overline{v_y})^i - \frac{I_y \left[I_x (\overline{v_x})^i + I_y (\overline{v_y})^i + I_t \right]}{\lambda^2 + I_x^2 + I_y^2} \tag{3.20}$$

donde λ permite ajustar el efecto del término de suavizado sobre el frame; i es el número de iteración; v_x^0 y v_y^0 es el valor de velocidad inicial que se suele resetear a cero; y $(\overline{v_x})^i$ y $(\overline{v_y})^i$ es la media de la vecindad de v_x^i y v_y^i, respectivamente.

Las características principales de este algoritmo son las que siguen:

- Sigue un proceso iterativo que disminuye las operaciones necesarias una vez el algoritmo alcance su funcionamiento estable.
- El *Flujo Óptico* calculado está retrasado respecto al frame capturado en el instante actual.
- Necesita de filtros paso bajas y paso altas para calcular las derivadas espacio-temporales sobre el frame de forma correcta.
- El valor de *Flujo Óptico* calculado tiene un grado de fiabilidad bastante alto si se utilizan un número de iteraciones alto (en torno a 100).

3.3. Algoritmo de interpolación de Srinivasan

El algoritmo de Srinivasan [49] se basa en la suposición de que los cambios entre dos frames consecutivos en una secuencia de frames separados por un corto período de tiempo son lineales. De esta forma, el movimiento de un frame respecto al anterior, es decir, el *Flujo Óptico*, se puede calcular mediante técnicas de interpolación. Este algoritmo presenta varias ventajas muy interesantes respecto a los demás:

- No necesita de la detección de características en el frame.
- Solo necesita de una iteración para conseguir un resultado de *Flujo Óptico*.
- No requiere de filtros temporales o espaciales sobre el frame. Estas operaciones son muy costosas computacionalmente.
- El resultado es robusto al ruido en el frame.

Imagínese un plano que tiene libertad de traslación y está siendo capturado por una cámara que devuelve un patrón de intensidades cuyo centro será considerado el centro cartesiano $(0, 0)$ de un sistema de coordenadas. Se pretende estudiar el movimiento del plano mediante el estudio de una región del frame capturado definida por una función ventana $\Psi(x, y)$. La función $\Psi(x, y)$ puede ser una función Gaussiana bidimensional si se desea implementar la visión foveal [68, 69] o uniforme bidimensional si simplemente se desea seleccionar una región del frame.

Con esta configuración se almacena dos frames consecutivos de una secuencia de frames en los instantes t_0 y t. En cada coordenada de la región definida por $\Psi(x, y)$ la intensidad del pixel vendrá dada por la función $f_0(x, y)$ y $f(x, y)$, respectivamente. El movimiento del plano que se produce desde el instante t_0 a t se va a expresar en relación a cantidades discretas de píxeles $\widehat{\Delta x}$ y $\widehat{\Delta y}$ de los frames capturados. En este momento se definen cuatro frames f_1, f_2, f_3 y f_4 [3] como sigue:

- f_1 y f_2 son el resultado de desplazar la ventana $\Psi(x, y)$ una cantidad Δx_{ref} de referencia en el sentido positivo y negativo de x, respectivamente (Ecuación 3.21).
- f_3 y f_4 son el resultado de desplazar la ventana $\Psi(x, y)$ una cantidad Δy_{ref} de referencia en el sentido positivo y negativo de y, respectivamente (Ecuación 3.22).

$$\begin{aligned} f_1(x, y) &= f_0(x + \Delta x_{ref}, y) \\ f_2(x, y) &= f_0(x - \Delta x_{ref}, y) \end{aligned} \tag{3.21}$$

$$\begin{aligned} f_3(x, y) &= f_0(x, y + \Delta y_{ref}) \\ f_4(x, y) &= f_0(x, y - \Delta y_{ref}) \end{aligned} \tag{3.22}$$

A efectos ilustrativos en la Figura 3.3 se muestra una representación gráfica de las distintas regiones derivadas de un frame de 8×8 pixeles y un valor de Δx_{ref} y Δy_{ref} igual a la unidad.

Como se comenta anteriormente, para desplazamientos relativos de pequeña magnitud[4] se considera pequeña magnitud se supone que el cambio de intensidades entre los frames es lineal desde f_0 a f, y que, por tanto, es posible obtener f a partir de f_0 y los frames de referencia f_1, f_2, f_3 y f_4. La Ecuación 3.23 muestra la operación de interpolación realizada para obtener $\widehat{f}$, que es una

[3] Señalar que f_1, f_2, f_3 y f_4 son funciones bidimensionales en (x, y), es decir, $f_1(x, y)$, $f_2(x, y)$, $f_3(x, y)$ y $f_4(x, y)$. Se ha omitido (x, y) para mayor claridad del texto.

[4] El desplazamiento se compara con la longitud de onda de la mayor componente en frecuencia del frame para establecer si se considera que es de pequeña magnitud y, por tanto, que es válida la operación.

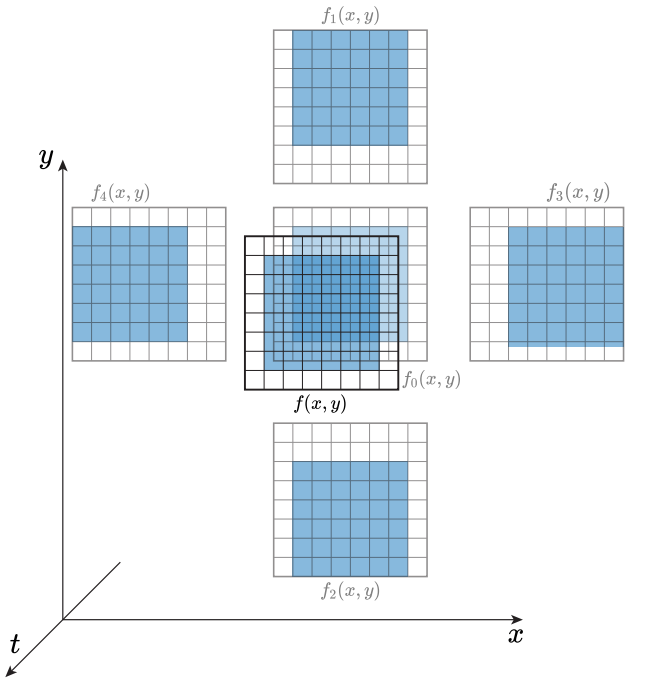

Figura 3.3: Regiones resultado de un frame de 8×8 píxeles y $\Delta x_{ref} = \Delta y_{ref} = 1$

versión estimada mediante interpolación de la región f. El problema ahora consiste en obtener los valores $\widehat{\Delta x}$ y $\widehat{\Delta y}$ que minimicen el error entre f y $\widehat{f}$.

$$\widehat{f} = f_0 + 0{,}5\left(\frac{\widehat{\Delta x}}{\Delta x_{ref}}\right)(f_2 - f_1) + 0{,}5\left(\frac{\widehat{\Delta y}}{\Delta y_{ref}}\right)(f_4 - f_3) \tag{3.23}$$

La minimización del error entre f y $\widehat{f}$ se realiza mediante la optimización de la expresión del error cuadrático medio estudiado sobre la función ventana $\Psi(x,y)$ que define la región de estudio en el frame. En la Ecuación 3.24 se muestra la expresión del error cuadrático medio para un frame de $M \times N$ píxeles.

$$E = \sum_{x=0}^{M}\sum_{y=0}^{N}\left\{\Psi(x,y)\cdot\left[f(x,y) - \widehat{f}(x,y)\right]^2\right\} \tag{3.24}$$

Si se sustituye la Ecuación 3.23 en la Ecuación 3.24 se tiene la Ecuación 3.25. Para su minimización se obtienen las derivadas parciales respecto a $\widehat{\Delta x}$ y $\widehat{\Delta y}$ y se igualan a cero, obteniéndose el sistema de ecuaciones formado por la Ecuación 3.26 y la Ecuación 3.27.

$$E = \sum_{x=0}^{M}\sum_{y=0}^{N}\left\{\Psi\cdot\left\{f - \left[f_0 + 0{,}5\left(\frac{\widehat{\Delta x}}{\Delta x_{ref}}\right)(f_2 - f_1) + 0{,}5\left(\frac{\widehat{\Delta y}}{\Delta y_{ref}}\right)(f_4 - f_3)\right]\right\}^2\right\} \tag{3.25}$$

$$\left(\frac{\widehat{\Delta x}}{\Delta x_{ref}}\right)\cdot\sum_{x=0}^{M}\sum_{y=0}^{N}\left[\Psi\cdot(f_2-f_1)^2\right] + \left(\frac{\widehat{\Delta y}}{\Delta y_{ref}}\right)\cdot\sum_{x=0}^{M}\sum_{y=0}^{N}\left[\Psi\cdot(f_4-f_3)\cdot(f_2-f_1)\right] = 2\cdot\sum_{x=0}^{M}\sum_{y=0}^{N}\left[\Psi\cdot(f-f_0)\cdot(f_2-f_1)\right] \quad (3.26)$$

$$\left(\frac{\widehat{\Delta x}}{\Delta x_{ref}}\right)\cdot\sum_{x=0}^{M}\sum_{y=0}^{N}\left[\Psi\cdot(f_2-f_1)\cdot(f_4-f_3)\right] + \left(\frac{\widehat{\Delta y}}{\Delta y_{ref}}\right)\cdot\sum_{x=0}^{M}\sum_{y=0}^{N}\left[\Psi\cdot(f_4-f_3)^2\right] = 2\cdot\sum_{x=0}^{M}\sum_{y=0}^{N}\left[\Psi\cdot(f-f_0)\cdot(f_4-f_3)\right] \quad (3.27)$$

El valor de $\widehat{\Delta x}$ y $\widehat{\Delta y}$ resultado de la resolución del sistema de ecuaciones (Ecuación 3.28 y Ecuación 3.29) es el valor del *Flujo Óptico* para la región $\Psi(x,y)$.

$$\widehat{\Delta x} = \frac{C\cdot D - B\cdot E}{A\cdot D - B^2} \quad (3.28)$$

$$\widehat{\Delta y} = \frac{A\cdot E - B\cdot C}{A\cdot D - B^2} \quad (3.29)$$

donde

$$A = \sum_{x=0}^{M}\sum_{y=0}^{N}\left[\Psi\cdot(f_2-f_1)^2\right]$$

$$B = \sum_{x=0}^{M}\sum_{y=0}^{N}\left[\Psi\cdot(f_2-f_1)\cdot(f_4-f_3)\right]$$

$$C = \sum_{x=0}^{M}\sum_{y=0}^{N}\left[\Psi\cdot(f-f_0)\cdot(f_2-f_1)\right]$$

$$D = \sum_{x=0}^{M}\sum_{y=0}^{N}\left[\Psi\cdot(f_4-f_3)^2\right]$$

$$E = \sum_{x=0}^{M}\sum_{y=0}^{N}\left[\Psi\cdot(f-f_0)\cdot(f_4-f_3)\right]$$

Existen varias situaciones donde el sistema resulta ser indeterminado:

- El frame es una estructura homogénea donde no existe cambios en la intensidad de los píxeles. En este caso todos los coeficientes de la solución son nulos debido a que $f_1 = f_2 = f_3 = f_4$.
- El frame solo contiene características en una dimensión, por ejemplo, una línea horizontal o una línea vertical. En el caso de características horizontales $f_2 - f_1$ es nulo, mientras que en el caso de características verticales $f_4 - f_3$ es nulo. En estos dos casos las funciones dejarían de ser independientes y no existiría una solución única. Este problema es el denominado Problema de la Apertura que se comenta en la Sección 3.1.

Estas situaciones se pueden detectar monitorizando los valores de los distintos coeficientes y así evitar resultados erróneos. Además, no suelen darse debido a la existencia de ruido en frames reales.

En este trabajo se va a utilizar una ventana $\Psi(x, y)$ uniforme que servirá simplemente para seleccionar una región dentro del frame.

Capítulo 4

Diseño hardware y firmware para la adquisición del frame y el cálculo del *Flujo Óptico*

Durante este capítulo se va a desarrollar la fase de diseño hardware y firmware necesarios para la adquisición del frame y el cálculo de *Flujo Óptico*. En primer lugar, se va a presentar el sensor de imagen de baja resolución que se va a usar; en función de las capacidades de este sensor se discutirá el diseño hardware necesario para su adaptación al sistema, así como la elección del hardware para la lectura del sensor. En segundo lugar, se presentará en detalle el firmware desarrollado para implementar la aplicación, así como las distintas decisiones tomadas en el proceso.

4.1. Sensor óptico ADNS2610

Los sensores de la familia ADNS son sensores ópticos orientados a su implementación en ratones ópticos utilizados como periféricos en PCs. Este dispositivo facilita un método que no precisa de elementos mecánicos para mover el puntero en la pantalla del PC de acuerdo con el movimiento del sensor integrado en el ratón de PC. El chip, mostrado en la Figura 4.1, está constituido fundamentalmente por un sensor de imagen CMOS de 18×18 píxeles y un DSP. Mediante el sensor CMOS se capta la imagen exterior al sensor, que en la aplicación particular de un ratón de PC será la superficie sobre la que se desliza el ratón, y en el DSP se procesa el frame adquirido para obtener las características del movimiento (magnitud, dirección y sentido) que se desea [70].

Figura 4.1: Chip ADNS2610 y su pinout

Pixel Data Address: 0x08
Access: Read/Write Reset Value: 0x00

Bit	7	6	5	4	3	2	1	0
Field	SOF	Data_Valid	PD_5	PD_4	PD_3	PD_2	PD_1	PD_0

Figura 4.2: Descripción de los bits en el registro PixelData

Todos los parámetros de configuración del sensor, el frame adquirido, así como las medidas realizadas se encuentran disponibles en los distintos registros internos del chip a través de una interfaz serie de dos hilos. Dicha interfaz está compuesta por una señal de reloj `SCK` que sirve de sincronismo durante la comunicación y una señal de dato `SDIO` mediante la cual se transmiten los datos en serie.

De especial interés son los registros `Delta_X (0x03)` y `Delta_Y (0x04)` que contienen valores relacionados con el movimiento en el eje X y el eje Y. Estos datos son el resultado de un algoritmo interno implementado en el DSP para el cálculo de *Flujo Óptico* y no serán usados ya que el interés de este trabajo reside en implementar el algoritmo de interpolación presentado anteriormente.

Por otro lado, la lectura del frame se realiza mediante la lectura sucesiva del registro `PixelData (0x08)` hasta recorrer todos los pixeles del frame. Todas las transacciones de lectura y escritura con el chip son de 8 bits. El registro `PixelData` devuelve el valor del pixel medido en sus 6 bits menos significativos `PD<5:0>`, los dos bits restantes son bits de control, el bit `SOF` es un bit que identifica el primer pixel de un frame y el bit `DATA_VALID` permite comprobar si el registro contiene un valor válido (Ver Figura 4.2). Durante la lectura del frame han de comprobarse ambos bits para leer el frame correctamente.

La tasa de transferencia de datos del sensor dependerá de la frecuencia del cristal conectado al chip. Como máximo es posible la conexión de un cristal de 24MHz y la frecuencia máxima del reloj en el bus de datos viene dada por:

$$f_{SCK} = \frac{f_{CLK}}{12} \tag{4.1}$$

por tanto, la máxima frecuencia permitida para f_{SCK} es 2MHz. Además, será necesario respetar los tiempos entre operaciones de escritura/lectura especificados en la hoja de datos del chip. En la Subsección 4.3.3 se realiza un estudio detallado de los distintos tiempos involucrados en la transferencia del frame del sensor y se calcula la tasa de transferencia en FPS.

El consumo del chip oscila entre 12mA y 30mA dependiendo de si el sensor se encuentra en movimiento o permanece quieto [70]. Se indica que el consumo típico cuando el sensor se está moviendo es de 15mA, y puesto que en la operación del sistema el sensor estará generalmente en movimiento, este será el valor tomado como referencia para el cálculo del consumo del sistema.

4.2. Descripción de la arquitectura hardware del dispositivo

En la Figura 4.3 se muestran los distintos elementos que forman la arquitectura del sistema. Cada uno de los módulos 'EyeOF' (Ver Subsección 4.2.1) se encuentran relacionados mediante una interfaz SPI con el elemento central de la arquitectura: una placa de desarrollo de la serie NUCLEO del fabricante *ST Microelectronics*, concretamente la placa `STM32L476RG-NUCLEO`. Esta placa de desarrollo contiene el microcontrolador `STM32L476RG` compuesto por un núcleo ARM® Cortex®-M4 a 80MHz como máximo y diseñado para aplicaciones de bajo consumo [71].

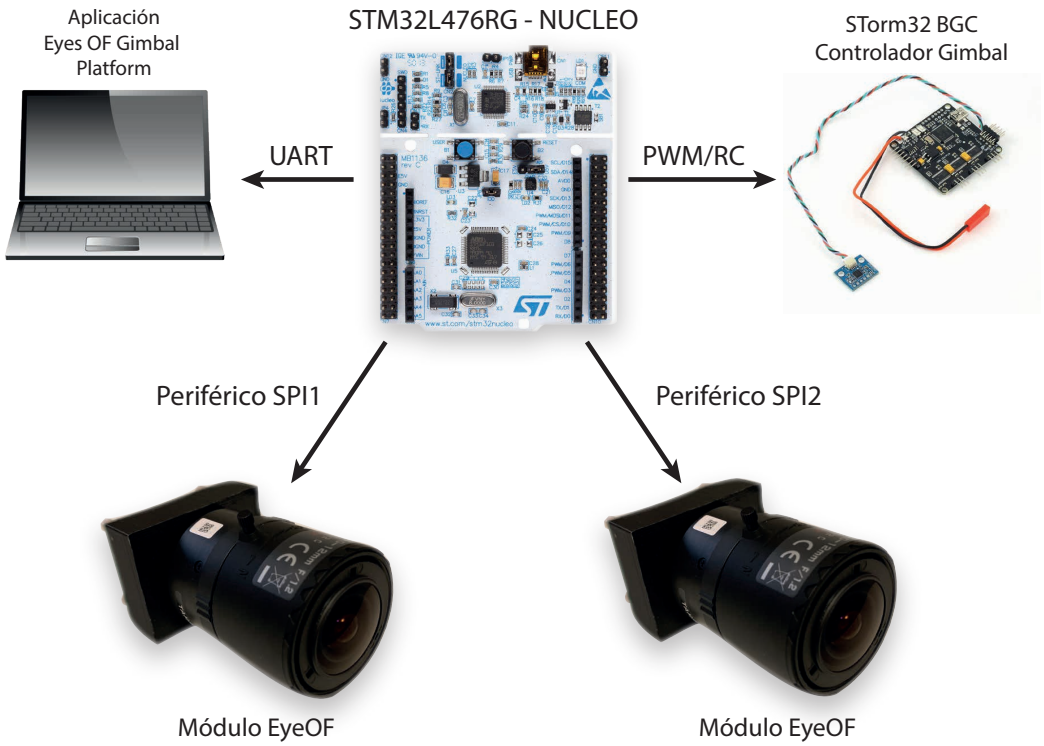

Figura 4.3: Arquitectura del dispositivo

Este microcontrolador posee una gran cantidad de periféricos hardware, lo que resulta muy útil en fases de prototipo donde distintas funcionalidades pueden surgir durante el proceso de diseño. Entre las características que han desembocado en la elección de este chip como núcleo del sistema se encuentran las siguientes:

- **Múltiples módulos SPI**: Aunque el estándar SPI permite que el bus sea compartido, en esta aplicación en específico donde se ha hecho una adaptación de las señales del estándar (Ver Subsección 4.2.1). Para la comunicación con el sensor ADNS2610 se hace necesario un módulo SPI por cada sensor. Sería posible utilizar un sólo módulo SPI mediante el uso de multiplexores que dirigieran el camino de la señal a cada sensor según sea preciso, sin embargo, en esta ocasión se ha decidido usar un módulo para cada sensor y disminuir el número de componentes del sistema.
- **DMA (*Direct Memory Access)***: Permite la transferencia de datos entre memoria y periféricos o entre distintas zonas de memoria sin la necesidad de hacer uso de la CPU. De esta forma es posible implementar la concurrencia entre tareas de procesamiento y transferencia de datos.
- **Consumo de potencia reducido**: Según la documentación del chip, el consumo es de 100 μA/MHz cuando trabaja con un regulador LDO como fuente de alimentación, como es el caso de la tarjeta NUCLEO elegida, por tanto, se tiene un consumo aproximado de 8 mA si se trabaja a la máxima velocidad permitida (80 MHz). Teniendo en cuenta el consumo medio del sensor ADNS2610 (Ver Sección 4.1) se tiene un consumo máximo total de:

$$15\ mA * 2 + 8\ mA = \textbf{38 mA@5 V} \rightarrow \textbf{P=190 mW}$$

Ha de anotarse que este consumo se relaciona con la adquisición del frame y el cálculo de *Flujo Óptico* y puede servir de referencia en diseños donde se desee implementar este sistema. En secciones posteriores se hará referencia al consumo total para la aplicación de seguimiento desarrollada en este trabajo.

4.2.1. Módulo EyeOF

La integración del sensor ADNS2610 en el sistema se ha realizado mediante el diseño de un pequeño módulo denominado 'EyeOF'. El módulo consta de tres elementos fundamentales:

- **PCB con electrónica de adaptación**: Esta PCB ha sido diseñada y fabricada ad hoc para la implementación del sistema. Se trata de una PCB que incluye todos los componentes necesarios para el correcto funcionamiento del chip, así como un conector con las señales necesarias para su alimentación y comunicación.
- **Soporte para lente montura C**: Permite la sujeción de una lente de montura C en la posición correcta, es decir, a una distancia entre el plano del sensor y el plano de la lente igual a la distancia focal y alineada con el centro de la zona activa del sensor ADNS2610.
- **Lente de montura C**: Cualquier lente de montura C es compatible con el módulo. En este caso se ha utilizado una lente C 1/2 4-12mm manual Iris de TAMRON.

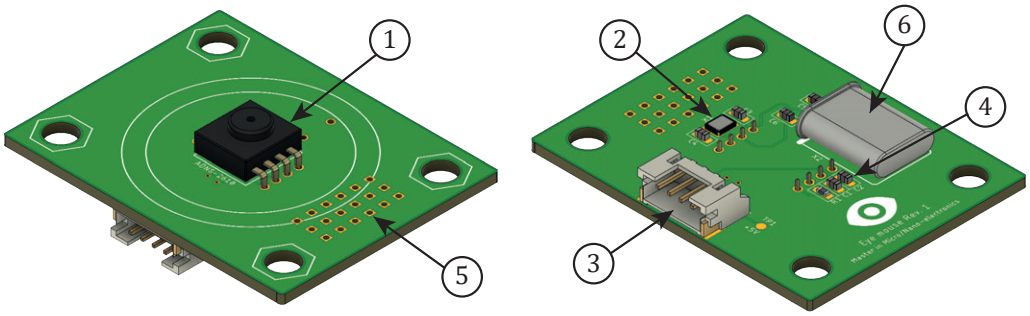

Figura 4.4: Módulo EyeOF: Renderizado 3D

Diseño de la PCB con electrónica de adaptación

El diseño de la PCB se ha desarrollado en el software *Eagle®* y *Fusion 360®* de la compañía *AutoDesk*. Este software permite desde la confección del esquema eléctrico que define las conexiones de la PCB (Figura 4.5) hasta el renderizado 3D de la PCB una vez finalizado el posicionamiento de los componentes y las pistas (Figura 4.4). Como es posible comprobar en la Figura 4.5 la PCB contiene además del sensor ADNS2610 (1), un cristal de cuarzo de 24MHz junto con sus capacidades de carga (2), un conector de cuatro pines para alimentación y señales de comunicación (3), condensadores de desacoplo (4) y una matriz de pads por si se necesitara de algún componente más no contemplado en el módulo (5). Además se ha añadido el footprint de un oscilador en formato Through-Hole (6) para tener disponible un abanico más amplio de osciladores posibles donde elegir. Sólo se soldará a la placa los componentes (2) o (6), es decir, estos dos componentes no deben coexistir en la PCB.

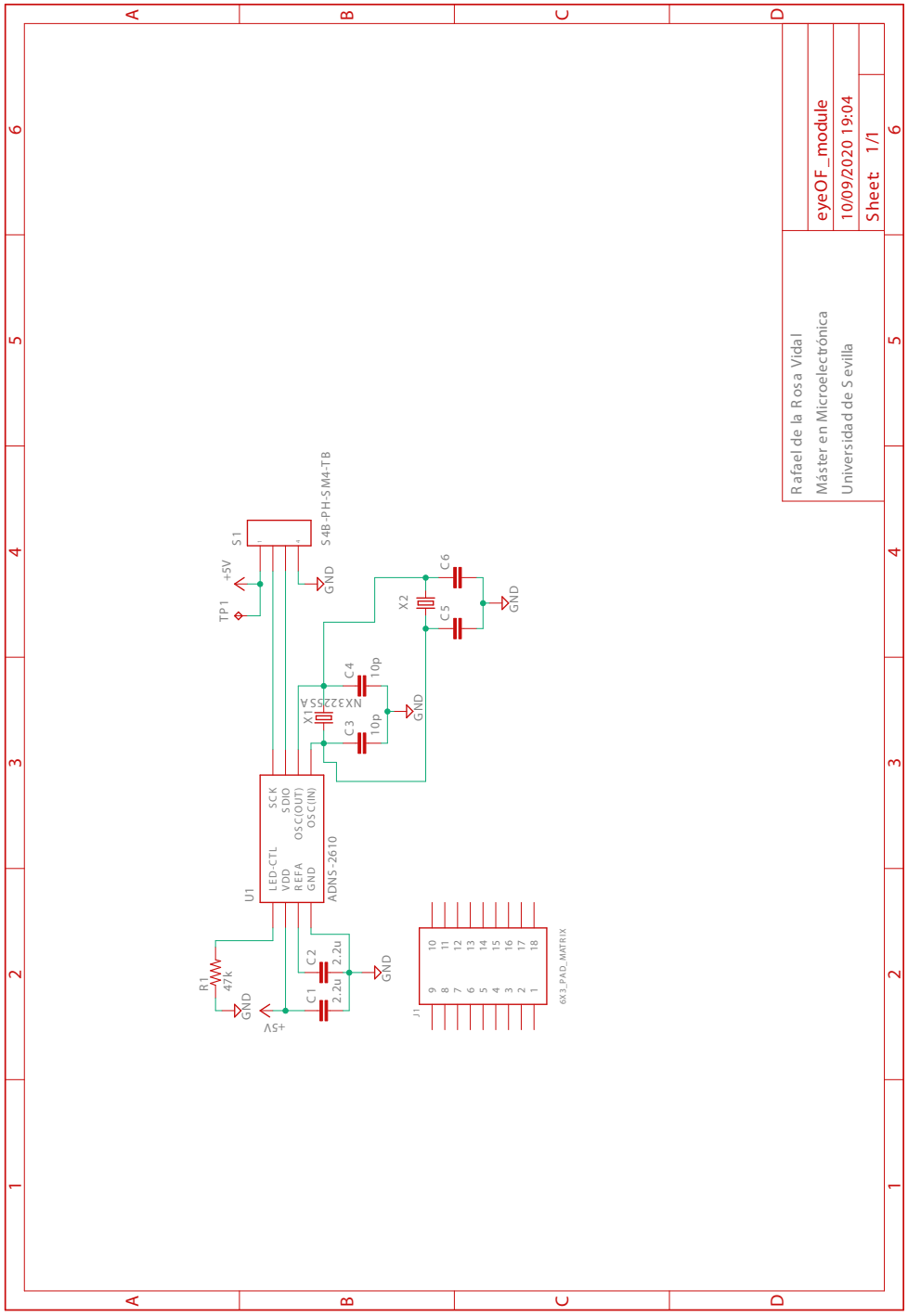

Figura 4.5: Esquema eléctrico del módulo EyeOF

Figura 4.6: Diseño 3D de cada uno de los componentes del módulo y su ensamblaje

Diseño mecánico del soporte y la lente

Los elementos que forman el dispositivo deben posicionarse de forma que la luz procedente del entorno sea dirigida al plano activo del sensor ADNS2610 a través de la lente del módulo. Para conseguir un resultado óptimo en la construcción del dispositivo, se decide realizar un diseño 3D del conjunto del dispositivo en escala 1:1 en el software *Fusion 360®* y, de esta forma, medir las distintas dimensiones de interés en el montaje. En la Figura 4.6 aparecen cada uno de los modelos 3D diseñados, así como el ensamblaje final de todas las piezas.

En [72] es posible consultar las distintas dimensiones relevantes para el diseño. Una montura de lente tipo C consta de una rosca 1-32 TPI y la distancia entre el panel frontal de la lente y la superficie del sensor de imagen debe ser de 0.69 inch, o lo que es lo mismo, 17.526 mm. *Fusion 360®* permite obtener la sección de un conjunto 3D y medir sobre el plano resultante. En la Figura 4.7 se muestra la medida de la distancia focal una vez dimensionado el soporte para que esta se cumpla.

Es necesario anotar que debido a que las dimensiones internas del chip no son conocidas, se ha estimado que el dado del chip se encuentra aproximadamente a la misma altura que los pines del encapsulado. Además, la medida de la distancia focal en la Figura 4.7 se ha realizado en condiciones ideales y habrá que tener en cuenta las tolerancias de la fabricación 3D para corregir posibles desajustes en el diseño. Una vez fabricada la pieza, la lente elegida permite un ajuste fino de la distancia focal, por lo que la tolerancia permitida es relativamente alta.

4.2.2. Construcción del dispositivo

Finalizado el diseño se procede a la fase de construcción del dispositivo. Para ello es necesario reunir cada uno de los componentes listados anteriormente en la Figura 4.6. La PCB se envió a fabricar externamente, el adaptador de rosca 1-32 TPI es estándar y junto con la lente se obtuvo de un proveedor y la estructura de unión de la lente y la PCB fue fabricada en una de las impresoras 3D disponibles en el IMSE.

En la Figura 4.8 se muestra el resultado obtenido tras la construcción de los dos módulos EyeOF junto con los cables necesarios para conectarlos a la placa `STM32L476RG` a través de un HAT de prototipado. Uno de los módulos se ha mantenido sin lente para que se pueda observar el interior del módulo en la figura.

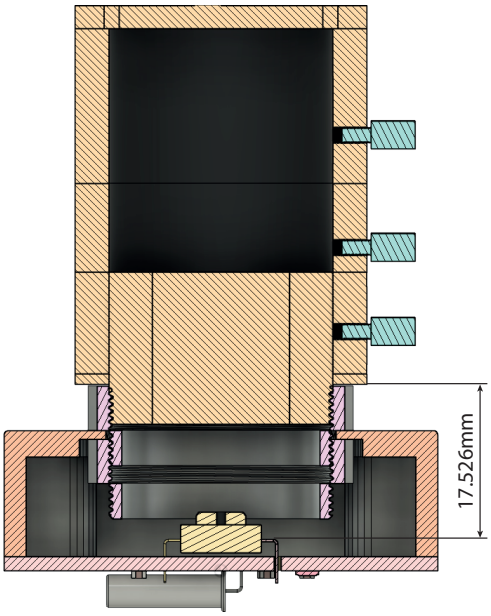

Figura 4.7: Sección del módulo EyeOF y medida de la distancia focal necesaria para cumplir con el estándar de montura tipo C

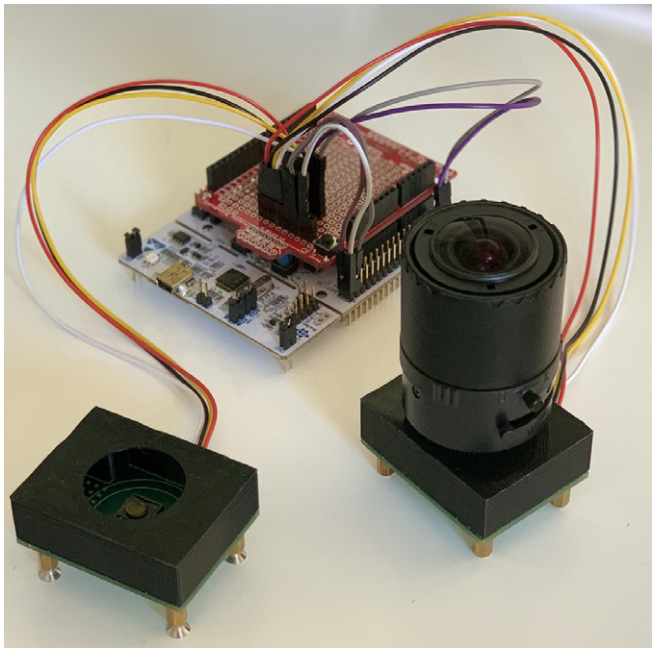

Figura 4.8: Módulos EyeOF conectados a la placa STM32L476RG

Una vez la construcción sea definitiva y se asegure que no se va a desmontar el módulo nuevamente se retirará la pared superior del encapsulado para permitir que le llegue la máxima cantidad

de luz posible a la zona activa del sensor de imagen. La razón por la cual se decide posponer esta operación es porque si la zona activa del sensor se ensucia durante su manipulación es muy difícil eliminar dicha suciedad y el frame obtenido no será válido, además de que no es necesario para comenzar con los primeros ensayos del dispositivo puesto que incluye una pequeña apertura por la cual entra la luz del exterior.

Adaptación estándar SPI a la interfaz del sensor ADNS2610

La interfaz SPI [73, 74] permite una comunicación full duplex y está formada por 3 señales principalmente: `MOSI`, `MISO` y `SCK`. La señal `MOSI` (MasterOuputSlaveInput) es unidireccional, el flujo de datos tiene su origen en el Master y como destino el Slave; la señal `MISO` (MasterInputSlaveOutput) es unidireccional y su sentido es el opuesto a la señal `MOSI`.

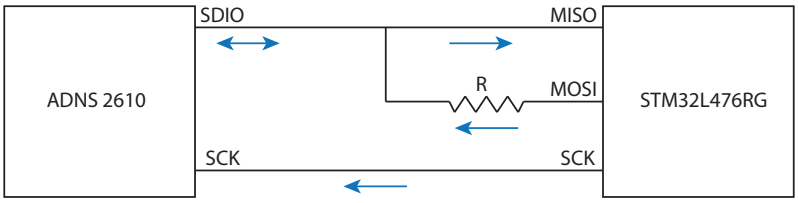

Figura 4.9: Adaptación de la interfaz SPI a la interfaz serie del chip ADNS2610

Puesto que la interfaz de comunicación del sensor ADNS2610 sólo consta de 2 señales, una de datos (`SDIO`) y otra de sincronismo (`SCK`) es necesaria una adaptación para poder establecer la comunicación correctamente. En la Figura 4.9 se muestra el esquema de la adaptación realizada. La resistencia `R` hará de carga en la recepción donde se conectará la señal `MOSI` a GND y su valor será tan bajo como la capacidad de carga de la salida `SDIO` lo permita, puesto que a medida que se aumenta el valor el ancho de banda del canal disminuye. En [70] se especifica que la capacidad de carga máxima de `SDIO` es 2 mA y la capacidad de entrada es de 10 pF, por lo que los valores de `R` deben encontrarse en el intervalo [2,5 kΩ, 5 kΩ] según la Ecuación 4.2 y la Ecuación 4.3[1]. El valor de `R` que se elige es 3,3 kΩ.

$$R_{min} = \frac{5\ V}{2\ mA} = 2{,}5\ k\Omega \tag{4.2}$$

$$R_{max} = \frac{1}{2\ MHz \cdot 10 \cdot 10\ pF} = 5\ k\Omega \tag{4.3}$$

4.3. Descripción de la arquitectura firmware del dispositivo

Como se comenta anteriormente, el núcleo del sistema es el microcontrolador `STM32L476RG` contenido en la placa `STM32L476RG - NUCLEO`. El fabricante ofrece una IDE propietaria de nombre STM32CubeIDE basada en la IDE Eclipse, ampliamente utilizada, y que, además, contiene una herramienta de configuración del chip denominada STM32 CubeMX. El software incluye el compilador *gcc* que es configurado automáticamente al indicar a qué tipo de hardware se encuentra dirigido el proyecto y permite la depuración del código mediante puntos de ruptura y análisis del contenido de la memoria entre otras funcionalidades. El lenguaje de programación permitido es C/C++, en este caso todo el código generado es en lenguaje C.

[1]Se ha multiplicado por 10 el ancho de banda que se desea (2 MHz) para asegurar que el canal de transmisión no influye nada en la señal.

La arquitectura del firmware desarrollada se basa en una máquina de estados finitos (FSM) dirigida mediante interrupciones hardware. Se trata de utilizar al máximo los recursos hardware de los que se dispone y evitar el uso de la CPU en operaciones que no la requieran de forma obligada. Por otro lado, la organización del código se orienta a la legibilidad del mismo, se han separado los distintos módulos, estructuras de memoria y funciones en librerías separadas y bien relacionadas.

En las sucesivas secciones se recorrerá el proceso de diseño del firmware, desde la configuración de los periféricos del microcontrolador, pasando por la definición de los estados y las transiciones entre estados de la máquina de estados y el desarrollo de varias funciones relevantes en el tratamiento y procesamiento del frame.

4.3.1. Configuración del microcontrolador en la herramienta STM32 CubeMX

Al iniciar un proyecto en STM32CubeIDE es posible indicar que tarjeta de desarrollo se pretende utilizar para el proyecto, de forma que la configuración del microcontrolador para esa placa se cargue en el proyecto. Por ejemplo, para la `STM32L476RG - NUCLEO` los pines PA2 y PA3 son usados por la USART2 para tareas de depuración y consola y el pin PA5 para un led y, por tanto, estos no están disponibles para realizar otras funciones (Ver Figura 4.10).

En el microcontrolador se establece la siguiente configuración:

- **Módulo EyeOF 1 → SPI2 y módulo EyeOF 2 → SPI3:** Ambos módulos serán configurados de la misma forma, palabras de 8 bits en la transferencia, MSB first, 2 MBs de baudrate, polaridad alta del reloj en estado idle y fase de reloj en el flanco de subida. Para

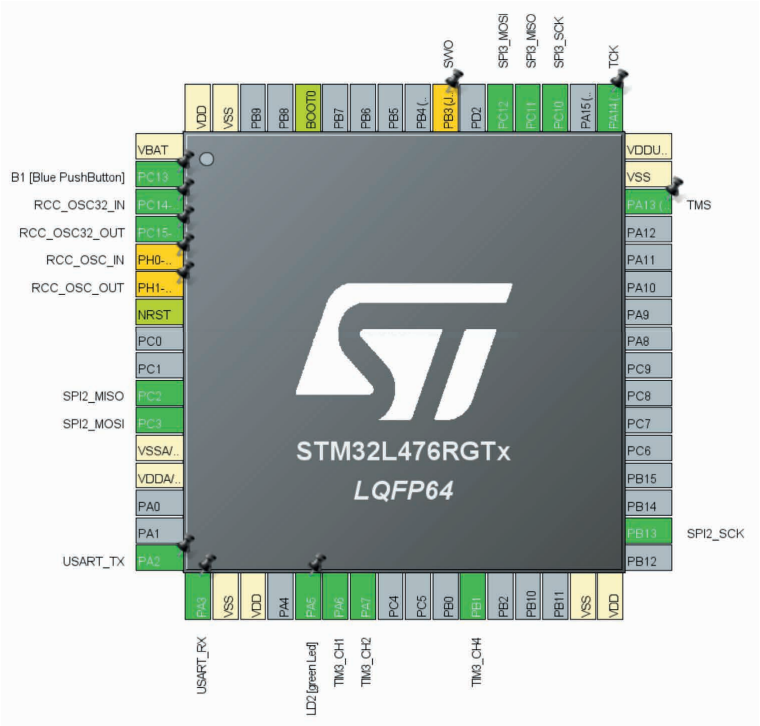

Figura 4.10: Pinout del microcontrolador STM32L465RGTx

la configuración del baudrate ha sido necesario la modificación de la frecuencia de reloj del microcontrolador a 64 MHz. Como se verá posteriormente, esta frecuencia de operación es suficiente para la aplicación y además se disminuye el consumo del dispositivo.

- **Transferencia de datos al PC para monitorización y control → UART2:** Se configura este periférico a la máxima velocidad de transferencia permitida, 921600 bps. La transferencia de datos se hará en binario según un formato conocido que será leído en una aplicación de PC. Por otro lado, el periférico será configurado para que use el módulo DMA en la transferencia.
- **Configuración y estado del controlador STorM32 BGC → UART1:** Mediante este periférico se mandarán diferentes comandos para configurar y conocer el estado del controlador STorM32 BGC según la API interna desarrollada en su documentación.
- **Señales RC para el control de la posición del gimbal → TIM3:** Se utilizará el timer 3 para generar tres señales PWM a través de tres de los cuatro canales de los que dispone. Cada canal se encuentra vinculado con un pin del microcontrolador, en este caso, se tiene PA6 (canal 1), PA7 (canal 2) y PB1 (canal 4). Cada una de las señales PWM controlarán el Pitch, el Roll y el Yaw del gimbal (Ver Sección 5.3). Las características del PWM se determinarán en la Subsección 5.4.2.

La herramienta STM32 CubeMX permite el uso de dos tipos de librerías en la implementación de la configuración en código: las librerías HAL (High Abstraction Layer) y las librerías LL (Low Level). Las librerías HAL, aunque más fáciles de usar, añaden una sobrecarga de operaciones que disminuyen la eficiencia del código. Por otro lado, las librerías LL implementan el código a bajo nivel permitiendo alcanzar un nivel de eficiencia mayor. En esta caso se utiliza las librerías LL aunque no es demasiado relevante porque solo se implementa dicho código en la fase de inicialización del dispositivo, no durante su funcionamiento.

4.3.2. Diseño de la máquina de estados finitos que define el funcionamiento del dispositivo

El dispositivo se puede encontrar en 5 estados posibles de funcionamiento. En la Figura 4.11 se muestra un diagrama de estados de la máquina de estados finitos definida. Cada uno de los estados se irán sucediendo en el tiempo según los estados anteriores y otros factores que se irán detallando a continuación. Todo el código generado para la implementación de la FSM se encuentra en el Anexo D.

Estado SENSOR_RESET

Todas las variables del sensor se devuelven a su estado inicial. La operación del dispositivo es interrumpida hasta que se indique el inicio de operación. Este estado es accesible por todos los demás estados.

Estado TRIGGER_FRAME

Se indica al sensor ADNS2610 que se va a proceder a la lectura de un frame. Para ello se realizar una operación de escritura al registro `PixelData`. El próximo valor de `PixelData` que se lea debe tener el bit SOF ('Start of Frame') a 1.

Tras la operación de escritura debe esperarse un tiempo para que el hardware de adquisición del frame se rearme. Este tiempo se aprovecha para configurar una transferencia DMA de todos los datos del sensor que se desee monitorizar desde el PC. Ver Subsección 4.3.6 para más detalles.

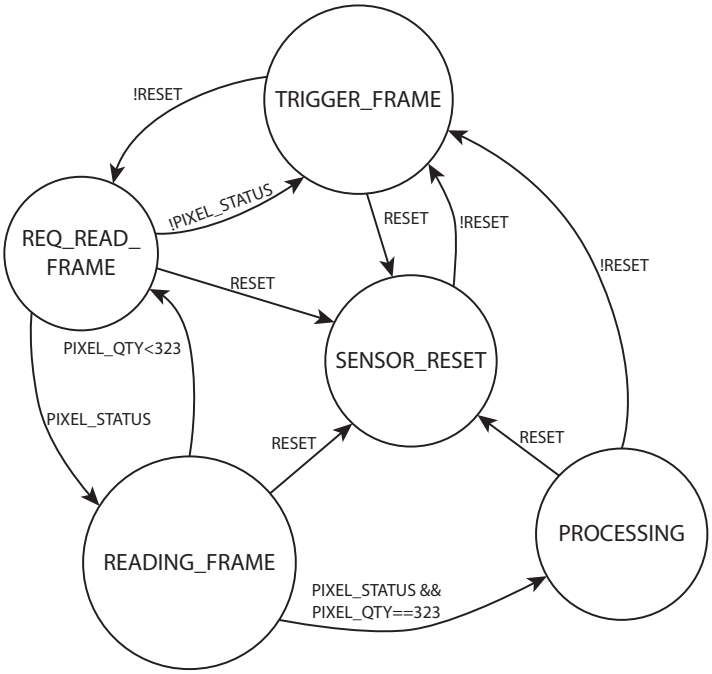

Figura 4.11: Diagrama de estados de la FSM del módulo EyeOF

Estado REQ_READ_FRAME

Se le envía al sensor ADNS2610 un requerimiento de lectura del registro `PixelData`. Mientras se espera a que el valor del registro `PixelData` sea válido se realizan dos operaciones:

- Se comprueba el estado del pixel leído en la operación anterior. Esto solo se realiza tras la lectura del primer pixel, es decir, cuando se envía el requerimiento de lectura del segundo pixel se comprueba el estado del primero y si el estado es no válido el siguiente valor leído sobrescribirá al actual, sin incrementar el número del pixel.

- Se realizan los cálculos relacionados con el *Flujo Óptico* que sean posibles. En función del número de pixeles que se hayan leído se podrá comenzar a realizar una serie de operaciones para el cálculo del *Flujo Óptico*, así la carga computacional de procesamiento se distribuye entorno a la operación de lectura (Ver Subsección 4.3.4).

Ambas operaciones se realizan en este punto para aprovechar el tiempo de holgura producido por la espera necesaria para que el sensor tenga preparado el valor del registro `PixelData`.

Estado READING_FRAME

Tras la espera, se realiza una operación de lectura del registro `PixelData` que contiene los datos relacionados con el pixel a leer.

Cuando la cantidad de pixeles llega a su final, en este estado se realiza la comprobación del contenido del pixel y si es válido se pasa al estado `PROCESSING`.

Estado PROCESSING

Como se puede consultar en la Sección 3.3, el cálculo de *Flujo Óptico* necesita del almacenamiento de dos frames, uno en el instante t y otro anterior en el instante t_0. Para ello, se disponen dos arrays por cada modulo EyeOF, donde los frames serán almacenados alternativamente. Llegados a este estado, se tienen las siguientes opciones:

- Si es la primera vez que se llega a este estado quiere decir que sólo se ha completado la lectura de un frame de cada módulo, es decir, dos de los arrays contienen datos, mientras que los dos restantes no. En este caso se continúa directamente al estado `TRIGGER_FRAME` y los frames almacenados son los correspondientes al instante t_0.

- Si los dos arrays de cada módulo contienen datos de frames, se calcula el *Flujo Óptico* que relaciona los frames almacenados: los frames en el instante t y los anteriores en el instante t_0. Para más detalle consultar Subsección 4.3.4.

- El siguiente frame debe almacenarse en el array que contiene el frame en t_0, por tanto, se modifica el puntero donde se va a almacenar el siguiente frame a dicha dirección de memoria. El array que contenía el frame en t ahora pasa a contener el frame en t_0 para el próximo cálculo de *Flujo Óptico*.

Por último, si se ha obtenido una nueva medida de *Flujo Óptico* se implementa la ley de control necesaria para que la plataforma adopte la posición adecuada para el seguimiento del objetivo. Tras esta operación se pasa al estado `TRIGGER_FRAME` y se repite la secuencia de estados descrita.

4.3.3. Configuración y lectura del sensor ADNS2610

Para la comunicación con el chip ADNS2610 se ha desarrollado una librería recogida en los ficheros `adns2610.c` y `adns2610.h` disponibles en el Anexo B. La librería se encarga de configurar los periféricos SPI para los dos sensores, configurar el sensor ADNS1610 y gestionar las operaciones de escritura y de lectura.

La configuración del sensor se concentra en el registro `Configuration`. En este registro se configura que el sensor se mantenga encendido ininterrumpidamente, ya que el sensor por defecto entra en modo bajo consumo y no se desea eso.

Como se comenta en secciones anteriores es necesario esperar un tiempo entre el requerimiento de la lectura del valor de un pixel y la propia lectura. Existen otros tiempo como el tiempo entre operaciones de lectura y el tiempo entre operación de escritura y operación de lectura de también han sido ajustados. Los valores finales tomados para los distintos tiempos se muestran en la Tabla 4.1 y según estos valores se tiene que el tiempo de transferencia de un frame viene dado por la Ecuación 4.4. El número de imágenes por segundo (FPS) es aproximadamente igual a 5 (Ecuación 4.5).

$$t_{acq} = (50\ \mu s + 600\ \mu s) \cdot 324 + 700\ \mu s = 211{,}3\ ms \tag{4.4}$$

$$FPS = \frac{1}{211{,}3\ ms} = 4{,}732 \approx 5\ FPS \tag{4.5}$$

Magnitud	**Valor**
Tiempo entre operaciones de lectura	50 µs
Tiempo entre requerimiento de lectura y lectura	600 µs
Tiempo entre rearme de hardware de adquisición y lectura	700 µs

Tabla 4.1: Tabla de tiempos en la operación de lectura del sensor ADNS2610

4.3.4. Implementación en código del cálculo de *Flujo Óptico*

La implementación del algoritmo de interpolación [49] se ha realizado de forma que la carga computacional se encuentre distribuida y no suponga un cuello de botella en el funcionamiento del dispositivo. Para ello, se ha basado la operación en tablas de índices que indican los valores que pueden ser computados en cada instante durante el proceso de lectura del frame.

Los frames son almacenados en arrays unidimensionales de 324 elementos, resultado de un frame monocromático de 18×18 pixeles de resolución. Los índices del array se corresponderán espacialmente según se muestra en la Figura 4.12. Como se desarrolló en la Sección 3.3 el cálculo de *Flujo Óptico* viene dado por la suma de una serie de coeficientes calculados sobre los pixeles de diferentes regiones del frame. La posición de los pixeles para cada coeficiente viene dada por el valor de Δx y Δy que en este caso han sido igualados a la unidad. De esta forma, el conjunto de índices necesarios para el cálculo de cada coeficiente viene dado según la Figura 4.13 donde se señalan los diferentes índices necesarios para el cálculo de cada coeficiente a medida que se avanza sobre el frame con un color más opaco: para el primer coeficiente, los índices 19, 18, 20, 1, 37; para el segundo coeficiente, los índices 20, 19, 21, 2, 38; para el tercer coeficiente, los índices 21, 20, 22, 3, 39; y así, sucesivamente. Se observa que el primer coeficiente no es posible calcularlo hasta que se alcanza el índice de mayor valor necesario para su cálculo, el índice 37. Esto es lo que se indica en el mapa de índices de la Figura 4.12, los índices coloreados en azul indican que cuando se lee el pixel que corresponde a ese índice, es posible calcular uno de los coeficientes del *Flujo Óptico*. En los bordes del frame no es posible calcular ningún coeficiente excepto en el borde inferior donde se calculan los coeficientes correspondientes a la fila superior.

Cada coeficiente calculado se suma directamente al resultado de la suma de los coeficientes anteriores, habiendo reseteado la suma al principio de cada frame. Cuando se alcanza la lectura del último pixel, los sumatorios han sido calculados y solo quedaría calcular el valor final del *Flujo Óptico* según las relaciones expresadas en la Ecuación 3.28 y la Ecuación 3.29. Es posible consultar todo el código de la librería en el Anexo C, particularmente las funciones que implementar el cálculo de los coeficientes y el *Flujo Óptico* son las funciones `OF_ComputeCoefficients` y `OF_Compute`.

4.3.5. Fusión de los vectores de *Flujo Óptico*

Los dos módulos EyeOF que contiene el sistema se disponen coplanarios, como se muestra en la Figura 4.14. Cada uno de ellos genera un vector de *Flujo Óptico* en el sistema de coordenadas $(x_{right\_eye}, y_{right\_eye})$ y $(x_{left\_eye}, y_{left\_eye})$ resultado del frame captado por cada uno de ellos. Debido a su disposición en el espacio, ambos vectores se pueden asociar al mismo escenario físico y, por tanto, es posible su fusión como si se trataran de vectores originados por dos regiones distintas de un mismo frame. El proceso de fusión tiene como resultado:

- Un vector de *Flujo Óptico* cuyas componentes son el resultado de realizar la media aritmética entre las componentes de los dos vectores generados por los módulos EyeOF. Su componentes se encuentran situadas en (x_{fused}, y_{fused})

17	35	53	71	89	107	125	143	161	179	197	215	233	251	269	287	305	323
16	34	52	70	88	106	124	142	160	178	196	214	232	250	268	286	304	322
15	33	51	69	87	105	123	141	159	177	195	213	231	249	267	285	303	321
14	32	50	68	86	104	122	140	158	176	194	212	230	248	266	284	302	320
13	31	49	67	85	103	121	139	157	175	193	211	229	247	265	283	301	319
12	30	48	66	84	102	120	138	156	174	192	210	228	246	264	282	300	318
11	29	47	65	83	101	119	137	155	173	191	209	227	245	263	281	299	317
10	28	46	64	82	100	118	136	154	172	190	208	226	244	262	280	298	316
9	27	45	63	81	99	117	135	153	171	189	207	225	243	261	279	297	315
8	26	44	62	80	98	116	134	152	170	188	206	224	242	260	278	296	314
7	25	43	61	79	97	115	133	151	169	187	205	223	241	259	277	295	313
6	24	42	60	78	96	114	132	150	168	186	204	222	240	258	276	294	312
5	23	41	59	77	95	113	131	149	167	185	203	221	239	257	275	293	311
4	22	40	58	76	94	112	130	148	166	184	202	220	238	256	274	292	310
3	21	39	57	75	93	111	129	147	165	183	201	219	237	255	273	291	309
2	20	38	56	74	92	110	128	146	164	182	200	218	236	254	272	290	308
1	19	37	55	73	91	109	127	145	163	181	199	217	235	253	271	289	307
0	18	36	54	72	90	108	126	144	162	180	198	216	234	252	270	288	306

Figura 4.12: Tabla de índices: En azul los índices en los que algún coeficiente de la operación de *Flujo Óptico* puede ser calculado; en rojo los índices en los que no.

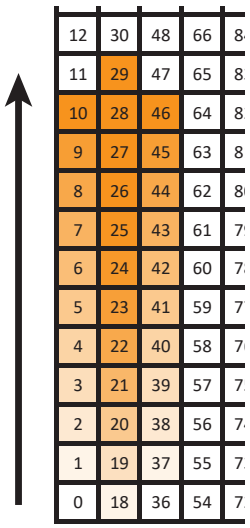

Figura 4.13: Indices tomados en el cálculo de cada coeficiente. Avance del cálculo de coeficientes.

- Un momento respecto al eje perpendicular al plano donde se encuentran dispuestos los módulos y que pasa por el centro geométrico de los dos módulos resultado de la resta entre las componentes en el eje y del frame. Esto se traduce en información sobre la rotación del escenario frente a los módulos.

Esta operación se realiza en la función `OF_ComputeFused` disponible en el Anexo C.

4.3.6. Transferencia de datos al PC

Todos los datos que genera el dispositivo se almacenan en estructuras empaquetadas en memoria. De esta forma, es posible realizar la transferencia de estos datos a través de una USART mediante DMA. La estructura y la descripción de sus miembros se encuentra en la Tabla 4.2 y es posible consultarla en el código en el Anexo D bajo el nombre `frameStruct`.

Solo es necesario indicar al periférico DMA la dirección donde comienza el bloque de memoria que se desea transferir y su longitud. El bloque hardware realizará la transferencia y avisará a la CPU mediante una interrupción cuando se haya completado. De esta forma, se puede verificar que no se corrompen los datos al empezar la transferencia mientras la anterior sigue en curso.

Mientras el periférico DMA realizar la transferencia de datos, la CPU continúa procesando el frame actual. Sólo se necesitan un número reducido de instrucciones para configurar el bloque hardware. De otra forma, la CPU estaría ocupada un gran número de instrucciones durante la transferencia y la capacidad de procesamiento de los frames disminuiría en gran proporción.

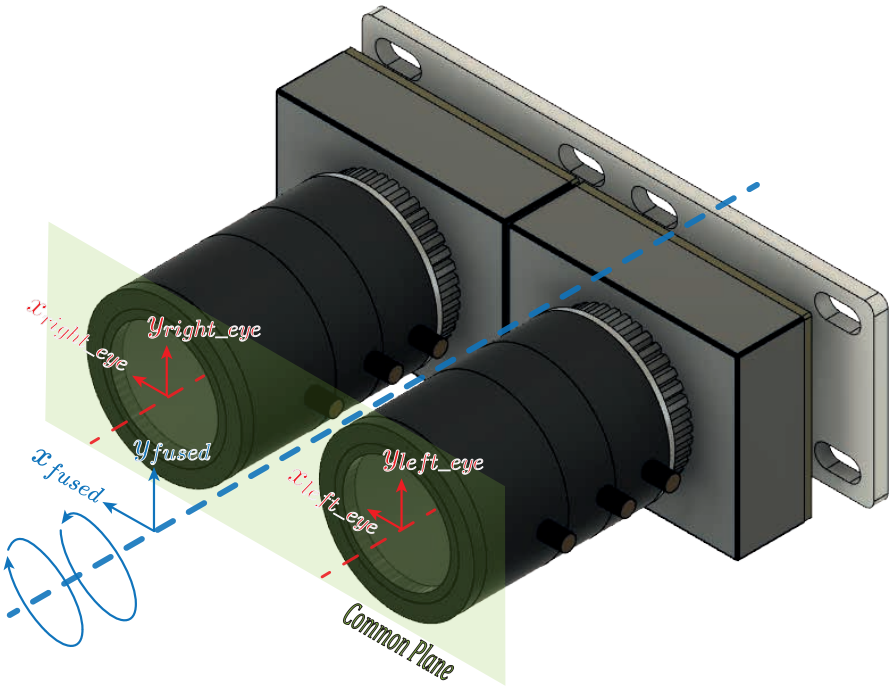

Figura 4.14: Disposición espacial de los módulos EyeOF en el sistema. En rojo los sistemas de coordenadas relacionados con cada uno de los módulos EyeOF del sistema; en azul el sistema de coordenadas para el vector resultado de la fusión y el eje sobre el cual se aplica el momento resultado de la fusión.

Dato	**Tipo de estructura**	**Descripción**
Header	`uint32_t`	Encabezado conocido del paquete de datos
Número de secuencia	`uint8_t`	Número de secuencia del paquete, es posible detectar si ha existido algún error en la transferencia si dos paquetes no contienen números de secuencia consecutivos
Imágenes	`uint8_t[2][324]`	Contienen los valores de los pixeles obtenidos en la lectura de las imágenes de los dos sensores ADNS2610 del dispositivo
Flujo óptico	`int32_t[7]`	Contiene los valores de los diferentes valores del *Flujo Óptico* (para las imágenes de cada sensor y el resultado de la fusión de ambos sensores)
Estado del sensor	`uint8_t`	Reporte del estado del dispositivo en distintos subconjuntos de bits en el registro

Tabla 4.2: Estructura de datos del bloque de memoria transferido al PC y su descripción

Capítulo 5

Diseño y control de la plataforma móvil

El array de módulos EyeOF desarrollado en el Capítulo 4 debe tener libertad de movimiento en el espacio tridimensional. Es necesario de un soporte mecánico y un controlador que añada esta capacidad al sistema: a partir de la información generada por el *Flujo Óptico*, mover los módulos de manera coherente. El diseño mecánico de la plataforma se va a basar en estructuras estabilizadoras denominadas gimbal o cardán, existentes en el mercado y que son utilizadas para la realización de videos estabilizados (sin vibración). Este tipo de sistemas suelen integrarse en UAVs con cámaras a bordo, ya que de otro modo, la dinámica del UAV y las perturbaciones como las corrientes de viento harían imposible la captura imágenes o vídeos de forma óptima. Como ejemplo de este tipo de sistemas, se tiene el dron DJI Phantom 4 Pro v2.0 [75] de la marca DJI mostrado en la Figura 5.1.

Por otro lado, es necesario un sistema de control que genere las señales de control necesarias para mantener los motores contenidos en la plataforma en la posición deseada. Generalmente los motores utilizados en los gimbals son motores DC brushless de baja velocidad. Como ejemplos de sistemas de control para gimbal con motores DC brushless se tiene `STEVAL-GMBL02V1` de ST Microelectronics [76] , `AlexMos` de BaseCam [77] o `STorM32-BGC` [78] que es un proyecto desarrollado y soportado por una gran comunidad OpenSource y actualmente es ampliamente utilizado en el mundo de los UAVs con cámaras estabilizadas integradas. Este último es el sistema que se va a utilizar en este trabajo.

En las siguientes secciones se va a desarrollar cada elemento que compone la plataforma móvil y las decisiones de diseño tomadas durante su desarrollo. Aspectos como el modelado mecánico de la plataforma y los motores no se realizan debido a que se encuentra fuera del alcance del trabajo. Se hará más hincapié en la parte de control y en la adaptación del controlador necesaria para su uso con los módulos EyeOF. En cualquier caso siempre se darán las referencias adecuadas que permitirán al lector profundizar en temas de su interés y que en este capítulo no se traten.

5.1. Motores DC brushless

La construcción de un motor DC brushless es muy parecida a la construcción de los motores DC brushed. En estos últimos se tiene un estátor con imanes permanentes y un rotor con bobinas que hacen de electroimanes. Al alimentar el motor con una tensión DC V_M, por acción de las escobillas se genera una tensión AC sobre las bobinas del rotor, que a su vez, genera un campo magnético

Figura 5.1: Dron DJI Phatom 4 Pro v2.0 de la marca DJI que contiene un gimbal o cardán para estabilizar la cámara

que interacciona con los imanes permanentes del estátor y produce la rotación. Un esquema básico de la construcción de un motor DC brushed es mostrado es la Figura 5.2 (a).

En los motores DC brushless (Ver Figura 5.2 (b)), los imanes permanentes se disponen en el rotor y las bobinas en el estátor. Las escobillas son reemplazadas por un inversor electrónico externo al motor, es decir, los motores DC brushless deben ser alimentados con tensiones AC [79, 80, 81]. Como consecuencia el control de estos motores es más complejo debido a la necesidad de la generación de señales AC aplicadas a los terminales de entrada (V_A, V_B y V_C en la Figura 5.2 (b)) siguiendo un patrón adecuado según la posición del rotor. En Wach [81] se desarrollan distintos métodos para controlar este tipo de motores.

Los motores DC brushless presentan una serie de ventajas relevantes en esta aplicación respecto a otros motores:

- Su peso es reducido, por tanto la plataforma es más ligera y necesita de menos potencia para su movimiento.
- Este tipo de motores permite desarrollar el máximo torque durante toda la rotación, siempre y cuando el control sea el óptimo. Otros motores solo alcanzan el torque máximo en ciertos puntos de la rotación.
- Permiten el control de su velocidad y su aceleración en el movimiento entre posiciones, lo que se traduce en movimiento más suaves que mejora la calidad de la estabilización. Otros motores como los motores paso a paso o los servomotores permiten el control de su posición pero no de la curva de velocidad y aceleración entre posiciones, produciéndose movimientos bruscos que degradan el rendimiento de la estabilización.

En este trabajo se van a utilizar motores brushless 2208 de 90 KV y 14 polos[1]. El valor KV de un motor brushless es una especificación que aporta el fabricante que permite conocer a priori si el motor es adecuado para la aplicación deseada. Este valor depende fundamentalmente del número de espiras en las bobinas del motor, el diámetro del hilo de cobre utilizado en dichas bobinas, la potencia de los imanes permanentes y la geometría del motor; y se refiere al número de revoluciones por minuto (RPM) que es capaz de ofrecer el motor cuando se le aplique señales de 1 voltio de

[1]número de imanes permanentes en el rotor y bobinas en el estátor

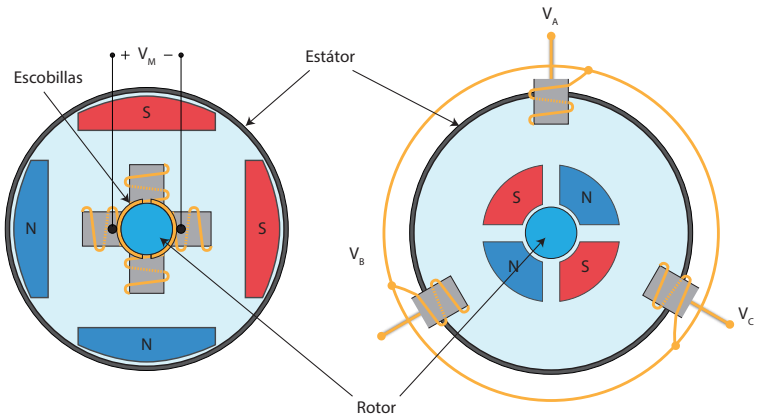

Figura 5.2: Construcción básica de motores DC brushed (a) y motores DC brushless (b)

amplitud sin existencia de carga sobre el eje del motor. Para aplicaciones de gimbal se suelen usar motores DC brushless en un rango de 50 KV a 150 KV, mientras que para otras aplicaciones como motores para hélices de drones se utilizan motores con valores KV mucho más altos, de 1200 KV en adelante. Por último, ha de tenerse en cuenta la resistencia que presenta el motor en cada fase, ya que esto determina la cantidad de corriente que puede llegar a demandar el motor. En este caso, según los drivers del controlador que se va a usar, se necesita de un motor de más de 10 Ω por fase. En el caso del motor elegido se tienen 13 Ω.

5.2. Controlador STorM32 BGC

El controlador `STorM32 BGC` es un proyecto que nace en una gran comunidad OpenSource dedicada al mundo de los drones. En esta comunidad se desarrollan ideas procedentes de necesidades de los miembros, así como se plantean problemas que abordan en conjunto, aportando cada miembro el conocimiento que tenga sobre la problemática. El controlador `STorM32 BGC` [78] permite el control de un sistema gimbal o cardán de tres ejes implementado mediante motores brushless. En esta sección se va a explicar las principales características del controlador, cómo configurarlo y qué funciones se van a utilizar en esta aplicación.

En la Figura 5.3 se muestra la planta de la tarjeta junto con indicaciones sobre algunas de sus conexiones externas. Las conexiones superiores corresponden a los motores para los ejes de Pitch, Roll y Yaw (Ver Sección 5.3 para más información sobre estos términos). Cada pin se corresponde a una fase (V_A, V_B y V_C) como las mostradas en la Figura 5.2 (b), si se tienen más de 3 bobinas en el motor, se interconectan de manera coherente para activar grupos de bobinas al actuar sobre cada una de las fases. En la zona inferior se muestra la conexión a la alimentación principal de la tarjeta[2]. Los valores de tensión admitidos se encuentran en el rango de 9V a 18V, con un consumo de potencia máximo en torno a los 60W. En este caso particular se ha optado por una fuente de alimentación DC de 12V y 5A (60W). A la izquierda se tienen las conexiones a 3 elementos:

[2]La tarjeta también puede ser alimentada mediante la conexión USB, aunque se utilizará esta siempre que esté disponible.

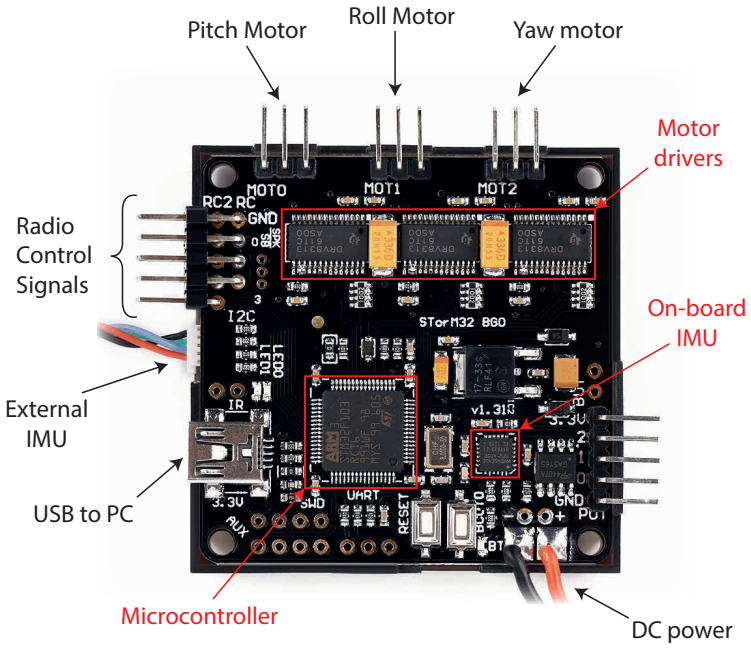

Figura 5.3: Planta de la tarjeta STorM32 BGC

conexiones a señales de Radio Control que permiten modificar la posición del sistema; una IMU[3] que permite conocer la posición de los módulos, se colocará en el mismo plano que los módulos y será conectada al controlador mediante un cable trenzado (Ver Sección 5.3); conexión USB para la configuración de la tarjeta desde el PC.

El núcleo de la tarjeta es el microcontrolador `STM32F103RCT6` de ST Microelectronics con arquitectura ARM® 32-bit Cortex®-M3 a 72 MHz [82]. Este dispositivo se encarga de realizar todos los cálculos relacionados con el control del sistema, la generación de las señales de control y la recepción de señales externas, como las señales de Radio Control. Como interfaz entre los motores y las señales de control generadas por el microcontrolador se tienen los drivers `DRV8313` [83]. Constituidos por 3 etapas push-pull N-MOSFET son adecuados para el control de cargas inductivas como solenoides o motores brushless. Permiten ser alimentados por 65 V como máximo, tienen capacidad de corriente de hasta 2,5 A e incluyen protecciones de sobretensión, sobrecorriente y sobretemperatura que son notificadas mediante uno de sus pines.

El controlador basa su función de control en el conocimiento de la posición de la tarjeta y la posición del objeto que se desea posicionar (en este caso los módulos EyeOF) [84, 85], para ello dispone de dos IMU[3] `MPU6050` [86, 87]: una integrada en la tarjeta y otra externa dispuesta junto el objeto a posicionar. Este circuito integrado se conecta mediante una interfaz I2C al microcontrolador y permite leer medidas de los giróscopos a 8 kHz y medidas de los acelerómetros a 1 kHz. En Kok, Hol, and Schön [88] se describen diversos algoritmos para la estimación de la posición y la orientación de un cuerpo a partir de la información entregada por IMUs. Este tema no se aborda en este trabajo puesto que es un campo muy amplio y se escapa de su alcance.

El controlador `STorM32 BGC` permite un grado de configurabilidad elevado. Entre las opciones que ofrece se encuentra la compatibilidad con diversas configuraciones de gimbal, la compatibilidad

[3] "Inertial Measurement Unit (IMU)" es un dispositivo constituido fundamentalmente por un giróscopo de 3 ejes y un acelerómetro de 3 ejes construidos generalmente en tecnologías MEMS.

con diferentes estándares de Radio Control (PWM [89], Spektrum Satellite [90], Futaba S-BUS [91], HoTT SUMD [92], entre otros) y el ajuste de los parámetros PID del controlador para poder estabilizar diferentes dinámicas. Todas estas opciones se encuentran disponibles a través de la GUI que aparece en la Figura 5.4. En las siguientes secciones se irán comentando los apartados de la aplicación relevantes en esta implementación.

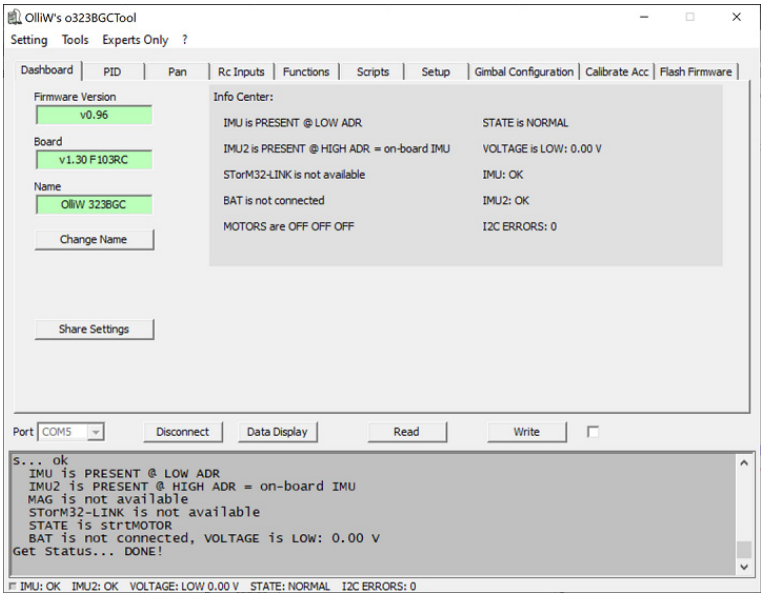

Figura 5.4: GUI para configuración del controlador STorM32 BGC

5.3. Diseño mecánico de la plataforma

Para el diseño mecánico de la plataforma se ha utilizado el software de modelado 3D *Fusion 360®* de la compañía *AutoDesk*, que, como se comentó en la Sección 4.2.1 permite integrar el diseño electrónico de los módulos EyeOF desde el software de diseño eléctrico *Eagle®*. El modelo 3D de la plataforma va a permitir estudiar la disposición de todos los elementos constituyentes (motores, estructuras de soporte, dispositivos de control, etc.), estudiar la posición relativa de cada elemento para evitar colisiones en el movimiento y generar archivos binarios (STL [93]) compatibles con métodos de fabricación aditiva (impresión 3D [93], en este caso) que permitirán obtener las piezas necesarias para la posterior construcción de la plataforma.

Para la representación de la posición y la actitud de un cuerpo en un espacio 3D se hace necesario de la definición de un sistema de referencia inercial ("Inertial frame[4]" en la Figura 5.5, que es fijo y no sufre ningún tipo de movimiento), y un sistema de referencia local al cuerpo cuyo origen de coordenadas es su centro de gravedad ("Body frame[4]" en la Figura 5.5), sus ejes se disponen como se muestra en la Figura 5.5 y es no inercial. Sobre el sistema de referencia local al cuerpo existen dos formas de definir la actitud de un objeto: mediante los ángulos de Euler o

[4]No confundir la traducción al inglés de sistema de referencia a frame y el concepto de frame definido en el Capítulo 3.

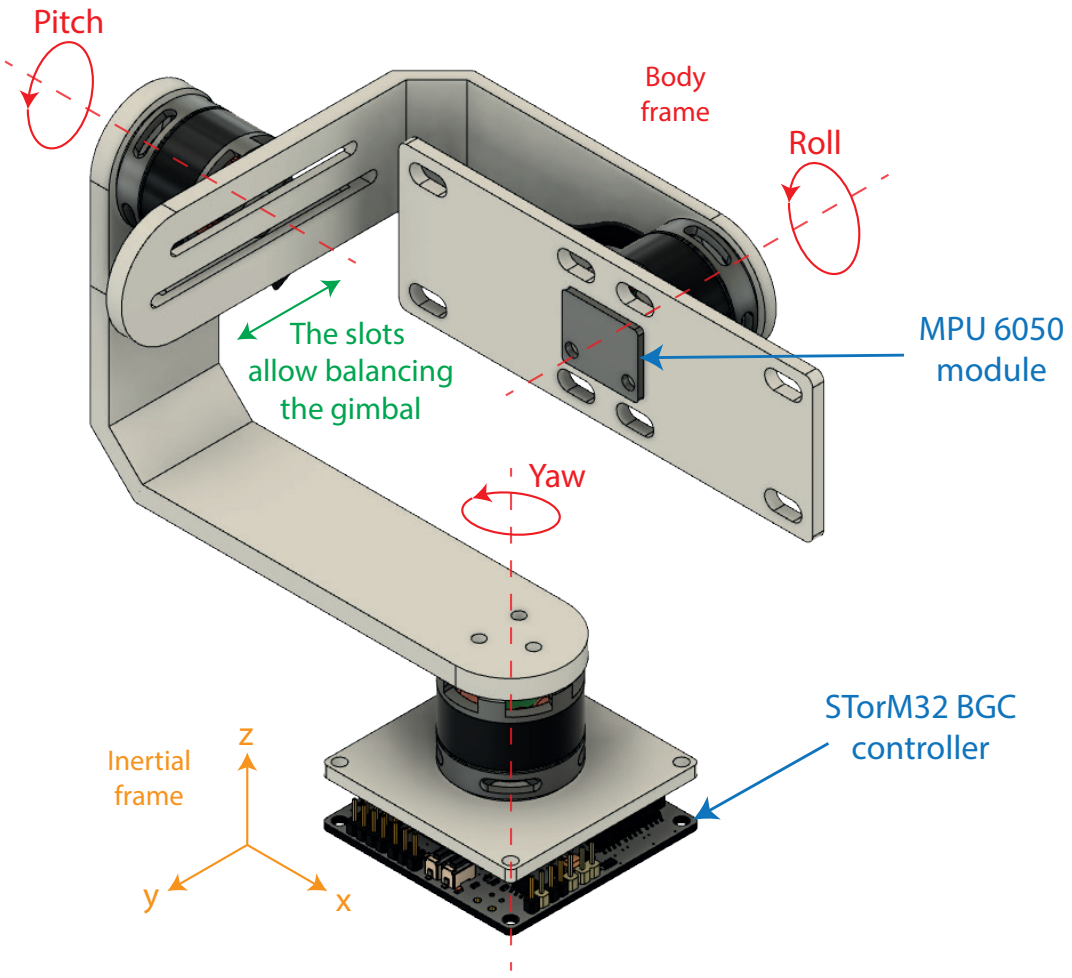

Figura 5.5: Diseño mecánico de la plataforma: modelo 3D del soporte de los módulos

mediante cuaterniones [85]. Los ángulos de Euler se definen como sigue:

- **Pitch:** Ángulo de rotación respecto al eje X del sistema de referencia inercial.
- **Roll:** Ángulo de rotación respecto al eje Y del sistema de referencia inercial.
- **Yaw:** Ángulo de rotación respecto al eje Z del sistema de referencia inercial.

Este tipo de representación de la actitud es ambigua cuando dos de los ejes se posicionan en paralelo (Ver *"Gimbal Lock Problem"* en [94]). Para solucionar esta singularidad se tiene la representación mediante cuaterniones. Un cuaternión es un vector cuadrimensional en la que la primera de sus componentes representa un ángulo y las tres componentes restantes describen el eje sobre el que se gira el ángulo definido por la primera componente. Cada posición en el espacio 3D da lugar a un cuaternión valor único.

En la plataforma que se desarrolla en este trabajo, los sistemas de referencia inercial y local al cuerpo vienen definidos por la posición de las IMUs. En la Sección 5.2 se comentó que el controlador

Figura 5.6: Modelo 3D del sistema y vistas ortogonales con movimientos a ±45°

contiene una IMU incorporada, que define el sistema de referencia inercial, y una IMU externa conectada al controlador que se dispone como se muestra en la Figura 5.5 y define el sistema de referencial local al cuerpo. La interacción entre sistemas de referencia se realiza mediante matrices de transformación [94]. Puesto que en la operación de la plataforma no se van a alcanzar posiciones en el espacio que hagan que dos de los ejes del sistema de referencia local al cuerpo se dispongan en paralelo, la representación de la actitud del cuerpo se va a realizar mediante ángulos de Euler.

Para que la plataforma funcione correctamente la estructura mecánica debe ser balanceada. Esto es que el centro de gravedad de la carga que soporta cada motor se sitúe en su eje de rotación. Al efecto, la posición del cuerpo en la plataforma quedará fija en cualquiera de las posiciones que pueda adoptar en el espacio 3D y los motores solo tendrán que ejercer una fuerza de rotación en las transiciones entre posiciones. De lo contrario, además de que exista un consumo de potencia excesivo y, como consecuencia, sobrecalentamiento de los motores que produzcan daños permanentes en el sistema, pueden producirse vibraciones generadas por la corrección continua que debe hacer el sistema para mantener la posición actual. Con el objetivo de balancear la estructura diseñada, puesto que no se conocen datos estructurales que permitan un ajuste fino, se han dispuesto en una de las piezas unas ranuras que permiten desplazar el cuerpo. Estas ranuras solo permiten balancear el eje de Pitch. Para el eje de Roll se tiene la certeza de que va a quedar equilibrado debido a que las cargas que se van a disponer (en este caso, los módulos EyeOF) pesan lo mismo y, a tal efecto, los momentos respecto al eje de rotación se anulan. Para el eje de Yaw la condición de equilibrio no es demasiado relevante puesto que la plataforma se va a mantener en posición vertical en todo momento y, por tanto, no se va a ejercer ningún momento de magnitud importante; todas las fuerzas relevantes son longitudinales al eje de rotación del motor.

Por último, se ha verificado que no existen colisiones en el movimiento de la plataforma para terminar el diseño mecánico. En la Figura 5.6 se muestra el diseño 3D de la plataforma completa. En esta ocasión se han añadido al modelo los módulos EyeOF diseñados en el Capítulo 4. Las vistas ortogonales dispuestas en los bordes de la figura muestran movimientos a $\pm 45°$ respecto a la posición inicial. Este rango de movimiento es suficiente para la aplicación que se desea implementar. Además de las colisiones mecánicas deben de tenerse en cuenta el enrutamiento de los cables para las conexiones necesarias en el sistema (conexiones a los módulos EyeOF, conexiones a los motores desde el controlador y conexión desde el controlador al módulo MPU 6050).

5.4. Definición e implementación de la función de control

La Figura 5.7 muestra el resultado de la construcción del diseño mecánico desarrollado en la sección anterior. Las piezas diseñadas se han fabricado en las impresoras 3D del IMSE y las PCBs se han enviado a fabricar a una empresa externa. En ella se integran todos los sistemas descritos y las conexiones necesarias. Ahora se trata de configurar el controlador para adaptarlo a la dinámica que esta presenta. Para tal fin, hay que realizar una serie de pasos en los que se define la configuración de las IMUs, las características de los motores, los parámetros de los reguladores PIDs, entre otras características. Una vez configurado el controlador, la plataforma quedará estabilizada y se pasará a analizar las señales de control que el sistema formado por los módulos EyeOF enviará al controlador `STorM32 BGC` para modificar la posición de la plataforma de forma coherente. Por ende, se tiene que el sistema en su conjunto va a estar formado por dos bucles de control independientes pero relacionados: uno para el control de la posición de la estructura y otro para la aplicación del seguimiento. La Figura 5.8 muestra un diagrama de bloques donde se presentan los dos bucles de control y su relación.

Las secciones que siguen describen el proceso de configuración, estabilización, ajuste y puesta en marcha del sistema.

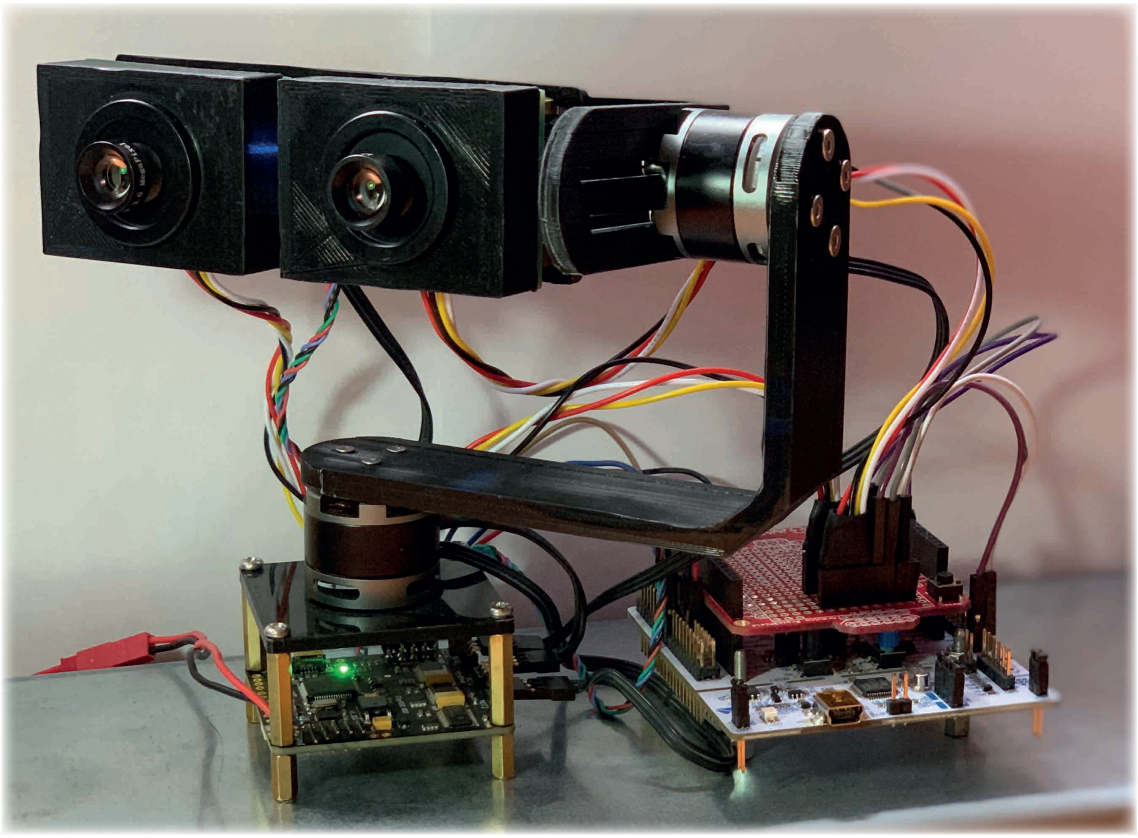

Figura 5.7: Vista de la plataforma construida

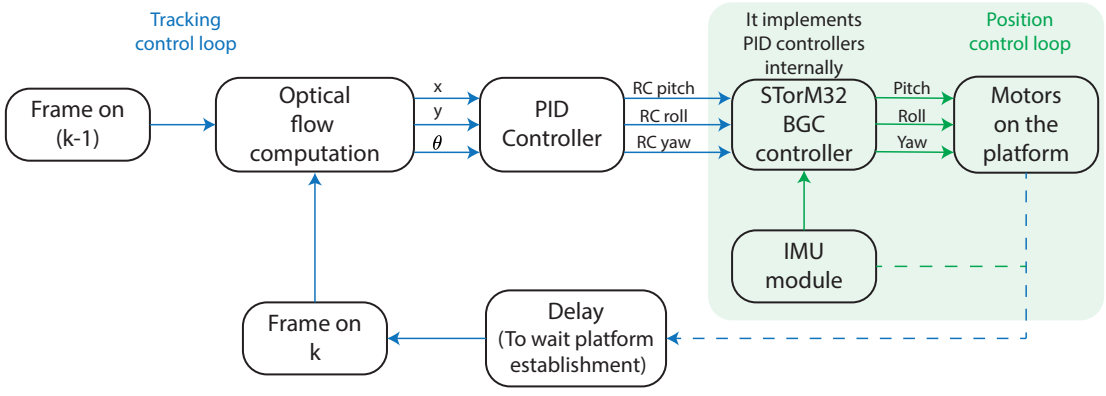

Figura 5.8: Bucles de control implementados en el sistema

5.4.1. Bucle de control para la posición: Configuración del controlador STorM32 BGC y estabilización de la plataforma

Como se comenta en la Sección 5.2 el controlador `STorM32 BGC` dispone de una interfaz que permite configurar diferentes parámetros del controlador. Esta interfaz será la principal herramienta utilizada durante esta sección.

El primer paso previo a la configuración del controlador `STorM32 BGC` consiste en la calibración del módulo MPU 6050 que contiene la IMU que se ha colocado junto a los módulos EyeOF. Mediante la calibración se compensan offsets en la medición de los giróscopos debidos a factores internos al chip y a la propia construcción del módulo en la estructura, y que pueden resultar en una posición de la plataforma errónea. Se sigue el siguiente proceso para completar la calibración:

1. Se desmonta los tornillos traseros al motor de Roll para separar de la plataforma los módulos EyesOF, el motor de Roll y el módulo MPU 6050.

2. Se dispone el conjunto extraído en horizontal en una superficie totalmente plana.

3. En la pestaña *Calibration Acc* de la interfaz se elige la opción *Run 1-point Calibration.*

4. Tras la calibración se mostrarán los parámetros de calibración obtenidos. En este caso, los parámetros obtenidos son los anotados en la Figura 5.9.

5. Se guardan los parámetros en el controlador y se vuelve a montar la plataforma.

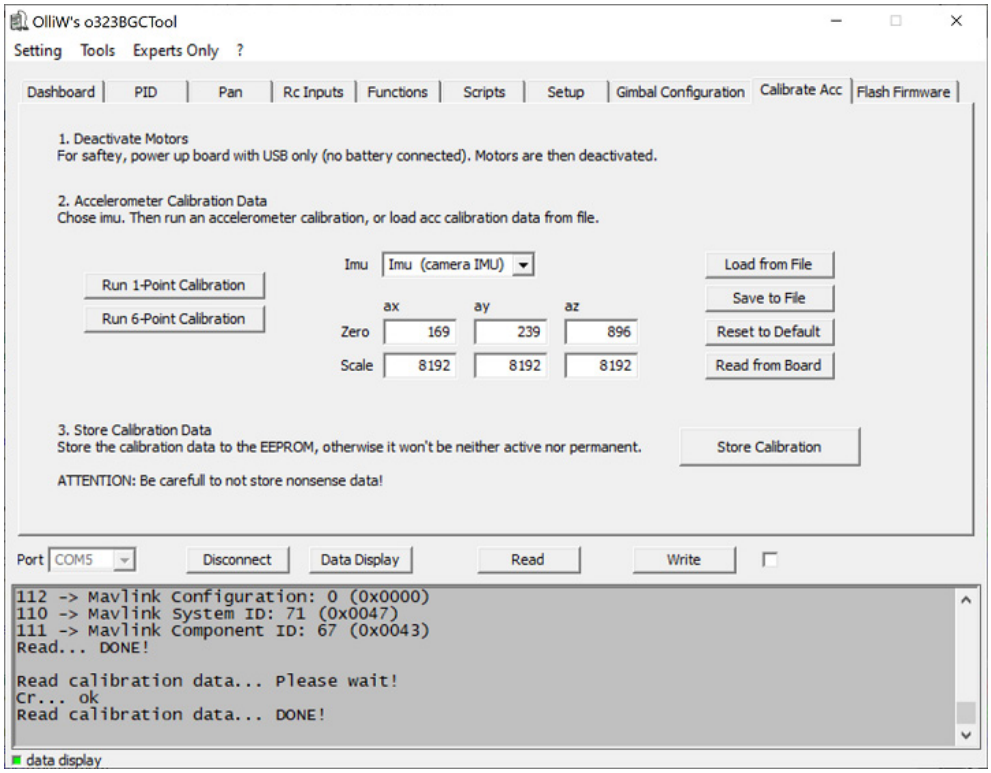

Figura 5.9: Calibración de la IMU contenida en el módulo MPU 6050

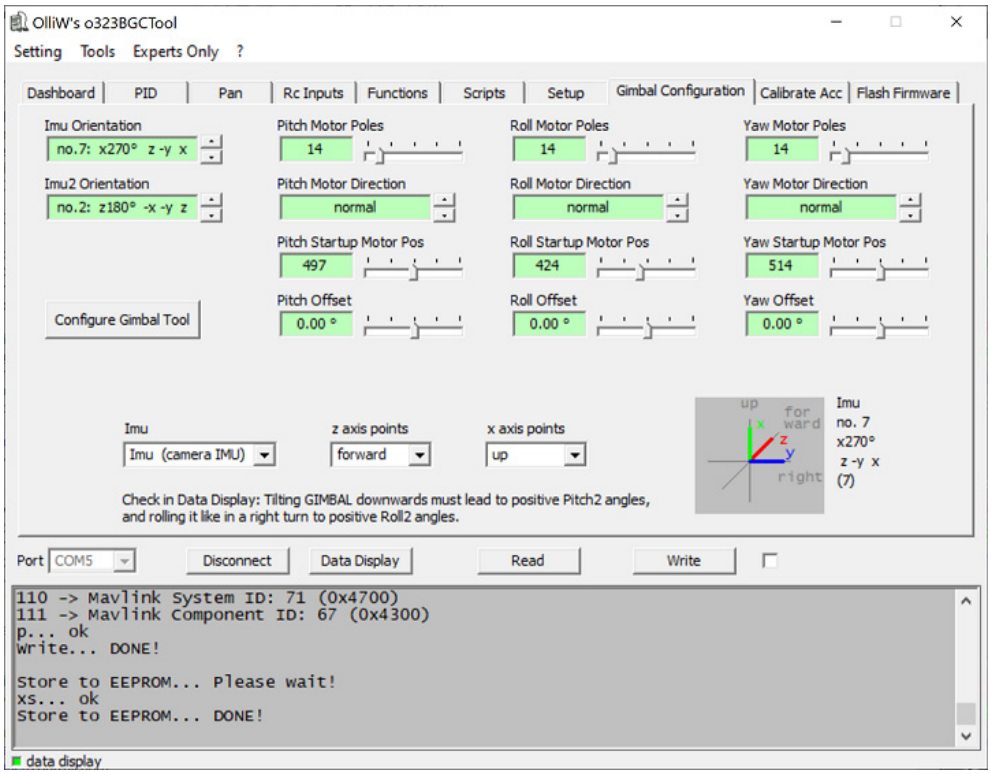

Figura 5.10: Configuración resultado de seguir el proceso guiado de configuración asistida

Puesto que el controlador `STorM32 BGC` es compatible con varias estructuras de gimbal, es necesario configurarlo específicamente para la plataforma que se ha construido. La interfaz del controlador contiene un asistente que hace de guía en el proceso de configuración. El asistente se encuentra disponible a través del botón *Configure Gimbal Tool* en la pestaña *Gimbal configuration* de la interfaz.

En la Figura 5.10 se muestra la configuración resultante de realizar el proceso guiado. Entre los parámetros más importantes en la configuración se encuentran las orientaciones de las IMUs (la montada en el controlador y la contenida en el módulo MPU 6050). Además, quedan especificados el número de polos de los motores, necesarios para generar sus señales de control de forma óptima, y la dirección de rotación positiva para cada uno de los motores.

Configurado el controlador, se sigue con el ajuste de los reguladores PID. El bucle de control relacionado con la posición de la plataforma es el marcado por la zona verde en la Figura 5.8. La función de control se basa en la implementación de 3 reguladores PIDs, uno para cada uno de los ejes (Pitch, Roll y Yaw). Un regulador PID es un controlador cuya ley de control viene definida por el sumatorio de un término Proporcional a su entrada, un término proporcional a la Integral de su entrada y un último término proporcional a la Derivada de su entrada [95, 96]. La entrada al regulador es el error existente entre el punto que se desea alcanzar (*Set Point*) y la salida actual del sistema. En la Figura 5.11 se muestra un esquema de bloques de un regulador PID básico implementado en un sistema de control realimentado.

Los valores para K_P, K_D y K_I de los reguladores PID del controlador son ajustables mediante la pestaña *PID* de la interfaz de configuración. Para ajustar estos parámetros se siguen los siguientes pasos:

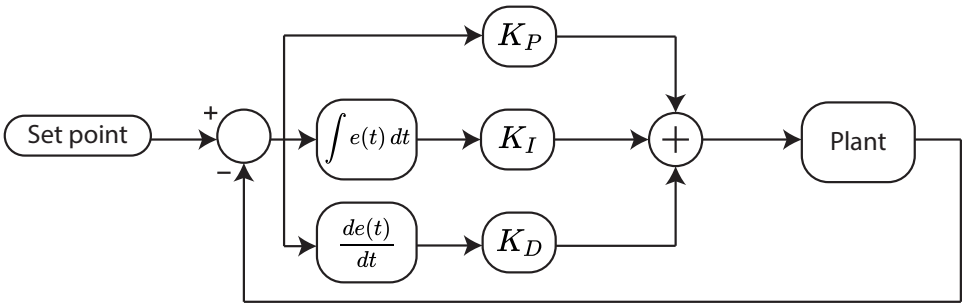

Figura 5.11: Diagrama básico de un controlador PID

1. Desactivar todas las salidas de control excepto la correspondiente al eje de Pitch. Realizar los ajustes de los parámetros del PID relacionado con este eje como sigue.

2. Comenzar a aumentar el valor de K_D hasta que el sistema comience a vibrar a alta frecuencia. A partir de este momento, comenzar a disminuir el valor de K_D hasta que la vibración cese. Comprobar que el sistema no vibra para cualquiera de las posiciones posibles en este eje.

3. Aumentar el valor de K_I al mínimo posible por encima de cero, en el caso del controlador `STorM32 BGC` este valor es 5. Comenzar a aumentar el valor de K_P hasta que el sistema comience a oscilar a baja frecuencia. Una vez llegamos a este punto, disminuir el valor de K_P hasta que la oscilación cese. Comprobar que el sistema no oscila para cualquiera de las posiciones posibles en este eje.

4. Aumentar el valor de K_I hasta que el sistema se vuelva inestable. Esto se notará debido a que la posición del motor en el eje comenzará a variar de manera aleatoria. A partir de este punto, disminuir el valor de K_I hasta alcanzar la estabilidad.

5. A continuación se activa el motor del eje Roll y se realizan los pasos del 2 al 4. Finalmente se activa el motor del eje Yaw y se realizan los pasos del 2 al 4.

En caso de que la plataforma comience a sufrir vibraciones una vez ajustados todos los ejes, habrá que comenzar a disminuir los valores de los parámetros hasta alcanzar el punto óptimo donde la plataforma es estable. Los valores de los parámetros ajustados para la plataforma construida se muestran en la Figura 5.12.

Ha de anotarse que la prioridad es tener un valor de K_I alto, ya que la parte integradora del PID es la principal encargada de corregir el error en la posición de la plataforma. Se prefiere disminuir el valor de K_P y K_D para conseguir aumentar el valor de K_I. Por otro lado, el valor K_P y K_D modifican la respuesta dinámica de la plataforma, que, en este caso, se han ajustado para que las respuestas sean sobreamortiguadas, es decir, que no existan sobreoscilaciones [95, 96] (Ver Subsección 5.4.2).

Otro de los parámetros a tener en cuenta en el ajuste de los reguladores PID es el valor *Motor Vmax* que se corresponde con el valor máximo de tensión que se va a aplicar en las fases de los motores. La modificación de este valor afectará al ajuste realizado en los parámetros del PID, es decir, será necesario reajustar los parámetros si el valor de *Motor Vmax* cambia. Se desea un valor bajo de este parámetro, de otra forma, los motores sufren un calentamiento excesivo y pueden llegar a dañarse.

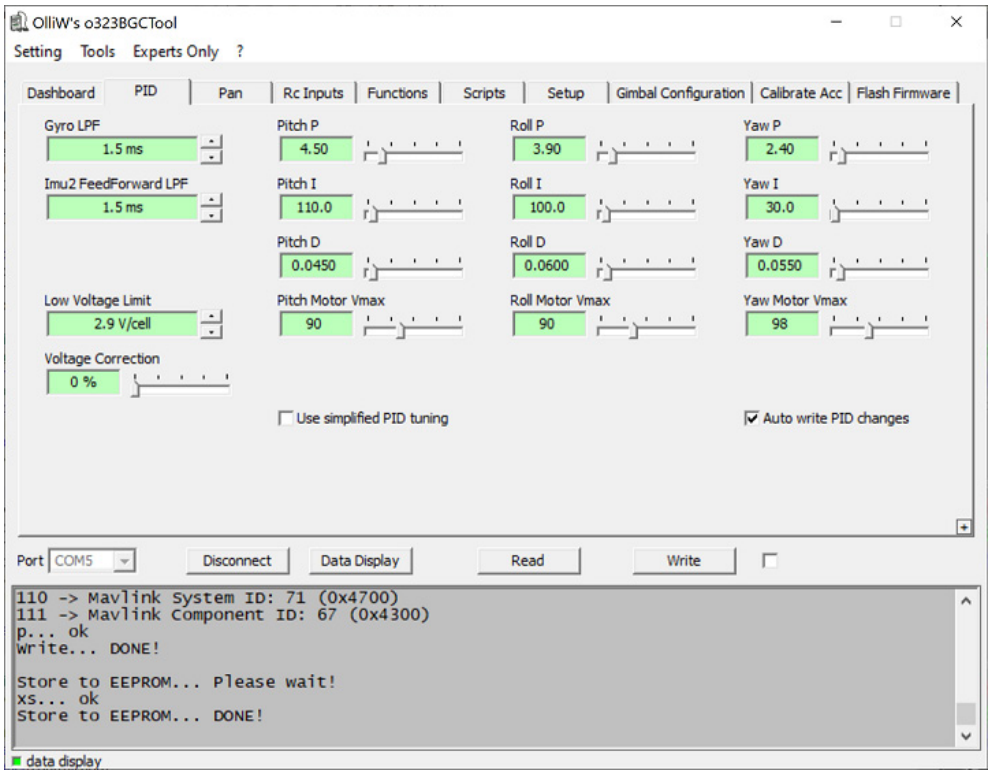

Figura 5.12: Valores de los parámetros de los PIDs ajustados

En este momento, la plataforma queda estabilizada. Los módulos EyeOF montados en ella permanecen en la misma posición aunque la superficie sobre la que se encuentra montado el sistema sea desplazada en cualquier dirección.

5.4.2. Bucle de control para el seguimiento: Configuración de las entradas de control de posición y ajuste de los reguladores PID

Para la implementación del bucle de control relacionado con la aplicación de seguimiento se necesitan estudiar varios factores importantes. En primer lugar, las características de las señales de control que deben enviarse desde el sistema formado por los módulos EyeOF al controlador `STorM32 BGC` y su configuración. En segundo lugar, conocer el tiempo que necesita la plataforma para modificar su posición, es decir, obtener la respuesta dinámica del sistema.

El formato de las señales de Radio Control es PWM. En este formato el valor del ángulo que se desea girar respecto a la posición neutral viene definido por el ciclo de trabajo de una señal cuadrada con período de 20 ms (50 Hz). La posición neutral viene dada por un ciclo de trabajo igual a 7,5 %, es decir, un pulso de $1,5\ ms$. Desde este valor, el ciclo de trabajo puede aumentar o disminuir un 2,5 %, es decir, el rango va desde el 5 % al 10 % (pulso de duración de 1 ms a 2 ms). En este rango de ciclos de trabajo de la señal se encuentran codificadas todas las posiciones posibles en el rango de las señales de Radio Control configurado en el controlador.

La pestaña *RC Inputs* de la interfaz de configuración permite modificar las fuentes y características que van a adoptar las señales de Radio Control enviadas al controlador. En la Figura 5.13 se muestra la configuración adoptada en este caso. A cada eje se le ha asignado una señal que co-

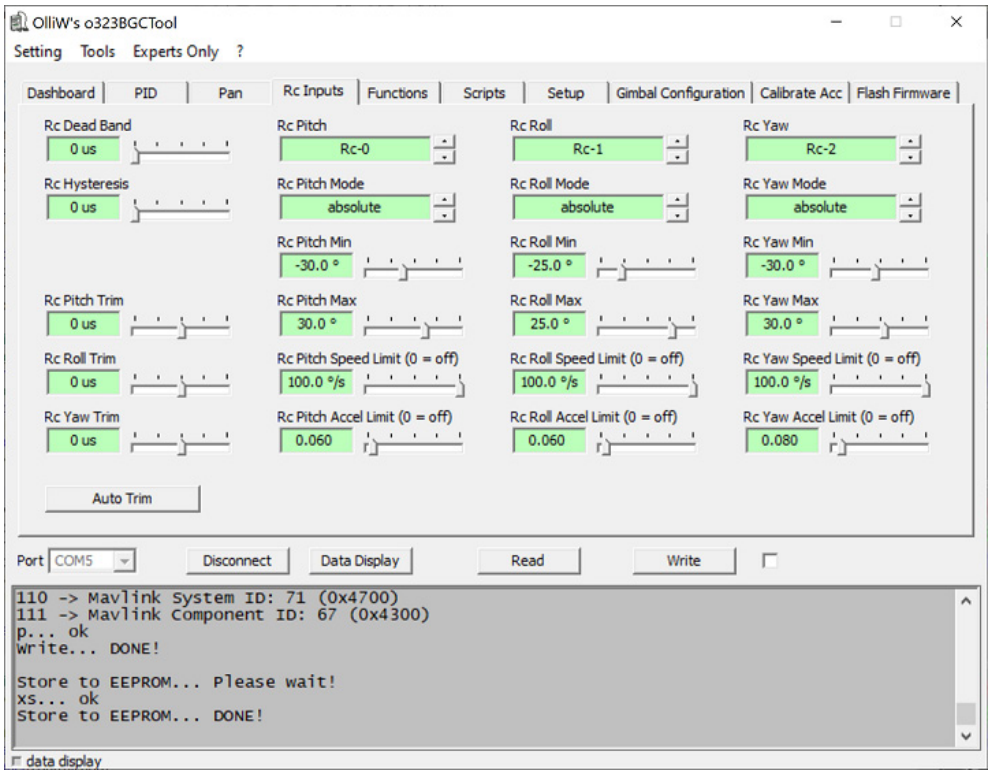

Figura 5.13: Configuración de las señales de Radio Control

rresponde con uno de los pines del conjunto correspondiente a las señales de Radio Control (Ver Figura 5.3): RC-0 para el Pitch, RC-1 para el Roll y RC-2 para el Yaw. Para cada señal se puede observar que existen una serie de parámetros configurables como son el modo, el rango para los ángulos permitidos en cada eje y los limites en la aceleración y la velocidad adoptada en el movimiento de la plataforma. Para este caso particular se tiene:

- El modo de todas las señales será el modo absoluto. En este modo la plataforma volverá a la posición inicial configurada si deja de recibir señales de Radio Control.
- Los límites para los ángulos de rotación desde la posición neutral en cada eje se han ajustado de forma que el cableado de la plataforma no interfiera en la operación del sistema.
- Los límites de velocidad y aceleración se han disminuido lo suficiente para no provocar sobreoscilaciones en el movimiento. Se desea que el movimiento sea lo más rápido posible. Que estos límites puedan ser de menor valor dependerá de los ajustes realizados en los reguladores PID y de las características dinámicas de la plataforma.

Por último, relacionado con la configuración de las señales de control se tienen los parámetros *RC Dead Band* y *RC Hysteresis*. Ambos parámetros se encuentran relacionados con la presencia de jitter en la señal de Radio Control [78]. Este ruido depende de la fuente que genera dicha señal y puesto que en este caso es generada por un microcontrolador con una resolución temporal en el orden de los nanosegundos su valor se ha igualado a cero. Todos los demás parámetros contenidos en este apartado de la interfaz no son utilizados y por ello se dejan con su valor por defecto.

Para conocer el tiempo que debe esperar el sistema desde que se envía una señal de control hasta que la plataforma realiza el cambio de posición indicado se estudia la respuesta al escalón unitario de la plataforma. Para ello, la interfaz de configuración permite grabar los valores de las señales de control entrantes así como la salida de los giróscopos de cada eje de la IMU contenida en el módulo MPU 6050 con una resolución de decenas de milisegundos. En la Figura 5.14 se muestran una gráficas realizadas a partir de los datos obtenidos de la plataforma con el software *Matlab®*. Uno de los parámetros más utilizados para la caracterización de respuestas dinámicas en sistemas de primer orden [95, 96] es la constante de tiempo (τ), que viene dada por el tiempo transcurrido desde que se aplica el estímulo hasta que la señal toma un valor igual al 63 % ($1 - \exp(-t/\tau) = 0{,}63$) de su valor final. Los valores de las constantes de tiempo para cada eje vienen anotados en las gráficas correspondientes. Para un sistema de primer orden se tiene que el tiempo de establecimiento es aproximadamente igual a tres veces la constante de tiempo. En este caso, los tiempos de establecimiento van desde 366 ms para el Pitch hasta 462 ms para el Yaw. El tiempo de retardo que debe adoptar el sistema para esperar a que la plataforma se posicione se encuentra en este rango de tiempos, lo que limita la operación de la plataforma al seguimiento de objetos que se muevan a una velocidad moderada y, que además, se encuentren alejadas de esta, ya que cuanto más alejado esté el objetivo, mayor será la distancia equivalente en el espacio para un movimiento de la plataforma de la misma magnitud. El objetivo es disminuirlo al máximo y se deberá ajustar junto con el regulador PID que se describe a continuación para conseguir la respuesta deseada.

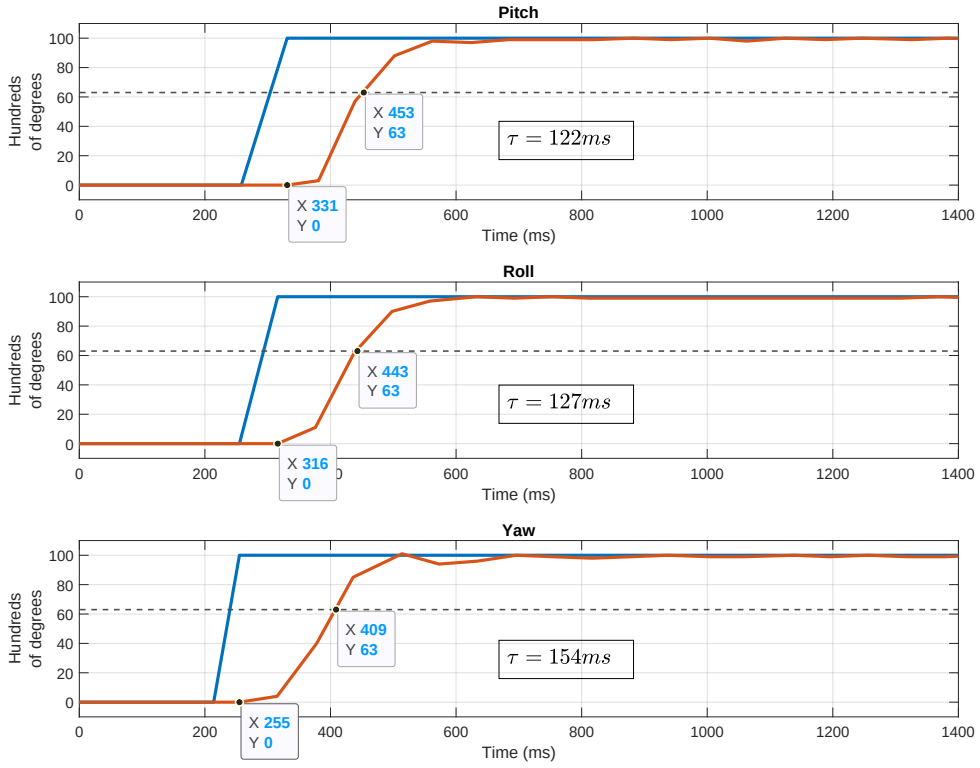

Figura 5.14: Respuestas dinámica de la actitud del sistema a una entrada de tipo escalón unitario

En la función `applyControlLaw` en el Anexo E se muestra el código que implementa los tres reguladores PID, uno para cada eje de la plataforma. La entrada a los reguladores es el valor de

Flujo Óptico calculado, que representa el error entre las imágenes anteriores y las imágenes actuales capturadas por los módulos EyeOF. Se conoce de [97] que la respuesta del sensor de *Flujo Óptico* desarrollado es lineal en un rango de error entre los frames. Los valores de K_P, K_I y K_D para cada regulador implementado vienen dados por los valores de los macros definidos en la zona superior del Anexo E: `PITCH_P`, `PITCH_D`, `PITCH_I`, `ROLL_P`, `ROLL_D`, `ROLL_I`, `YAW_P`, `YAW_D` y `YAW_I`. Estos valores han de ajustarse mediante prueba y error, primer se realiza un preajuste para cada uno de los ejes por separado y finalmente se hace un ajuste fino con los tres ejes activados.

Además, se disponen dos parámetros de ajuste adicionales que permiten modificar la acción de control del regulador PID. Por un lado, los valores `PITCH_WINDUP`, `ROLL_WINDUP` y `YAW_WINDUP` ajustan el fenómeno conocido como *"Integral Windup"* [95] que se da en implementaciones de reguladores PID. Este fenómeno se produce cuando el actuador que modifica la salida del sistema satura, es decir, se encuentra en uno de los extremos de su rango de operación, lo que produce un rápido incremento del término integral del controlador PID que puede terminar desestabilizando al sistema. El valor de ajuste implementado permite poner a cero el valor de la integral cuando su valor alcance el valor indicado en cada parámetro. Por otro lado, se tiene el valor `DELTALIMIT` que indica el valor mínimo a la salida del regulador PID que genera una acción del actuador. Así, señales de control de pequeña magnitud, que pueden dar lugar a vibraciones no deseadas, son evitadas.

Para ayudar en el proceso de configuración y ajuste del sistema en el siguiente capítulo se desarrolla una interfaz de escritorio para monitorizar la información generada por la plataforma y controlar las distintas funciones que permite.

Capítulo 6

Desarrollo de la interfaz de usuario

Las diferentes funcionalidades y medidas que realiza la plataforma se encuentran disponibles a través de una aplicación de escritorio desarrollada bajo la tecnología WPF de Microsoft incluida en las versiones posteriores a .NET Framework 3.0 y .NET Core 3.0 para el sistema operativo de Windows. En las secciones que siguen se desarrollaran las diferentes características que ofrece dicha tecnología, así como el proceso de diseño gráfico y funcional de la aplicación.

6.1. Tecnología WPF y .NET Core

WPF son las siglas de Windows Presentation Fundation y se trata de una API que permite el desarrollo de aplicaciones de escritorio enriquecidas y sofisticadas para el sistema operativo Windows [98]. Entre las principales ventajas de esta herramienta se encuentra la integración con DirectX para el renderizado altamente eficiente de los contenidos gráficos y la división entre el diseño gráfico y el comportamiento de la interfaz que permite la implementación del paradigma Model-View-ViewModel (MVVM) muy útil en el desarrollo de aplicaciones de escritorio.

Por otro lado, en los últimos años Microsoft ha desarrollado una nueva versión de .NET Framework denominada .NET Core cuya principal ventaja es que se encuentra orientado al uso de .NET multiplataforma. Este framework se encuentra en plena fase de evolución y fue hace varios meses cuando incorporaron la tecnología WPF al framework. Actualmente, aunque WPF trabaje bajo .NET Core no tiene capacidad multiplataforma, pero se espera que en el futuro sí que lo sea y por ello se ha decidido utilizar .NET Core en esta ocasión, para poder lanzar esta aplicación en macOS o Linux en un futuro cercano.

6.2. Paradigma de programación MVVM (Model-View-ViewModel)

El paradigma MVVM para aplicaciones WPF fue presentado por el arquitecto de software en Microsoft, Gossman [99] en 2005. Se trata de un patrón de programación software que facilita la partición entre el desarrollo de la parte gráfica y la parte funcional en una aplicación de interfaz de usuario [98]. De esta forma, es posible editar cada una de las partes sin influir en la otra, permitiendo el desarrollo por separado de ambas partes.

Este paradigma se encuentra integrado en WPF y lo componen tres elementos principales como se muestra en la Figura 6.1:

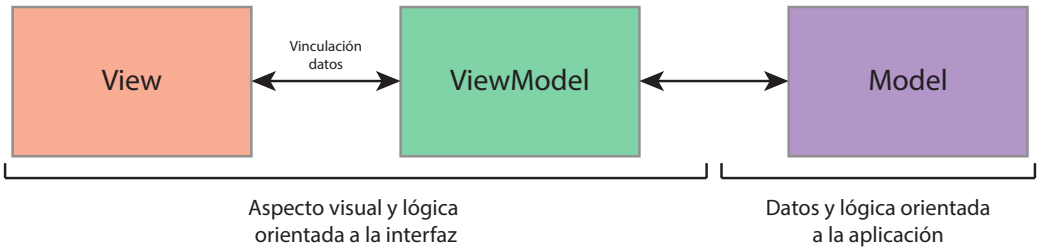

Figura 6.1: MVVM - Bloques funcionales y su relación

- **View**: En este bloque se implementan todos los gráficos de la interfaz y permite la interacción con el usuario. La disposición de cada elemento de la aplicación se define mediante lenguaje XAML [100]. El entorno de desarrollo dispone de herramientas que permiten visualizar en todo momento el resultado del código XAML desarrollado.
- **Model**: Contiene todos los datos y métodos relacionados con el contenido de la aplicación sin relación con la interfaz de usuario. En este bloque se desarrolla en C# [101] y en él se definen las estructura de datos (bases de datos, clases, etc.) y los procesos necesarios para el manejo de dichos datos.
- **ViewModel**: Este bloque es el nexo entre el View y el Model. Por un lado, le expone a la View las diferentes propiedades públicas que contiene el Model. Por otro lado, recibe los distintos eventos generados por la View e implementa diferentes funciones lógicas interactuando con el Model.

Es necesario anotar que la vinculación de los datos entre el View y el ViewModel es implementado bajo la arquitectura de WPF mientras que la llamada a los datos y métodos del Model desde el ViewModel es realizada por el usuario en función del comportamiento que se desee en la aplicación.

6.3. Diseño gráfico de la aplicación

El nombre de la aplicación es 'Eyes OF Gimbal Platform' y permite la visualización de los datos generados por la plataforma, así como algunas funciones de configuración y posicionamiento. Presenta un estilo moderno basado en un toolkit de Google denominado MaterialDesign [102] que ha sido modificado para ser compatible con WPF. En la Figura 6.2 se muestra la vista inicial de la aplicación.

Cada una de las distintas posibilidades que permite la aplicación se habilitarán en función de las funciones habilitadas anteriormente. De esta forma la interacción con el usuario es segura y se prohíbe acciones no permitidas. En la Sección 6.4 se desarrollarán cada una de las funciones y el procedimiento a seguir para su implementación.

6.3.1. Monitorización del frame

En la interfaz se muestran los frames adquiridos por ambos sensores en tiempo real. Para ello se utiliza la clase interna de C# *WriteableBitmap* [103] que permite crear y modificar frames a

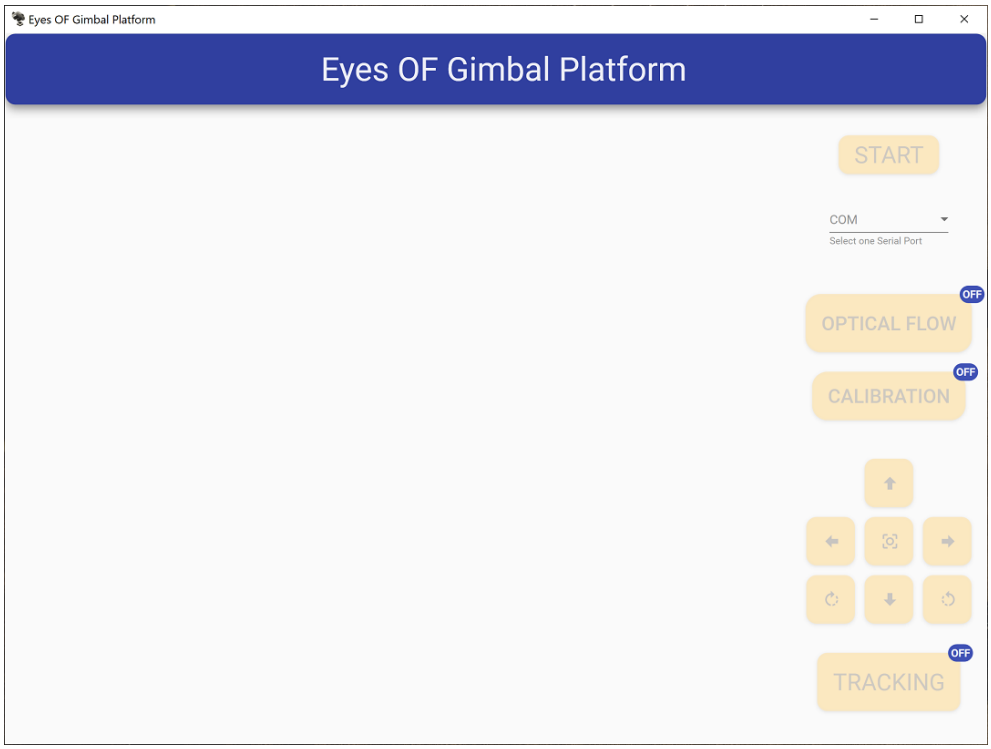

Figura 6.2: Eyes OF Gimbal Platform - vista inicial

nivel de pixel. Esta clase contiene dos buffers, un buffer en segundo plano que contiene información que no se está mostrando y un buffer frontal que contiene la información que se está mostrando actualmente. Un sistema de renderizado interno a la clase copia el contenido del buffer secundario al buffer frontal para su visualización.

Cada uno de los buffers son gestionados mediante dos hilos de procesamiento distintos (Ver figura 6.3). Por un lado, el buffer frontal es gestionado por el hilo de renderización y por otro lado el buffer en segundo plano es gestionado por el hilo de interfaz gráfica (‘*UI Thread*’). El hilo gráfico escribe datos en el buffer secundario y el hilo de renderizado lee el contenido del buffer frontal y los copia a la memoria de video. El intercambio de datos entre el buffer frontal y el buffer secundario se realiza mediante llamadas a métodos que indican qué píxeles del frame han cambiado mediante zonas rectangulares del frame.

En la Figura 6.4 (a) se muestra un frame adquirido por el módulo EyeOF y representado mediante la clase *WriteableBitmap*. En el frame se muestra un rostro, en el que se puede distinguir perfectamente detalles como los ojos, la boca y la nariz, aunque la resolución del frame sea muy baja.

6.3.2. Monitorización del *Flujo Óptico*

La monitorización del *Flujo Óptico* generado por el módulo EyeOF es muy importante puesto que nos permite verificar el correcto funcionamiento de este. En un principio se limitó la representación de esta magnitud mediante valores mostrados en la interfaz, sin embargo, este tipo de representación puede resultar confusa y difícil de interpretar. Por ello se decide representar esta magnitud mediante vectores sobre el frame (ver Figura 6.4 (b)) cuya magnitud es el valor de *Flujo*

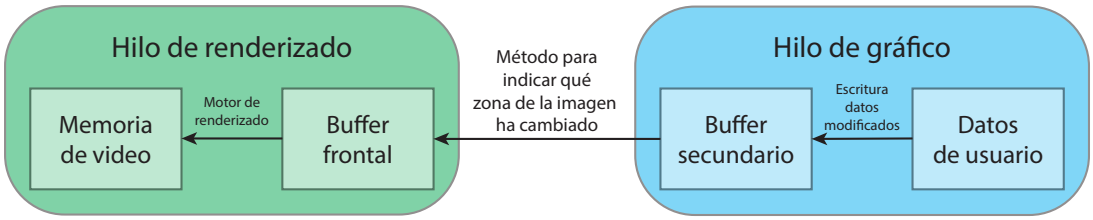

Figura 6.3: Flujo de datos en la clase *WriteableBitmap* de C#

Óptico y además es posible representar su sentido y dirección. De esta forma, la interpretación del *Flujo Óptico* es mucho más rápida y sencilla.

WPF no contiene de forma nativa controles para representar vectores, sin embargo, sí que permite dibujar líneas y figuras geométricas. Esto ha sido lo que ha aprovechado Charles Petzold [104] en la creación de una clase que permite incorporar vectores al código XAML que define la vista de la interfaz. Por otro lado, para suavizar las transiciones entre los distintos valores de *Flujo Óptico* y que el usuario pueda captar correctamente los cambios se implementa un filtro paso bajas sobre el *Flujo Óptico* recibido desde los módulos EyeOF. La función que implementa el filtro se denomina `FilteringOF` y se puede consultar en el Anexo J. Ha de tenerse en cuenta que esto no afecta a la dinámica del *Flujo Óptico* recibido puesto que la frecuencia de muestreo de los vectores en la interfaz es muy superior a la frecuencia de recepción de datos desde los módulos EyeOF, sólo se añaden valores intermedios para enriquecer la representación del *Flujo Óptico*.

(a) (b)

Figura 6.4: Frame de un rostro capturado por el módulo EyeOF y representado mediante la clase *WriteableBitmap*: (a) frame representado sin *Flujo Óptico*; (b) frame representado junto con el *Flujo Óptico*

En la Figura 6.4 (b) se pueden observar dos vectores, uno rojo y otro verde. El vector rojo muestra el resultado del cálculo de *Flujo Óptico*, es decir, cuanto se ha movido el frame actual respecto al frame de referencia siendo el frame actual el origen del vector. De esta forma, en este caso particular, se tiene que el rostro se ha movido a la derecha, que es lo que expresa el vector verde, y que simplemente es un vector con el mismo módulo y dirección pero en sentido contrario al vector rojo. El vector verde, será el que se utilice para controlar el sistema de seguimiento, ya que indica la corrección que se debe realizar en la posición de la plataforma para que el frame captado siga siendo el mismo que el de referencia, es decir, que se siga al objetivo en cuestión.

El código XAML que define las vistas para la monitorización del *Flujo Óptico* y la representación de los frames puede consultarse en el Anexo G .

6.4. Diseño funcional de la aplicación

La aplicación 'Eyes OF Gimbal Platform' permite tanto la monitorización como el control de la plataforma. De esta forma es posible observar la respuesta de los módulos EyeOF en todo momento, realizar tareas de calibración y caracterización y cambiar entre los distintos modos de funcionamiento posibles.

Es necesario seguir un proceso de inicialización de la aplicación (Ver Subsección 6.4.1) por parte del usuario para llegar desde la vista inicial Figura 6.2 a la vista mostrada en la Figura 6.5. Se pueden distinguir dos zonas principales: en la zona de la derecha se encuentran todos los comandos disponibles y en la zona de la izquierda se disponen los distintos datos generados por la plataforma.

6.4.1. Inicialización de la aplicación y comandos disponibles

Antes de lanzar la aplicación, la plataforma deberá ser conectada al PC mediante un puerto USB. Al lanzarla, aparece la ventana mostrada en la Figura 6.2, para comenzar a trabajar con esta es necesario realizar los siguientes pasos previos de configuración:

1. Desplegar la lista de puertos de comunicación (COM) disponibles (3). En ella debe aparecer el puerto COM relacionado con la plataforma, si existe más de uno, deberá comprobarse en *Sistema* → *Administrador de dispositivos* qué puerto COM es el correcto.

2. Seleccionar el puerto COM y pulsar el botón `START` dispuesto justo encima (3).

3. En este punto aparecerán las secciones (1) y (2) sin las medidas de *Flujo Óptico* que aparecen abajo en la Figura 6.5.

4. Para activar las medidas y la representación del *Flujo Óptico* es necesario pulsar el botón `OPTICAL FLOW` (4). Cuando lo pulsemos el indicar de este botón que se encuentra en su esquina superior derecha cambiará a `ON` indicando que esta opción se encuentra activa.

5. Al pulsar el botón `OPTICAL FLOW` (4) también aparecerá la sección de *Flujo Óptico* fusionado (8) que muestra los valores resultado de la fusión de los vectores de *Flujo Óptico* representados en (1) y (2).

Tras realizar todos estos pasos los frames capturados por los módulos EyeOF se representarán en (1) y (2) en tiempo real. Además, de la misma forma, los valores de *Flujo Óptico* recibidos serán mostrados en (1), (2) y (8).

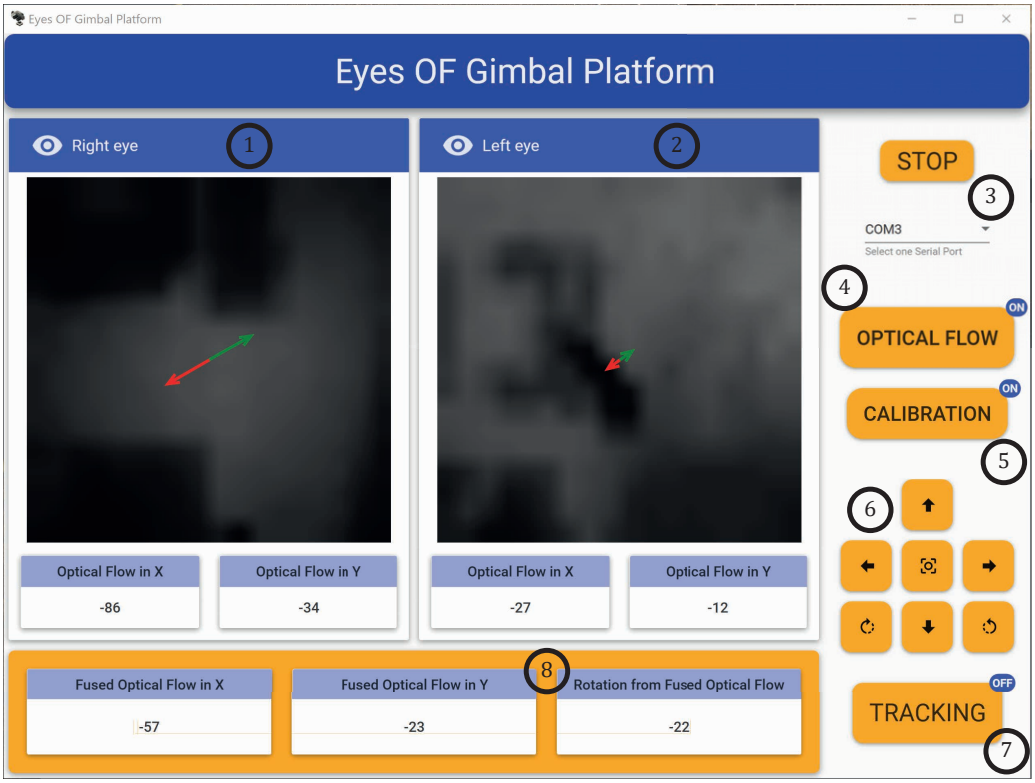

Figura 6.5: Eyes OF Gimbal Platform - vista que contiene todas las funcionalidades posibles de la aplicación e índices para especificar cada una de las zonas relevantes

6.4.2. Posición de la plataforma y función seguimiento

La posición de la plataforma se puede controlar mediante el joystick de la interfaz (6). Cada vez que se presione un botón se moverá una cantidad predefinida en la plataforma hacia la dirección y sentido mostrado en las flechas del botón. El botón central devuelve a la plataforma a su posición inicial y los dos botones a cada lado del botón que mueve la plataforma hacia abajo (↓) permiten rotar la posición de los módulos.

Una vez se tenga posicionada la plataforma tal que los frames adquiridos contengan la información relacionada con el objetivo que se desea seguir se puede activar la función seguimiento pulsando el botón `TRACKING` (7). A partir de este momento la plataforma es controlada por el *Flujo Óptico* y no es posible el control mediante el joystick, una vez deshabilitada la función `TRACKING`, se habilita de nuevo el control mediante joystick.

6.4.3. Función calibración

La función calibración se activa mediante el botón `CALIBRATION` (5). En este modo de funcionamiento el cálculo de *Flujo Óptico* se calcula de forma estática, es decir:

- El frame que sirve de referencia para el *Flujo Óptico*, el adquirido en el instante t_0, se mantiene constante en el tiempo desde que se pulsa el botón `CALIBRATION` (5).

- El frame actual, el adquirido en t, sí cambia en el tiempo, es sobrescrito cada vez que se adquiere un nuevo frame.
- Cuando se pulsa el botón `CALIBRATION` (5) de nuevo se vuelve al funcionamiento de operación normal.

Con esto se consigue que el *Flujo Óptico* se mantenga constante mientras los frames adquiridos por los módulos sean los mismos, es decir, si el entorno es estático, mientras que la plataforma permanezca estática el *Flujo Óptico* es constante. Se recuerda que en el funcionamiento normal del dispositivo, como se desarrolla en la Subsección 4.3.2, el frame tomado en t_0 se actualiza continuamente por el frame adquirido en t y el nuevo frame adquirido es el correspondiente al instante t.

Esta funcionalidad permite observar los valores generados por el *Flujo Óptico* en función del movimiento de la plataforma y el frame adquirido por los módulos. Es posible así, recolectar datos para diferentes movimientos de la plataforma y representarlos en una curva, obteniendo una recta de calibración [97].

Capítulo 7

Conclusiones y líneas de trabajo futuras

Durante este estudio se ha llevado a cabo una serie de tareas que han resultado en la construcción de un sistema de seguimiento basado en sensores de *Flujo Óptico* implementados con cámaras de baja resolución. Para ello se ha estudiado la teoría tras el concepto de *Flujo Óptico*, así como se ha realizado una revisión del estado del arte de algoritmos de *Flujo Óptico* y su aplicación en diferentes sistemas. Ha de destacarse que no se ha encontrado ningún sistema que realice la misma función, aunque sí existe un trabajo [46] en el que se realiza una implementación del mismo algoritmo [49] utilizado en los sensores diseñados, lo que ha servido para comparar y verificar los resultados obtenidos.

Se han llevado a cabo tareas de programación a bajo nivel en *Lenguaje C* orientadas al aumento del rendimiento del código para su implementación en un sistema microcontrolador. Se ha conseguido eliminar todas las holguras temporales en el código, manteniendo ocupado al dispositivo en tareas de procesamiento y adquisición de datos durante todo el tiempo de ejecución del programa. Como consecuencia, ha sido posible la reducción de la frecuencia de reloj en el microcontrolador de 80 MHz, que es el máximo soportado, a 64 MHz, lo que, a su vez, se traduce en un menor consumo del dispositivo.

Se han ejecutado tareas de programación a alto nivel orientadas al desarrollo de aplicaciones de escritorio en lenguaje C# y XAML. La aplicación de escritorio desarrollada tiene un aspecto moderno y permite observar las imágenes captadas junto con los datos generados por la plataforma en tiempo real incluso cuando se activa la función de seguimiento de objetivos, momento en el que la carga computacional en el microcontrolador es más elevada debido a los cálculos requeridos por el regulador PID. Por otro lado, la plataforma contiene el diseño de una librería que permite controlar la plataforma, esta se podría usar de manera directa en el desarrollo de otras aplicaciones que requieran de un sistema de plataforma móvil similar. Dadas sus características, esta aplicación de escritorio podría servir para tareas de aprendizaje e iniciación en temáticas relacionadas con el procesamiento de imágenes.

Han sido afrontadas con éxito tareas de diseño mecánico asistido por software, realizándose no sólo el modelado 3D de todo el sistema, sino también la construcción de la plataforma y su puesta en marcha. Para esto último, ha sido necesario el estudio de la dinámica del sistema construido, como base para la implementación de su control y estabilización.

La función de seguimiento se ha desarrollado a través de la implementación en *Lenguaje C* de un regulador PID, previo al desarrollo numérico y estudio de posibles errores (*"Integral Windup"*, ruido en la señal de control, etc. Ver Subsección 5.4.2) en la implementación numérica del regulador.

La plataforma es capaz de seguir a objetos con velocidad moderada y que se encuentren alejados de ella. Esta limitación se debe principalmente a las características del sensor de imagen elegido, puesto que, aunque la cantidad de datos a transmitir es baja a causa de la resolución de la imagen que presenta, la tasa de transferencia de datos que permite también lo es. Apuntar que este sensor ha sido elegido debido a que no se encuentran sensores de imagen de tan baja resolución en el mercado.

Es necesario anotar que la plataforma que se ha desarrollado es compatible con cualquier tipo de sensor de imagen que necesite de posicionamiento en el espacio tridimensional para seguir trayectorias o cualquier otra tarea de interés. Además, permite estudiar otros algoritmos que necesiten de imágenes con condiciones dinámicas conocidas. La función que se implementa en este estudio, el seguimiento de objetos en movimiento mediante el cálculo de flujo óptico, no es más que un ejemplo particular de aplicación elegida con el objetivo de mostrar el potencial del sistema. Existen un amplio abanico de posibilidades orientadas a la mejora de esta aplicación en particular y a la implementación de otras muchas aplicaciones como extensión de este trabajo.

En el desarrollo de este Trabajo Fin de Máster se han evaluado distintas posibilidades que han generado las siguientes líneas de estudio a desarrollar en el futuro:

- Sustitución de los sensores ADNS2610 por sensores de imagen de baja resolución asíncronos [105, 106, 107, 108, 109] como los desarrollados en el grupo de investigación de Microelectrónica Analógica y de Señal Mixta (TIC-179) del IMSE. Esto incrementaría las prestaciones dinámicas del sistema al disminuir los tiempos de adquisición y procesamiento de la imagen.
- Aplicar procesamiento sobre la imagen (por ejemplo, filtros espaciales y temporales, segmentación, etc.) previo a su inyección al algoritmo de estimación de *Flujo Óptico*. Comprobar si los resultados mejoran en términos de rendimiento computacional y precisión del *Flujo Óptico*.
- La plataforma podría formar parte de un banco de ensayos que sirva para la caracterización de especificaciones dinámicas de los sensores de imagen desarrollados en el grupo de investigación de Microelectrónica Analógica y de Señal Mixta (TIC-179) del IMSE.
- En Farian et al. [110] se desarrolla un sensor que devuelve datos relacionados con la posición del Sol a partir de la imagen adquirida por el sensor. Sería posible adaptar la plataforma, añadir este sensor e implementar el seguimiento del Sol, ya que esta operación se encuentra dentro de las prestaciones que presenta la plataforma diseñada.
- En Leñero-Bardallo et al. [111, 112] se desarrolla un sensor de imagen para detección de incendios mediante la detección de llamas en la imagen. Este sensor podría acoplarse a la plataforma y servir como sistema de vigilancia contra incendios de grandes áreas, ya que, sería posible realizar barridos sobre dichas áreas en busca de llamas.

Bibliografía

[1] C. Mead. Neuromorphic electronic systems. *Proceedings of the IEEE*, 78(10):1629–1636, 1990. doi: 10.1109/5.58356.

[2] Shih-Chii Liu, Tobi Delbruck, Giacomo Indiveri, Adrian Whatley, and Rodney Douglas. *Event-Based Neuromorphic Systems*. John Wiley & Sons, December 2014. ISBN 978-1-118-92762-5.

[3] Kwabena A. Boahen. *Retinomorphic Vision Systems: Reverse Engineering the Vertebrate Retina*. PhD thesis, Cal. Inst. of Tech., Pasadena, California, 1996.

[4] C. Mead and M. A. Mahowald. A silicon model of early visual processing. *New York: Pergamon*, 1988.

[5] Festo. Festo. https://www.festo.com/group/es/cms/index.htm. (last accessed 2020-11-19).

[6] European Union. Human Brain Project. https://www.humanbrainproject.eu/en/. (last accessed 2020-11-28).

[7] Emanuel Diamant. Machine learning: When and where the horses went astray? In Yagang Zhang, editor, *Machine Learning*, chapter 1. IntechOpen, Rijeka, 2010. doi: 10.5772/9156.

[8] Alcides Fonseca and Bruno Cabral. Designing a neural network from scratch for big data powered by multi-node GPUs. In Valentina Emilia Balas, Sanjiban Sekhar Roy, Dharmendra Sharma, and Pijush Samui, editors, *Handbook of Deep Learning Applications*, pages 1–19. Springer International Publishing, Cham, 2019. ISBN 978-3-030-11479-4. doi: 10.1007/978-3-030-11479-4.

[9] John E. Dowling. *The Retina : An Approachable Part of the Brain*. Belknap Press of Harvard University Press, Cambrigde, Mass, 1987. ISBN 0-674-76680-6.

[10] James J Gibson. *The Perception of the Visual World*. Houghton Mifflin, Boston, 1950.

[11] HLF von Helmholtz. *Treatise on Physiological Optics (Translated by JP Southall, 1925)*. NY Dover Publications, Dover, New York, 1910.

[12] Jan J. Koenderink. Optic flow. *Vision Research*, 26(1):161–179, 1986. ISSN 0042-6989. doi: 10.1016/0042-6989(86)90078-7.

[13] Berthold K.P. Horn and Brian G. Schunck. Determining optical flow. *Artificial Intelligence*, 17(1):185–203, 1981. ISSN 0004-3702. doi: 10.1016/0004-3702(81)90024-2.

[14] Michael J. Black and P. Anandan. The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields. *Computer Vision and Image Understanding*, 63(1):75–104, 1996. ISSN 1077-3142. doi: 10.1006/cviu.1996.0006.

[15] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, pages 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc.

[16] Andrés Bruhn, Joachim Weickert, and Christoph Schnörr. Lucas/Kanade Meets Horn/Schunck: Combining Local and Global Optic Flow Methods. *International Journal of Computer Vision*, 61(3):211–231, February 2005. ISSN 1573-1405. doi: 10.1023/B:VISI.0000045324.43199.43.

[17] Hubert Eichner, Maximilian Joesch, Bettina Schnell, Dierk F. Reiff, and Alexander Borst. Internal structure of the fly elementary motion detector. *Neuron*, 70(6):1155–1164, 2011. ISSN 0896-6273. doi: 10.1016/j.neuron.2011.03.028.

[18] Franz Weber, Christian K. Machens, and Alexander Borst. Spatiotemporal response properties of optic-flow processing neurons. *Neuron*, 67(4):629–642, 2010. ISSN 0896-6273. doi: 10.1016/j.neuron.2010.07.017.

[19] Michael S. Drews, Aljoscha Leonhardt, Nadezhda Pirogova, Florian G. Richter, Anna Schuetzenberger, Lukas Braun, Etienne Serbe, and Alexander Borst. Dynamic signal compression for robust motion vision in flies. *Current Biology*, 30(2):209 – 221.e8, 2020. ISSN 0960-9822. doi: 10.1016/j.cub.2019.10.035.

[20] Barbara Webb. Insect behaviour: Controlling flight altitude with optic flow. *Current Biology*, 17(4):R124 – R125, 2007. ISSN 0960-9822. doi: 10.1016/j.cub.2006.12.008.

[21] Simon B. Laughlin. Visual motion: Dendritic integration makes sense of the world. *Current Biology*, 9(1):R15 – R17, 1999. ISSN 0960-9822. doi: 10.1016/S0960-9822(99)80035-9.

[22] Mandyam V Srinivasan. Visual control of navigation in insects and its relevance for robotics. *Current Opinion in Neurobiology*, 21(4):535–543, 2011. ISSN 0959-4388. doi: 10.1016/j.conb.2011.05.020.

[23] Mandyam V Srinivasan. Honeybees as a model for the study of visually guided flight, navigation, and biologically inspired robotics. *Physiological reviews*, 91(2):413–460, April 2011. ISSN 0031-9333. doi: 10.1152/physrev.00005.2010.

[24] Emily Baird, Mandyam V. Srinivasan, Shaowu Zhang, and Ann Cowling. Visual control of flight speed in honeybees. *The Journal of experimental biology*, 208(Pt 20):3895–3905, October 2005. ISSN 0022-0949 0022-0949. doi: 10.1242/jeb.01818.

[25] Nicolas Franceschini, Franck Ruffier, and Julien Serres. A bio-inspired flying robot sheds light on insect piloting abilities. *Current Biology*, 17(4):329–335, 2007. ISSN 0960-9822. doi: 10.1016/j.cub.2006.12.032.

[26] Franck Ruffier and Nicolas Franceschini. Optic flow regulation: The key to aircraft automatic guidance. *Robotics and Autonomous Systems*, 50(4):177–194, 2005. ISSN 0921-8890. doi: 10.1016/j.robot.2004.09.016.

[27] Marko Pudas, Stéphane Viollet, Franck Ruffier, Arvi Kruusing, Stéphane Amic, Seppo Leppävuori, and Nicolas Franceschini. A miniature bio-inspired optic flow sensor based on low temperature co-fired ceramics (LTCC) technology. *Sensors and Actuators A: Physical*, 133(1):88–95, 2007. ISSN 0924-4247. doi: 10.1016/j.sna.2006.03.013.

[28] J. L. Barron, D. J. Fleet, and S. S. Beauchemin. Performance of optical flow techniques. *International Journal of Computer Vision*, 12(1):43–77, February 1994. ISSN 1573-1405. doi: 10.1007/BF01420984.

[29] Simon Baker, Daniel Scharstein, J. P. Lewis, Stefan Roth, Michael J. Black, and Richard Szeliski. A Database and Evaluation Methodology for Optical Flow. *International Journal of Computer Vision*, 92(1):1–31, March 2011. ISSN 0920-5691, 1573-1405. doi: 10.1007/s11263-010-0390-2.

[30] D. W. Murray and B. F. Buxton. Scene segmentation from visual motion using global optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9 (2):220–228, 1987. doi: 10.1109/TPAMI.1987.4767896.

[31] M. J. Black and P. Anandan. Robust dynamic motion estimation over time. In *Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 296–302, 1991. doi: 10.1109/CVPR.1991.139705.

[32] Thomas Brox, Andrés Bruhn, Nils Papenberg, and Joachim Weickert. High Accuracy Optical Flow Estimation Based on a Theory for Warping. In Tomás Pajdla and Jiří Matas, editors, *Computer Vision - ECCV 2004*, pages 25–36, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-24673-2.

[33] Ho Yub Jung, Kyoung Mu Lee, and Sang Uk Lee. Toward Global Minimum through Combined Local Minima. In David Forsyth, Philip Torr, and Andrew Zisserman, editors, *Computer Vision – ECCV 2008*, pages 298–311, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-88693-8.

[34] V. Lempitsky, S. Roth, and C. Rother. FusionFlow: Discrete-continuous optimization for optical flow estimation. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008. doi: 10.1109/CVPR.2008.4587751.

[35] He Zheng, Hanbo Chen, Junzhou Huang, Xuzhi Li, Xiao Han, and Jianhua Yao. Polyp Tracking in Video Colonoscopy Using Optical Flow With an On-The-Fly Trained CNN. In *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)*, pages 79–82, Venice, Italy, April 2019. IEEE. ISBN 978-1-5386-3641-1. doi: 10.1109/ISBI.2019.8759180.

[36] C. Karthika Pragadeeswari, G. Yamuna, and G. Yasmin Beham. Optical flow detection and tracking of gauzes and cancer cells. *Materials Today: Proceedings*, 2020. ISSN 2214-7853. doi: 10.1016/j.matpr.2020.05.525.

[37] S. Periyasamy, M. Kleedehn, and P. Laeseke. Abstract No. 683 An optical flow-based technique for localizing arterial bleeds on diagnostic angiograms. *Journal of Vascular and Interventional Radiology*, 31(3, Supplement):S294, 2020. ISSN 1051-0443. doi: 10.1016/j.jvir.2019.12.742.

[38] Prasad. D. Garje, M.S. Nagmode, and Kiran. C. Davakhar. Optical Flow Based Violence Detection in Video Surveillance. In *2018 International Conference On Advances in Communication and Computing Technology (ICACCT)*, pages 208–212, Sangamner, February 2018. IEEE. ISBN 978-1-5386-0926-2. doi: 10.1109/ICACCT.2018.8529501.

[39] Amira Ben Mabrouk and Ezzeddine Zagrouba. Spatio-temporal feature using optical flow based distribution for violence detection. *Pattern Recognition Letters*, 92:62–67, 2017. ISSN 0167-8655. doi: 10.1016/j.patrec.2017.04.015.

[40] Chetan G. Kagalagomb and Sunanda Dixit. Tracking of Soccer Players Using Optical Flow. In Deepak Gupta, Ashish Khanna, Siddhartha Bhattacharyya, Aboul Ella Hassanien, Sameer Anand, and Ajay Jaiswal, editors, *International Conference on Innovative Computing and Communications*, pages 117–129, Singapore, 2021. Springer Singapore. ISBN 978-981-15-5148-2.

[41] Bo Du, Shihan Cai, and Chen Wu. Object Tracking in Satellite Videos Based on a Multiframe Optical Flow Tracker. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(8):3043–3055, August 2019. ISSN 1939-1404, 2151-1535. doi: 10.1109/JSTARS.2019.2917703.

[42] Hao Su, Yaran Chen, Shiwen Tong, and Dongbin Zhao. Real-time multiple object tracking based on optical flow. In *2019 9th International Conference on Information Science and Technology (ICIST)*, pages 350–356, Hulunbuir, China, August 2019. IEEE. ISBN 978-1-72812-106-2. doi: 10.1109/ICIST.2019.8836764.

[43] Yuanlu Wu, Minghao Chen, Yan Wo, and Guoqiang Han. Video smoke detection base on dense optical flow and convolutional neural network. *Multimedia Tools and Applications*, October 2020. ISSN 1573-7721. doi: 10.1007/s11042-020-09870-x.

[44] Tobi Delbruck and Manuel Lang. Robotic goalie with 3 ms reaction time at 4 % CPU load using event-based dynamic vision sensor. *Frontiers in Neuroscience*, 7:223, 2013. ISSN 1662-453X. doi: 10.3389/fnins.2013.00223.

[45] T. Delbrück, B. Linares-Barranco, E. Culurciello, and C. Posch. Activity-driven, event-based vision sensors. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 2426–2429, 2010. doi: 10.1109/ISCAS.2010.5537149.

[46] R. Pericet-Camara, G. Bahi-Vila, J. Lecoeur, and D. Floreano. Miniature artificial compound eyes for optic-flow-based robotic navigation. In *2014 13th Workshop on Information Optics (WIO)*, pages 1–3, 2014. doi: 10.1109/WIO.2014.6933290.

[47] Dario Floreano, Ramon Pericet-Camara, Stéphane Viollet, Franck Ruffier, Andreas Brückner, Robert Leitel, Wolfgang Buss, Mohsine Menouni, Fabien Expert, Raphaël Juston, Michal Karol Dobrzynski, Geraud LEplattenier, Fabian Recktenwald, Hanspeter A. Mallot, and Nicolas Franceschini. Miniature curved artificial compound eyes. *Proceedings of the National Academy of Sciences*, 110(23):9267–9272, 2013. ISSN 0027-8424. doi: 10.1073/pnas.1219068110.

[48] Microchip Technology. dsPIC33F Family Data Sheet, 2005.

[49] M. V. Srinivasan. An image-interpolation technique for the computation of optic flow and egomotion. *Biological Cybernetics*, 71(5):401–415, 1994. ISSN 03401200. doi: 10.1007/BF00198917.

[50] Jl L Barron and Na a Thacker. Tutorial: Computing 2D and 3D optical flow. *Imaging Science and Biomedical Engineering Division, Medical School, University of Manchester*, (2004):1–12, 2005.

[51] James Gibson. *The Senses Considered as Perceptual Systems*. Westport, Conn, June 1983. ISBN 978-0-313-23961-8.

[52] James J. Gibson. On the analysis of change in the optic array. *Scandinavian Journal of Psychology*, 18(1):161–163, 1977. ISSN 1467-9450. doi: 10.1111/j.1467-9450.1977.tb00272.x.

[53] Riyanto Sigit and Eva Rochmawati. Segmentation echocardiography video using B-Spline and optical flow. In *2016 International Conference on Knowledge Creation and Intelligent Computing (KCIC)*, pages 226–231, Manado, Indonesia, November 2016. IEEE. ISBN 978-1-5090-5231-8. doi: 10.1109/KCIC.2016.7883651.

[54] Youssef Zinbi, Youssef Chahir, and Abder Elmoataz. Moving object Segmentation; using optical flow with active contour model. In *2008 3rd International Conference on Information and Communication Technologies: From Theory to Applications*, pages 1–5, Damascus, Syria, April 2008. IEEE. ISBN 978-1-4244-1751-3. doi: 10.1109/ICTTA.2008.4530112.

[55] Yuemei Zhu, Yan Song, Xin Zhang, Pengfei Lv, Guangliang Li, Bo He, and Tianhong Yan. Segmentation of Underwater Object in Videos. In *2018 OCEANS - MTS/IEEE Kobe Techno-Oceans (OTO)*, pages 1–4, Kobe, May 2018. IEEE. ISBN 978-1-5386-1654-3. doi: 10.1109/OCEANSKOBE.2018.8559112.

[56] Juhyoung Lee, Changhyeon Kim, Sungpill Choi, Dongjoo Shin, Sanghoon Kang, and Hoi-Jun Yoo. A 46.1 fps Global Matching Optical Flow Estimation Processor for Action Recognition in Mobile Devices. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, Florence, May 2018. IEEE. ISBN 978-1-5386-4881-0. doi: 10.1109/ISCAS.2018.8351177.

[57] Guillermo Botella, Antonio Garcia, Manuel Rodriguez-Alvarez, Eduardo Ros, Uwe Meyer-Baese, and María C. Molina. Robust Bioinspired Architecture for Optical-Flow Computation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(4):616–629, April 2010. ISSN 1063-8210, 1557-9999. doi: 10.1109/TVLSI.2009.2013957.

[58] Y. Li, X. Chen, and M. Yang. Optical flow based solar irradiance forecasting in satellite images. In *2019 IEEE International Conference on Real-Time Computing and Robotics (RCAR)*, pages 442–447, 2019. doi: 10.1109/RCAR47638.2019.9043950.

[59] The MIT Press. Experiments in the Machine Interpretation of Visual Motion — The MIT Press. https://mitpress.mit.edu/books/experiments-machine-interpretation-visual-motion. (last accessed 2020-10-22).

[60] Berthold Horn and B Schunck. "Determining optical flow". *Artificial Intelligence*, 17(1-2): 185–203, 1981. ISSN 00043702. doi: 10.1016/0004-3702(93)90173-9.

[61] M. I. Sereno and M. E. Sereno. Learning to discriminate senses of rotation and dilation with a Hebb rule. In *Invest. Ophthal. Vis. Sci., Abstr*, volume 31, page 528, 1990.

[62] Vicki Bruce. *Visual Perception : Physiology, Psychology, & Ecology*. Psychology Press, Hove, [England] ;, 4th ed. edition, 2010. ISBN 0-203-42724-6.

[63] Andrew B. Watson and Albert J. Ahumada. Model of human visual-motion sensing. *JOSA A*, 2(2):322–342, 1985.

[64] David J. Heeger. Optical flow using spatiotemporal filters. *International journal of computer vision*, 1(4):279–302, 1988.

[65] David J. Fleet and Allan D. Jepson. Computation of component image velocity from local phase information. *International journal of computer vision*, 5(1):77–104, 1990.

[66] Ajit Singh. An estimation-theoretic framework for image-flow computation. In *Image Understanding Workshop: Proceedings of a Workshop Held at Pittsburgh, Pennsylvania, September 11-13, 1990*, page 314. Morgan Kaufmann Pub, 1990.

[67] Padmanabhan Anandan. *Measuring Visual Motion from Image Sequences*. PhD thesis, University of Massachusetts Amherst, 1987.

[68] C. Bandera and P.D. Scott. Foveal machine vision systems. In *Conference Proceedings., IEEE International Conference on Systems, Man and Cybernetics*, pages 596–599, Cambridge, MA, USA, 1989. IEEE. doi: 10.1109/ICSMC.1989.71367.

[69] F. Robert and E. Dinet. An image filtering process based on foveal mechanism simulation. In *Conference Record of the Thirty-First Asilomar Conference on Signals, Systems and Computers (Cat. No.97CB36136)*, volume 2, pages 1725–1729, Pacific Grove, CA, USA, 1997. IEEE Comput. Soc. ISBN 978-0-8186-8316-9. doi: 10.1109/ACSSC.1997.679197.

[70] Avago Technologies. Data Sheet: Optical sensor ADNS2610, September 2008.

[71] ST Microelectronics. Data Sheet: Ultra-low-power Arm® Cortex®-M4 32-bit MCU+FPU, 100DMIPS, up to 1MB Flash, 128 KB SRAM, USB OTG FS, LCD, ext. SMPS, June 2019.

[72] *CCTV Surveillance*. Elsevier, 1995. ISBN 978-0-7506-9028-7. doi: 10.1016/C2009-0-25247-0.

[73] F. Leens. An introduction to I2C and SPI protocols. *IEEE Instrumentation Measurement Magazine*, 12(1):8–13, February 2009. ISSN 1941-0123. doi: 10.1109/MIM.2009.4762946.

[74] Scott Campbell. Basics of the SPI Communication Protocol. https://www.circuitbasics.com/basics-of-the-spi-communication-protocol/, February 2016. (last accessed 2020-10-30).

[75] DJI. DJI Phantom 4 Pro V2.0 - Drones profesionales - DJI. https://www.dji.com/es/phantom-4-pro-v2. (last accessed 2020-11-04).

[76] ST Microelectronics. STEVAL-GMBL02V1. https://www.st.com/en/evaluation-tools/steval-gmbl02v1.html. (last accessed 2020-11-04).

[77] BaseCam. BaseCam SimpleBGC 32-bit : BaseCam Electronics. https://www.basecamelectronics.com/simplebgc32bit/. (last accessed 2020-11-04).

[78] OlliW. STorM32-BGC Wiki. http://www.olliw.eu/storm32bgc-wiki/Main_Page. (last accessed 2020-11-04).

[79] Peter Moreton. *Industrial Brushless Servomotors*. Newnes Power Engineering Series. Newnes, Oxford, 2000. ISBN 1-281-05128-4.

[80] Irving M. Gottlieb. *Practical Electric Motor Handbook*. Newnes, Oxford ;, 1997. ISBN 1-281-30864-1.

[81] Piotr Wach. *Brushless DC Motor Drives (BLDC)*, pages 281–380. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-20221-6 978-3-642-20222-3. doi: 10.1007/978-3-642-20222-3_4.

[82] ST Microelectronics. Data Sheet: Mainstream Performance line, Arm® Cortex®-M3 MCU with 256 Kbytes of Flash memory, 72 MHz CPU, motor control, USB and CAN, December 2018.

[83] Texas Instruments. Data Sheet: DRV8313 2.5-A Triple 1/2-H BridgeDriver, October 2012.

[84] Hong Cheng. *Autonomous Intelligent Vehicles*. Springer London, London, 2011. ISBN 978-1-4471-2279-1 978-1-4471-2280-7. doi: 10.1007/978-1-4471-2280-7.

[85] Aboelmagd Noureldin, Tashfeen B. Karamat, and Jacques Georgy. *Fundamentals of Inertial Navigation, Satellite-Based Positioning and Their Integration*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-30465-1 978-3-642-30466-8. doi: 10.1007/978-3-642-30466-8.

[86] TDK. MPU-6050 Six-Axis (Gyro + Accelerometer) MEMS MotionTracking™ Devices. https://invensense.tdk.com/products/motion-tracking/6-axis/mpu-6050/. (last accessed 2020-11-08).

[87] Haoran Wen. *Toward Inertial-Navigation-on-Chip: The Physics and Performance Scaling of Multi-Degree-of-Freedom Resonant MEMS Gyroscopes*. Springer Theses. Springer International Publishing, Cham, 2019. ISBN 978-3-030-25469-8 978-3-030-25470-4. doi: 10.1007/978-3-030-25470-4.

[88] Manon Kok, Jeroen D. Hol, and Thomas B. Schön. Using inertial sensors for position and orientation estimation. *Foundations and Trends® in Signal Processing*, 11(1-2):1–153, 2017. ISSN 1932-8346. doi: 10.1561/2000000094.

[89] Nathaniel Pinckney. Pulse-width modulation for microcontroller servo control. *IEEE potentials*, 25(1):27–29, 2006.

[90] Horizon Hobby. Specification for Spektrum Remote Receiver Interfacing. Enabling Use of Spektrum Remotes in Third-Party Products, April 2016.

[91] ARM mbed. Futaba S-BUS controlled by mbed. https://os.mbed.com/users/Digixx/notebook/futaba-s-bus-controlled-by-mbed/, March 2012. (last accessed 2020-11-09).

[92] MH and Ralf Helbing. Technical Specification Document: HoTT SUMD Data Protocol, December 2012.

[93] Mathilde. Berchon. *La Impresión 3D : Guía Definitiva Para Makers, Diseñadores, Estudiantes, Profesionales, Artistas y Manitas En General*. Editorial Gustavo Gili, Barcelona, 2016. ISBN 84-252-2855-7.

[94] Evan G. Hemingway and Oliver M. O'Reilly. Perspectives on Euler angle singularities, gimbal lock, and the orthogonality of applied forces and applied moments. *Multibody System Dynamics*, 44(1):31–56, September 2018. ISSN 1573-272X. doi: 10.1007/s11044-018-9620-0.

[95] Su Whan Sung. *Process Identification and PID Control*. Process Identification and Proportional-Integral-Derivative Control. John Wiley, Singapore ;, 1st edition edition, 2009. ISBN 1-282-38214-4. doi: 10.1002/9780470824122.

[96] Katsuhiko Ogata. *Ingeniería de Control Moderna*. Pearson Educación, Madrid, 4ª ed. edition, 2003. ISBN 84-205-3678-4.

[97] Rafael de la Rosa-Vidal, J. M. Guerrero-Rodriguez, and J. A. Lenero-Bardallo. Live Demonstration: A Tracking System Based on a Real-Time Bio-Inspired Optical Flow Sensor. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–1, Sevilla, October 2020. IEEE. ISBN 978-1-72813-320-1. doi: 10.1109/ISCAS45731.2020.9181258.

[98] Pavel Yosifovich. *Windows Presentation Foundation 4.5 Cookbook*. Packt Publishing, Birmingham, 2012. ISBN 9781283637404.

[99] John Gossman. Introduction to Model/View/ViewModel pattern for building WPF apps. https://docs.microsoft.com/en-us/archive/blogs/johngossman/introduction-to-modelviewviewmodel-pattern-for-building-wpf-apps, August 2005. (last accessed 2020-10-30).

[100] Ashish Ghoda and Mamta Dalal. *XAML Developer Reference*. Pearson Education, December 2011. ISBN 978-0-7356-6809-6.

[101] Anders Hejlsberg, Scott Wiltamuth, and Peter Golde. *C# Language Specification*. Addison-Wesley Longman Publishing Co., Inc., USA, 2003. ISBN 978-0-321-15491-0.

[102] Material Design In XAML Toolkit. http://materialdesigninxaml.net. (last accessed 2020-10-13).

[103] Microsoft Documentation. WriteableBitmap Class (System.Windows.Media.Imaging). https://docs.microsoft.com/en-us/dotnet/api/system.windows.media.imaging.writeablebitmap. (last accessed 2020-10-14).

[104] Charles Petzold. Lines with Arrows. http://www.charlespetzold.com/blog/2007/04/191200.html. (last accessed 2020-10-14).

[105] Juan A. Leñero-Bardallo, Ricardo Carmona-Galán, and Ángel Rodríguez-Vázquez. A bio-inspired vision sensor with dual operation and readout modes. *Sensors Journal, IEEE*, 16 (2):1–14, January 2016. ISSN 1530-437X. doi: 10.1109/JSEN.2015.2483898.

[106] Juan A. Leñero-Bardallo, Ricardo Carmona-Galán, and Ángel Rodríguez-Vázquez. A high dynamic range image sensor with linear response based on asynchronous event detection. In *22nd European Conference on Circuit Theory and Design, ECCTD 2015*, pages 1–4, August 2015. doi: 10.1109/ECCTD.2015.7300079.

[107] Juan A. Leñero-Bardallo, D.H. Bryn, and P. Häfliger. Bio-inspired asynchronous pixel event tricolor vision sensor. *Biomedical Circuits and Systems, IEEE Transactions on*, 8(3):345–357, June 2014. ISSN 1932-4545. doi: 10.1109/TBCAS.2013.2271382.

[108] Juan A. Leñero-Bardallo and P. Häfliger. A dual operation mode bio-inspired vision sensor. In *Biomedical Circuits and Systems Conference (BioCAS), 2013 IEEE*, pages 310–313, October 2013. doi: 10.1109/BioCAS.2013.6679701.

[109] J. A. Leñero-Bardallo, T. Serrano-Gotarredona, and Bernabe Linares-Barranco. A 3.6s latency asynchronous frame-free event-driven dynamic-vision-sensor. *IEEE Journal of Solid-State Circuits*, 46(6):1443–1455, June 2011.

[110] L. Farian, P. Häfliger, and J. A. Leñero-Bardallo. A miniaturized two-axis ultra low latency and low-power sun sensor for attitude determination of micro space probes. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 65(5):1543–1554, May 2018. ISSN 1549-8328. doi: 10.1109/TCSI.2017.2763990.

[111] Juan A. Leñero-Bardallo, Jose M. Guerrero-Rodríguez, Ricardo Carmona-Galán, and Ángel Rodríguez-Vázquez. On the analysis and detection of flames with an asynchronous spiking image sensor. *IEEE Sensors Journal*, 18(16):6588–6595, 2018. doi: 10.1109/JSEN.2018.2851063.

[112] Juan A. Leñero-Bardallo, D.H. Bryn, and P. Häfliger. Flame monitoring with an AER color vision sensor. In *Circuits and Systems (ISCAS), 2013 IEEE International Symposium On*, pages 2404–2407, May 2013. doi: 10.1109/ISCAS.2013.6572363.

Anexos

Anexo A

Rutina principal

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "spi.h"
#include "tim.h"
#include "usart.h"
#include "gpio.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "eyes.h"
#include "gimbalControl.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

```

```
/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */

  LL_APB2_GRP1_EnableClock(LL_APB2_GRP1_PERIPH_SYSCFG);
  LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_PWR);

  NVIC_SetPriorityGrouping(NVIC_PRIORITYGROUP_4);

  /* System interrupt init*/

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_SPI2_Init();
  MX_TIM1_Init();
  MX_SPI3_Init();
  MX_TIM3_Init();
  MX_TIM4_Init();
  /* USER CODE BEGIN 2 */

  startupPrint();

  gimbalControlInit();

```

```
111   eyes_init();
112   eyes_start();
113
114   /* USER CODE END 2 */
115
116   /* Infinite loop */
117   /* USER CODE BEGIN WHILE */
118   while (1)
119   {
120     /* USER CODE END WHILE */
121
122     /* USER CODE BEGIN 3 */
123   }
124   /* USER CODE END 3 */
125 }
126
127 /**
128   * @brief System Clock Configuration
129   * @retval None
130   */
131 void SystemClock_Config(void)
132 {
133   LL_FLASH_SetLatency(LL_FLASH_LATENCY_3);
134   while(LL_FLASH_GetLatency()!= LL_FLASH_LATENCY_3)
135   {
136   }
137   LL_PWR_SetRegulVoltageScaling(LL_PWR_REGU_VOLTAGE_SCALE1);
138   LL_RCC_HSI_Enable();
139
140    /* Wait till HSI is ready */
141   while(LL_RCC_HSI_IsReady() != 1)
142   {
143
144   }
145   LL_RCC_HSI_SetCalibTrimming(16);
146   LL_RCC_PLL_ConfigDomain_SYS(LL_RCC_PLLSOURCE_HSI, LL_RCC_PLLM_DIV_1, 8, LL_RCC_PLLR_DIV_2);
147   LL_RCC_PLL_EnableDomain_SYS();
148   LL_RCC_PLL_Enable();
149
150    /* Wait till PLL is ready */
151   while(LL_RCC_PLL_IsReady() != 1)
152   {
153
154   }
155   LL_RCC_SetSysClkSource(LL_RCC_SYS_CLKSOURCE_PLL);
156
157    /* Wait till System clock is ready */
158   while(LL_RCC_GetSysClkSource() != LL_RCC_SYS_CLKSOURCE_STATUS_PLL)
159   {
160
161   }
162   LL_RCC_SetAHBPrescaler(LL_RCC_SYSCLK_DIV_1);
163   LL_RCC_SetAPB1Prescaler(LL_RCC_APB1_DIV_1);
164   LL_RCC_SetAPB2Prescaler(LL_RCC_APB2_DIV_1);
165
166   LL_Init1msTick(64000000);
167
168   LL_SetSystemCoreClock(64000000);
169   LL_RCC_SetUSARTClockSource(LL_RCC_USART2_CLKSOURCE_PCLK1);
170 }
171
172 /* USER CODE BEGIN 4 */
173
174 /* USER CODE END 4 */
175
```

```
176 /**
177   * @brief  This function is executed in case of error occurrence.
178   * @retval None
179   */
180 void Error_Handler(void)
181 {
182   /* USER CODE BEGIN Error_Handler_Debug */
183   /* User can add his own implementation to report the HAL error return state */
184 
185   /* USER CODE END Error_Handler_Debug */
186 }
187 
188 #ifdef  USE_FULL_ASSERT
189 /**
190   * @brief  Reports the name of the source file and the source line number
191   *         where the assert_param error has occurred.
192   * @param  file: pointer to the source file name
193   * @param  line: assert_param error line source number
194   * @retval None
195   */
196 void assert_failed(uint8_t *file, uint32_t line)
197 {
198   /* USER CODE BEGIN 6 */
199   /* User can add his own implementation to report the file name and line number,
200      tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
201   /* USER CODE END 6 */
202 }
203 #endif /* USE_FULL_ASSERT */
204 
205 /************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

Anexo B

Librería para la comunicación con el sensor ADNS2610

B.1. Header

```
/*
 * adns-2610.h
 *
 *  Created on: Sep 14, 2020
 *      Author: deros
 */

#ifndef INC_ADNS2610_H_
#define INC_ADNS2610_H_

#include "main.h"

/* Exported constants --------------------------------------------------------*/
/** @defgroup ADNS2610 constants
 *  @{
 */

/** @defgroup ADNS2610 transference times defined in base of 250ns counter
 * @{
 */
/* Times defined in data sheet with some additional tolerance -------------*/
//#define ADNS2610_TIM_BTW_RD     500/250         // 500ns
//#define ADNS2610_TIM_BTW_WR     150000/250        // 150us
//#define ADNS2610_TIM_TO_RD    150000/250        // 150us

/* Times adjusted by hand to reduce the errors in the frame transference -*/
/* Time to acquire a frame (50 + 600) * (324 pixels) + 700 = 211.3ms -> 5 GPS roughly*/
#define ADNS2610_TIM_BTW_RD     50000/250       // 50us
#define ADNS2610_TIM_BTW_WR     700000/250        // 300us
#define ADNS2610_TIM_TO_RD      600000/250        // 300us
/**
 * @}
 */

/** @defgroup ADNS2610 internal registers and constants
 * @{
 */
// Pixel quantity in an ADNS2610 frame
#define PIXEL_QTY   324
```

```
40
41 /** @defgroup ADNS2610 internal registers addresses
42  * @{
43  */
44 #define ADNS2610_CONFIG_REG       0x00
45 #define ADNS2610_STATUS_REG       0x01
46 #define ADNS2610_SQUAL_REG        0x04
47 #define ADNS2610_PIXEL_SUM_REG      0x07
48 #define ADNS2610_PIXEL_DATA_REG     0x08
49 #define ADNS2610_INVERSE_ID_REG     0x11
50 /**
51  * @}
52  */
53 /** @defgroup ADNS2610 CONFIGURATION register bit definitions
54  *
55  */
56 #define ADNS2610_CONFIG_C0_Pos    (0U)
57 #define ADNS2610_CONFIG_C0_Msk    (0x1U << ADNS2610_CONFIG_C0_Pos)
58 #define ADNS2610_CONFIG_C0      ADNS2610_CONFIG_C0_Msk
59 #define ADNS2610_CONFIG_C6_Pos    (6U)
60 #define ADNS2610_CONFIG_C6_Msk    (0x1U << ADNS2610_CONFIG_C0_Pos)
61 #define ADNS2610_CONFIG_C6      ADNS2610_CONFIG_C0_Msk
62 #define ADNS2610_CONFIG_C7_Pos    (7U)
63 #define ADNS2610_CONFIG_C7_Msk    (0x1U << ADNS2610_CONFIG_C0_Pos)
64 #define ADNS2610_CONFIG_C7      ADNS2610_CONFIG_C0_Msk
65 /**
66  * @}
67  */
68 /** @defgroup ADNS2610 STATUS register bit definitions
69  * @{
70  */
71 #define ADNS2610_STATUS_AWAKE_Pos (0U)
72 #define ADNS2610_STATUS_AWAKE_Msk (0x1U << ADNS2610_STATUS_AWAKE_Pos)
73 #define ADNS2610_STATUS_AWAKE   ADNS2610_STATUS_AWAKE_Msk
74 #define ADNS2610_STATUS_ID_Pos    (5U)
75 #define ADNS2610_STATUS_ID_Msk    (0x7U << STATUS_ID_Pos)
76 #define ADNS2610_STATUS_ID      STATUS_ID_Msk
77 /**
78  * @}
79  */
80 /** @defgroup ADNS2610 PIXEL DATA register bit definitions
81  * @{
82  */
83 #define ADNS2610_PIXEL_DATA_Pos   (0U)
84 #define ADNS2610_PIXEL_DATA_Msk   (0x3FU << ADNS2610_PIXEL_DATA_Pos)
85 #define ADNS2610_PIXEL_DATA     ADNS2610_PIXEL_DATA_Msk
86 #define ADNS2610_PIXEL_VALID_Pos  (6U)
87 #define ADNS2610_PIXEL_VALID_Msk  (0x1U << ADNS2610_PIXEL_VALID_Pos)
88 #define ADNS2610_PIXEL_VALID    ADNS2610_PIXEL_VALID_Msk
89 #define ADNS2610_PIXEL_SOF_Pos    (7U)
90 #define ADNS2610_PIXEL_SOF_Msk    (0x1U << ADNS2610_PIXEL_SOF_Pos)
91 #define ADNS2610_PIXEL_SOF      ADNS2610_PIXEL_SOF_Msk
92 /**
93  * @}
94  */
95 /** @defgroup ADNS2610 INVERSE PRODUCT register bit definitions
96  * @{
97  */
98 #define ADNS2610_INV_PROD_Pos   (0U)
99 #define ADNS2610_INV_PROD_Msk   (0xFU << ADNS2610_INV_PROD_Pos)
100 #define ADNS2610_INV_PROD     ADNS2610_INV_PROD_Msk
101 /**
102  * @}
103  */
104
```

```
105 /**
106  * @}
107  */
108
109 /**
110  * @}
111  */
112
113 /* Exported types ------------------------------------------------------------*/
114 typedef enum {
115   ADNS2610_RIGHT = 0,//!< ADNS2610_RIGHT
116   ADNS2610_LEFT = 1  //!< ADNS2610_LEFT
117 } Device;
118
119 typedef enum{
120   VALID_SOF,        //!< VALID_SOF: VALID bit and SOF bit are set
121   NON_VALID_SOF,    //!< NON_VALID_SOF: VALID bit is cleared and SOF is set
122   VALID,        //!< VALID: VALID bit is set
123   NON_VALID     //!< NON_VALID: VALID bit is cleared
124 } PixelStatus;
125
126 typedef uint8_t pixelTypeDef;
127
128 /* Exported variables ---------------------------------------------------------*/
129
130 /* Exported functions ---------------------------------------------------------*/
131 void adns2610_init(Device dev);
132 uint8_t adns2610_readRegister(Device dev, uint8_t reg);
133 void adns2610_writeRegister(Device dev, uint8_t reg, uint8_t value);
134 void adns2610_readFrame(Device dev, pixelTypeDef buffer[]);
135 void adns2610_receiveByte(Device dev, uint8_t* value);
136 void adns2610_sendByte(Device dev, uint8_t value);
137 void adns2610_printImage(pixelTypeDef frame[]);
138 PixelStatus adns2610_checkPixel(pixelTypeDef* Pixel);
139
140 #endif /* INC_ADNS2610_H_ */
```

B.2. Source

```
1  /*
2   * adns-2610.c
3   *
4   *  Created on: Sep 14, 2020
5   *      Author: deros
6   */
7
8  #include <adns2610.h>
9
10 // Private defines
11 #define FULL_DUPLEX_SPI true
12
13 // Global variables declared in adns-2610.h
14 extern pixelTypeDef pixBuf_adns_right [2][PIXEL_QTY];
15
16 // Private macros
17 #define GET_SPI_PERIPH(Device, PeriphPtr)  (PeriphPtr = Device == 0 ? SPI2 : SPI3)
18 #define GET_DEV_NAME(dev, str)  (str = dev == ADNS2610_RIGHT ? "ADNS2610_RIGHT" : "ADNS2610_LEFT")
19
20 // Private static variables
21 static SPI_TypeDef * SPIx;
22
23 // Private function prototypes
24 void adns2610_resetCOM(Device dev);
25 void adns2610_config(Device dev);
```

```
26 void adns2610_configureSPI(Device dev);
27
28 /**
29  * @brief Initialize the ADNS2610 sensor
30  * @param dev Device address, it refers to SPI peripheral where the sensor is connected
31  */
32 void adns2610_init(Device dev){
33   // Configure the SPI peripherals for each sensor
34   adns2610_configureSPI(dev);
35
36   // Reset communication with ADNS sensors
37   adns2610_resetCOM(dev);
38
39   // Configure sensors
40   adns2610_config(dev);
41 }
42 /**
43  * @brief Configure the SPI module pointed by Device argument
44  * @param dev Device address, it refers to SPI peripheral where the sensor is connected
45  */
46 void adns2610_configureSPI(Device dev){
47   GET_SPI_PERIPH(dev, SPIx);
48   // RX FIFO threshold adjusted to 8-bit word
49   SET_BIT(SPIx->CR2, SPI_CR2_FRXTH);
50   // Enable SPI
51   SET_BIT(SPIx->CR1, SPI_CR1_SPE);
52 }
53 /**
54  * @brief Reset the ADNS2610 serial port. It needs to be done at the beginning to establish the
       communication
55  *      correctly
56  * @param dev Device address, it refers to SPI peripheral where the sensor is connected
57  */
58 void adns2610_resetCOM(Device dev){
59
60   GET_SPI_PERIPH(dev, SPIx);
61
62   // Check TXE to send data
63   while(!(READ_BIT(SPIx->SR, SPI_SR_TXE)));
64   // Write DR to send data through SPI
65   WRITE_REG(*(__IO uint8_t*) &SPIx->DR, 0x01);
66   // Wait until RXNE is set
67   while(!(READ_BIT(SPIx->SR, SPI_SR_RXNE)));
68   READ_REG(*(__IO uint8_t*) &SPIx->DR);
69   // Wait until end the current transaction
70   while((READ_BIT(SPIx->SR, SPI_SR_FTLVL)) | (READ_BIT(SPIx->SR, SPI_SR_FRLVL)) | (READ_BIT(SPIx->SR
       , SPI_SR_BSY)));
71   LL_mDelay(100);
72 }
73 /**
74  * @brief Configure the ADNS2610 internal register. Set always awake and check the inverse product
       ID register
75  * @param dev Device address, it refers to SPI peripheral where the sensor is connected
76  */
77 void adns2610_config(Device dev){
78   // ADNS-2610 configuration
79   char * devName;
80
81   GET_DEV_NAME(dev, devName);
82
83   printf("--------------------------------\r\n--> %s CONFIGURATION \r\n
       --------------------------------\r\n", devName);
84
85   printf("Setting the sensor to always awake in %s...\r\n", _(ADNS2610_CONFIG));
86   adns2610_writeRegister(dev, ADNS2610_CONFIG_REG, ADNS2610_CONFIG_C0);
```

```
87
88    printf("Checking if %s has been written well... ", _(ADNS2610_CONFIG));
89    if(adns2610_readRegister(dev, ADNS2610_CONFIG_REG) == ADNS2610_CONFIG_C0) printf("OK.\r\n");
90    else{ printf("ERROR.\r\n"); while(1);}
91
92    printf("Checking into %s if the sensor is awake... ", _(ADNS2610_STATUS));
93    if(adns2610_readRegister(dev, ADNS2610_STATUS_REG) == ADNS2610_STATUS_AWAKE) printf("OK.\r\n");
94    else{ printf("ERROR.\r\n"); while(1);}
95
96    printf("Checking into %s if the sensor responds well... ", _(ADNS2610_INVERSE_ID));
97    if((adns2610_readRegister(dev, ADNS2610_INVERSE_ID_REG) & ADNS2610_INV_PROD)  == ADNS2610_INV_PROD
         ) printf("OK.\r\n");
98    else{ printf("ERROR.\r\n"); while(1);}
99
100   printf("\r\n");
101 }
102 /**
103  * @brief Read a ADNS2610 internal register by polling
104  * @param dev Device address, it refers to SPI peripheral where the sensor is connected
105  * @param reg Internal register ADDRESS, see adns2610.h
106  * @return Register value
107  */
108 uint8_t adns2610_readRegister(Device dev, uint8_t reg){
109
110   uint8_t value;
111
112   GET_SPI_PERIPH(dev, SPIx);
113
114   #if FULL_DUPLEX_SPI
115   // Check TXE to send data
116   while(!(READ_BIT(SPIx->SR, SPI_SR_TXE)));
117   // Write DR to send data through SPI
118   WRITE_REG(*(__IO uint8_t*) &SPIx->DR, reg);
119   // Wait until RXNE is set
120   while(!(READ_BIT(SPIx->SR, SPI_SR_RXNE)));
121   READ_REG(*(__IO uint8_t*) &SPIx->DR);
122   LL_mDelay(1);
123   // Write DR to send data through SPI
124   WRITE_REG(*(__IO uint8_t*) &SPIx->DR, 0x00);
125   // Wait until RXNE is set
126   while(!(READ_BIT(SPIx->SR, SPI_SR_RXNE)));
127   value = READ_REG(*(__IO uint8_t*) &SPIx->DR);
128   // Wait until end the current transaction
129   while((READ_BIT(SPIx->SR, SPI_SR_FTLVL)) | (READ_BIT(SPIx->SR, SPI_SR_FRLVL)) | (READ_BIT(SPIx->SR
         , SPI_SR_BSY)));
130   return value;
131   #else // HALF DUPLEX SPI MODE
132
133   #endif
134 }
135 /**
136  * @brief Write a ADNS2610 internal register by polling
137  * @param dev Device address, it refers to SPI peripheral where the sensor is connected
138  * @param reg Internal register ADDRESS, see adns2610.h
139  * @param value Value to write in the internal register
140  */
141 void adns2610_writeRegister(Device dev, uint8_t reg, uint8_t value){
142
143   GET_SPI_PERIPH(dev, SPIx);
144
145   #if FULL_DUPLEX_SPI
146   // RX FIFO threshold adjusted to 16-bit word
147   CLEAR_BIT(SPIx->CR2, SPI_CR2_FRXTH);
148   // Check TXE to send data
149   while(!(READ_BIT(SPIx->SR, SPI_SR_TXE)));
```

```
150   // Write DR to send data through SPI
151   WRITE_REG(SPIx->DR, (value << 8) | (1U << 7 | reg));
152   // Wait until RXNE is set
153   while(!(READ_BIT(SPIx->SR, SPI_SR_RXNE)));
154   READ_REG(SPIx->DR);
155   // Wait until end the current transaction
156   while((READ_BIT(SPIx->SR, SPI_SR_FTLVL)) | (READ_BIT(SPIx->SR, SPI_SR_FRLVL)) | (READ_BIT(SPIx->SR
        , SPI_SR_BSY)));
157   // Set again RX FIFO threshold adjusted to 8-bit word
158   SET_BIT(SPIx->CR2, SPI_CR2_FRXTH);
159   #else // HALF DUPLEX SPI MODE
160
161   #endif
162 }
163 /**
164  * @brief Receive a byte from ADNS2610 as reply of adns2610_sendByte(Device dev, uint8_t value)
        function
165  * @param dev Device address, it refers to SPI peripheral where the sensor is connected
166  * @param value Pointer to a variable where the received value is stored
167  */
168 void adns2610_receiveByte(Device dev, uint8_t* value){
169
170   GET_SPI_PERIPH(dev, SPIx);
171
172   #if FULL_DUPLEX_SPI
173   // Write DR to send data through SPI
174   WRITE_REG(*(__IO uint8_t*) &SPIx->DR, 0x00);
175   // Wait until RXNE is set
176   while(!(READ_BIT(SPIx->SR, SPI_SR_RXNE)));
177   *value = READ_REG(*(__IO uint8_t*) &SPIx->DR);
178   // Wait until end the current transaction
179   while((READ_BIT(SPIx->SR, SPI_SR_FTLVL)) | (READ_BIT(SPIx->SR, SPI_SR_FRLVL)) | (READ_BIT(SPIx->SR
        , SPI_SR_BSY)));
180   #else // HALF DUPLEX SPI MODE
181
182   #endif
183 }
184 /**
185  * @brief Send a byte to ADNS2610. It's used to request to ADNS2610 a register value in IT mode
186  * @param dev Device address, it refers to SPI peripheral where the sensor is connected
187  * @param value Value of the sent value
188  */
189 void adns2610_sendByte(Device dev, uint8_t value){
190
191   GET_SPI_PERIPH(dev, SPIx);
192
193   #if FULL_DUPLEX_SPI
194   // Check TXE to send data
195   while(!(READ_BIT(SPIx->SR, SPI_SR_TXE)));
196   // Write DR to send data through SPI
197   WRITE_REG(*(__IO uint8_t*) &SPIx->DR, value);
198   // Wait until RXNE is set
199   while(!(READ_BIT(SPIx->SR, SPI_SR_RXNE)));
200   READ_REG(*(__IO uint8_t*) &SPIx->DR);
201   #else // HALF DUPLEX SPI MODE
202
203   #endif
204 }
205 /**
206  * @brief Read a frame from ADNS2610 by polling
207  * @param dev Device address, it refers to SPI peripheral where the sensor is connected
208  * @param buffer Array where the frame is going to be stored
209  */
210 void adns2610_readFrame(Device dev, pixelTypeDef buffer[]){
211   uint16_t idx = 0;
```

```
212
213   while(idx < PIXEL_QTY){
214     if(!idx){
215       adns2610_writeRegister(dev, ADNS2610_PIXEL_DATA_REG, 0x01);
216       LL_mDelay(1);
217       buffer[idx] = adns2610_readRegister(dev, ADNS2610_PIXEL_DATA_REG);
218
219       if(buffer[idx] & (ADNS2610_PIXEL_VALID | ADNS2610_PIXEL_SOF)){
220         idx++;
221         continue;
222       }
223     }
224
225     buffer[idx] = adns2610_readRegister(dev, ADNS2610_PIXEL_DATA_REG);
226
227     if(buffer[idx] & ADNS2610_PIXEL_SOF){
228       idx = 0;
229       LL_mDelay(1);
230       continue;
231     }
232
233     if(buffer[idx] & ADNS2610_PIXEL_VALID){
234       idx++;
235     }
236   }
237 }
238 /**
239  * @brief Check the status of a pixel
240  * @param Pixel The PIXEL DATA register value received from ADNS2610
241  * @return See PixelStatus
242  */
243 PixelStatus adns2610_checkPixel(pixelTypeDef* Pixel){
244   if(*Pixel & ADNS2610_PIXEL_VALID){
245     if(*Pixel & ADNS2610_PIXEL_SOF){
246       return VALID_SOF;
247     }
248     return VALID;
249   }
250   else if(*Pixel & ADNS2610_PIXEL_SOF){
251     return NON_VALID_SOF;
252   }
253   else{
254     return NON_VALID;
255   }
256 }
257 /**
258  * @brief Print the received frame values in the console through UART
259  * @param frame The array which contains the pixel values
260  */
261 void adns2610_printImage(pixelTypeDef frame[]){
262   uint16_t i = 0;
263
264   printf("=========================================================\r\n||");
265
266   while(i < PIXEL_QTY){
267     if(!(i % 18) & (i > 1)){
268       printf("||\r\n||");
269     }
270     printf("%02d ", frame[i] & ADNS2610_PIXEL_DATA);
271     i++;
272   }
273
274   printf("||\r\n=========================================================\r\n");
275 }
```

Anexo C

Librería para el cálculo de flujo óptico

C.1. Header

```
/*
 * opticalFlow.h
 *
 *  Created on: 23 sept. 2020
 *      Author: deros
 */

#ifndef INC_OPTICALFLOW_H_
#define INC_OPTICALFLOW_H_

#include <stdio.h>
#include "framesIndexers.h"
#include "adns2610.h"

/* Private constant ----------------------------------------------------------*/

/* Exported typedefs ---------------------------------------------------------*/
typedef struct __attribute__((__packed__)){
  int32_t x;
  int32_t y;
} optical2DFlowStruct;

typedef struct __attribute__((__packed__)){
  int32_t x;
  int32_t y;
  int32_t theta;
} optical2DandRotateFlowStruct;

/* Exported functions ---------------------------------------------------------*/
void OF_ResetCoefficients();
void OF_ComputeCoefficients(Device dev, uint8_t currentFrame[], uint8_t lastFrame[], int32_t idx);
void OF_Compute(Device dev, int32_t* ofX, int32_t* ofY);
void OF_ComputeFused(optical2DFlowStruct* right, optical2DFlowStruct* left,
     optical2DandRotateFlowStruct* fused);

#endif /* INC_OPTICALFLOW_H_ */
```

C.2. Source

```c
/*
 * opticalFlow.c
 *
 *  Created on: 23 sept. 2020
 *      Author: deros
 */

#include "opticalFlow.h"

/* Private constants ---------------------------------------------------------*/
#define int24_t_Mask    0x80FFFFFF
#define bitsOfResolution  9

static int64_t A[2];
static int64_t B[2];
static int64_t C[2];
static int64_t D[2];
static int64_t E[2];
static int32_t deltaX;
static int32_t deltaY;
static int32_t deltaT;
static int16_t frameIdx;

/**
 * @brief Reset the coefficient values and the frame's pixel index to zero
 */
void OF_ResetCoefficients(){
  A[0] = B[0] = C[0] = D[0] = E[0] = 0;
  A[1] = B[1] = C[1] = D[1] = E[1] = 0;
  frameIdx = 0;
}

/**
 * @brief It computes the optical flow coefficients related to a certain pixel if the needed data is
     available
 * @param dev The device where the frames from
 * @param currentFrame  The current frame to process the optical flow
 * @param lastFrame   The reference frame to process the optical flow
 * @param idx     The frame's pixel index which is going to be processed
 */
void OF_ComputeCoefficients(Device dev, uint8_t currentFrame[], uint8_t lastFrame[], int32_t idx){

  if(fSelect[idx]){
    deltaX = (lastFrame[f2[frameIdx]] & ADNS2610_PIXEL_DATA) - (lastFrame[f1[frameIdx]] &
     ADNS2610_PIXEL_DATA);
    deltaY = (lastFrame[f4[frameIdx]] & ADNS2610_PIXEL_DATA) - (lastFrame[f3[frameIdx]] &
     ADNS2610_PIXEL_DATA);
    deltaT = (currentFrame[f0[frameIdx]] & ADNS2610_PIXEL_DATA) - (lastFrame[f0[frameIdx]] &
     ADNS2610_PIXEL_DATA);

    A[dev] += deltaX * deltaX;
    B[dev] += deltaY * deltaX;
    C[dev] += deltaT * deltaX;
    D[dev] += deltaY * deltaY;
    E[dev] += deltaT * deltaY;

    frameIdx++;
  }
}

/**
 * @brief It computes the optical flow value from the coefficients computed in the last iterations
 * @param dev The device where the frames from
 * @param ofX Pointer to the variable where the optical flow value in X direction is going to be
     stored
```

```
61  * @param ofY Pointer to the variable where the optical flow value in Y direction is going to be
        stored
62  */
63 void OF_Compute(Device dev, int32_t* ofX, int32_t* ofY){
64   int64_t num, den;
65
66   den = A[dev] * D[dev] - B[dev] * B[dev];
67
68   if(den > 0){
69     num = (C[dev]*D[dev]) - (B[dev]*E[dev]);
70     *ofX = (num << bitsOfResolution)  / den;
71     num = (A[dev]*E[dev]) - (B[dev]*C[dev]);
72     *ofY = (num << bitsOfResolution) / den;
73   }
74   else{
75     *ofX = *ofY = 0;
76   }
77 }
78
79 /**
80  * @brief It computes the optical flow fusion from the optical flow values computed for the two
        connected devices
81  * @param right Pointer to the struct which contains the optical flow values computed from the right
         positioned device
82  * @param left  Pointer to the struct which contains the optical flow values computed from the left
        positioned device
83  * @param fused Pointer to the struct where the fused values are going to be stored
84  */
85 void OF_ComputeFused(optical2DFlowStruct* right, optical2DFlowStruct* left,
        optical2DandRotateFlowStruct* fused){
86   fused->x = (right->x + left->x) >> 1;
87   fused->y = (right->y + left->y) >> 1;
88   if((right->y < 0 && left->y > 0) || (right->y > 0 && left->y < 0))
89     fused->theta = (right->y - left->y);
90 }
```

C.3. Header para almacenar la tabla de indices

```
1  /*
2   * framesIndexers.h
3   *
4   *  Created on: 23 sept. 2020
5   *      Author: deros
6   */
7
8  #ifndef INC_FRAMESINDEXERS_H_
9  #define INC_FRAMESINDEXERS_H_
10
11 #include "stdbool.h"
12
13 static const uint16_t f0[] = {   19,   20,   21,   22,   23,   24,   25,   26,   27,   28,   29,
       30,   31,   32,   33,   34,  // 1
14                  37,   38,   39,   40,   41,   42,   43,   44,   45,   46,   47,   48,   49,   50,
        51,   52,  // 2
15                  55,   56,   57,   58,   59,   60,   61,   62,   63,   64,   65,   66,   67,   68,
        69,   70,  // 3
16                  73,   74,   75,   76,   77,   78,   79,   80,   81,   82,   83,   84,   85,   86,
        87,   88,  // 4
17                  91,   92,   93,   94,   95,   96,   97,   98,   99,  100,  101,  102,  103,  104,
       105,  106,  // 5
18                 109,  110,  111,  112,  113,  114,  115,  116,  117,  118,  119,  120,  121,  122,
       123,  124,  // 6
19                 127,  128,  129,  130,  131,  132,  133,  134,  135,  136,  137,  138,  139,  140,
       141,  142,  // 7
```

```
20                145,  146,  147,  148,  149,  150,  151,  152,  153,  154,  155,  156,  157,  158,
      159,  160,  // 8
21                163,  164,  165,  166,  167,  168,  169,  170,  171,  172,  173,  174,  175,  176,
      177,  178,  // 9
22                181,  182,  183,  184,  185,  186,  187,  188,  189,  190,  191,  192,  193,  194,
      195,  196,  // 10
23                199,  200,  201,  202,  203,  204,  205,  206,  207,  208,  209,  210,  211,  212,
      213,  214,  // 11
24                217,  218,  219,  220,  221,  222,  223,  224,  225,  226,  227,  228,  229,  230,
      231,  232,  // 12
25                235,  236,  237,  238,  239,  240,  241,  242,  243,  244,  245,  246,  247,  248,
      249,  250,  // 13
26                253,  254,  255,  256,  257,  258,  259,  260,  261,  262,  263,  264,  265,  266,
      267,  268,  // 14
27                271,  272,  273,  274,  275,  276,  277,  278,  279,  280,  281,  282,  283,  284,
      285,  286,  // 15
28                289,  290,  291,  292,  293,  294,  295,  296,  297,  298,  299,  300,  301,  302,
      303,  304   // 16
29             };
30
31 static const uint16_t f1 [] = {  37,   38,   39,   40,   41,   42,   43,   44,   45,   46,   47,
      48,   49,   50,   51,   52,  // 1
32                 55,   56,   57,   58,   59,   60,   61,   62,   63,   64,   65,   66,   67,   68,
       69,   70,  // 2
33                 73,   74,   75,   76,   77,   78,   79,   80,   81,   82,   83,   84,   85,   86,
       87,   88,  // 3
34                 91,   92,   93,   94,   95,   96,   97,   98,   99,  100,  101,  102,  103,  104,
      105,  106,  // 4
35                109,  110,  111,  112,  113,  114,  115,  116,  117,  118,  119,  120,  121,  122,
      123,  124,  // 5
36                127,  128,  129,  130,  131,  132,  133,  134,  135,  136,  137,  138,  139,  140,
      141,  142,  // 6
37                145,  146,  147,  148,  149,  150,  151,  152,  153,  154,  155,  156,  157,  158,
      159,  160,  // 7
38                163,  164,  165,  166,  167,  168,  169,  170,  171,  172,  173,  174,  175,  176,
      177,  178,  // 8
39                181,  182,  183,  184,  185,  186,  187,  188,  189,  190,  191,  192,  193,  194,
      195,  196,  // 9
40                199,  200,  201,  202,  203,  204,  205,  206,  207,  208,  209,  210,  211,  212,
      213,  214,  // 10
41                217,  218,  219,  220,  221,  222,  223,  224,  225,  226,  227,  228,  229,  230,
      231,  232,  // 11
42                235,  236,  237,  238,  239,  240,  241,  242,  243,  244,  245,  246,  247,  248,
      249,  250,  // 12
43                253,  254,  255,  256,  257,  258,  259,  260,  261,  262,  263,  264,  265,  266,
      267,  268,  // 13
44                271,  272,  273,  274,  275,  276,  277,  278,  279,  280,  281,  282,  283,  284,
      285,  286,  // 14
45                289,  290,  291,  292,  293,  294,  295,  296,  297,  298,  299,  300,  301,  302,
      303,  304,  // 15
46                307,  308,  309,  310,  311,  312,  313,  314,  315,  316,  317,  318,  319,  320,
      321,  322   // 16
47             };
48
49 static const uint16_t f2 [] = {   1,    2,    3,    4,    5,    6,    7,    8,    9,   10,   11,
      12,   13,   14,   15,   16,  // 1
50                 19,   20,   21,   22,   23,   24,   25,   26,   27,   28,   29,   30,   31,   32,
       33,   34,  // 2
51                 37,   38,   39,   40,   41,   42,   43,   44,   45,   46,   47,   48,   49,   50,
       51,   52,  // 3
52                 55,   56,   57,   58,   59,   60,   61,   62,   63,   64,   65,   66,   67,   68,
       69,   70,  // 4
53                 73,   74,   75,   76,   77,   78,   79,   80,   81,   82,   83,   84,   85,   86,
       87,   88,  // 5
54                 91,   92,   93,   94,   95,   96,   97,   98,   99,  100,  101,  102,  103,  104,
```

```
    105,  106,  // 6
55              109,  110,  111,  112,  113,  114,  115,  116,  117,  118,  119,  120,  121,  122,
    123,  124,  // 7
56              127,  128,  129,  130,  131,  132,  133,  134,  135,  136,  137,  138,  139,  140,
    141,  142,  // 8
57              145,  146,  147,  148,  149,  150,  151,  152,  153,  154,  155,  156,  157,  158,
    159,  160,  // 9
58              163,  164,  165,  166,  167,  168,  169,  170,  171,  172,  173,  174,  175,  176,
    177,  178,  // 10
59              181,  182,  183,  184,  185,  186,  187,  188,  189,  190,  191,  192,  193,  194,
    195,  196,  // 11
60              199,  200,  201,  202,  203,  204,  205,  206,  207,  208,  209,  210,  211,  212,
    213,  214,  // 12
61              217,  218,  219,  220,  221,  222,  223,  224,  225,  226,  227,  228,  229,  230,
    231,  232,  // 13
62              235,  236,  237,  238,  239,  240,  241,  242,  243,  244,  245,  246,  247,  248,
    249,  250,  // 14
63              253,  254,  255,  256,  257,  258,  259,  260,  261,  262,  263,  264,  265,  266,
    267,  268,  // 15
64              271,  272,  273,  274,  275,  276,  277,  278,  279,  280,  281,  282,  283,  284,
    285,  286   // 16
65          };
66
67  static const uint16_t f3 [] = {  20,   21,   22,   23,   24,   25,   26,   27,   28,   29,   30,
    31,   32,   33,   34,   35,  // 1
68               38,   39,   40,   41,   42,   43,   44,   45,   46,   47,   48,   49,   50,   51,
     52,   53,  // 2
69               56,   57,   58,   59,   60,   61,   62,   63,   64,   65,   66,   67,   68,   69,
     70,   71,  // 3
70               74,   75,   76,   77,   78,   79,   80,   81,   82,   83,   84,   85,   86,   87,
     88,   89,  // 4
71               92,   93,   94,   95,   96,   97,   98,   99,  100,  101,  102,  103,  104,  105,
    106,  107,  // 5
72              110,  111,  112,  113,  114,  115,  116,  117,  118,  119,  120,  121,  122,  123,
    124,  125,  // 6
73              128,  129,  130,  131,  132,  133,  134,  135,  136,  137,  138,  139,  140,  141,
    142,  143,  // 7
74              146,  147,  148,  149,  150,  151,  152,  153,  154,  155,  156,  157,  158,  159,
    160,  161,  // 8
75              164,  165,  166,  167,  168,  169,  170,  171,  172,  173,  174,  175,  176,  177,
    178,  179,  // 9
76              182,  183,  184,  185,  186,  187,  188,  189,  190,  191,  192,  193,  194,  195,
    196,  197,  // 10
77              200,  201,  202,  203,  204,  205,  206,  207,  208,  209,  210,  211,  212,  213,
    214,  215,  // 11
78              218,  219,  220,  221,  222,  223,  224,  225,  226,  227,  228,  229,  230,  231,
    232,  233,  // 12
79              236,  237,  238,  239,  240,  241,  242,  243,  244,  245,  246,  247,  248,  249,
    250,  251,  // 13
80              254,  255,  256,  257,  258,  259,  260,  261,  262,  263,  264,  265,  266,  267,
    268,  269,  // 14
81              272,  273,  274,  275,  276,  277,  278,  279,  280,  281,  282,  283,  284,  285,
    286,  287,  // 15
82              290,  291,  292,  293,  294,  295,  296,  297,  298,  299,  300,  301,  302,  303,
    304,  305   // 16
83          };
84
85  static const uint16_t f4 [] = {   18,   19,   20,   21,   22,   23,   24,   25,   26,   27,   28,
     29,   30,   31,   32,   33,  // 1
86                36,   37,   38,   39,   40,   41,   42,   43,   44,   45,   46,   47,   48,   49,
       50,   51,  // 2
87                54,   55,   56,   57,   58,   59,   60,   61,   62,   63,   64,   65,   66,   67,
       68,   69,   // 3
88                72,   73,   74,   75,   76,   77,   78,   79,   80,   81,   82,   83,   84,   85,
       86,   87,   // 4
```

```
89                  90,   91,   92,   93,   94,   95,   96,   97,   98,   99,  100,  101,  102,  103,
        104,  105,    // 5
90                 108,  109,  110,  111,  112,  113,  114,  115,  116,  117,  118,  119,  120,  121,
        122,  123,  // 6
91                 126,  127,  128,  129,  130,  131,  132,  133,  134,  135,  136,  137,  138,  139,
        140,  141,  // 7
92                 144,  145,  146,  147,  148,  149,  150,  151,  152,  153,  154,  155,  156,  157,
        158,  159,  // 8
93                 162,  163,  164,  165,  166,  167,  168,  169,  170,  171,  172,  173,  174,  175,
        176,  177,  // 9
94                 180,  181,  182,  183,  184,  185,  186,  187,  188,  189,  190,  191,  192,  193,
        194,  195,  // 10
95                 198,  199,  200,  201,  202,  203,  204,  205,  206,  207,  208,  209,  210,  211,
        212,  213,  // 11
96                 216,  217,  218,  219,  220,  221,  222,  223,  224,  225,  226,  227,  228,  229,
        230,  231,  // 12
97                 234,  235,  236,  237,  238,  239,  240,  241,  242,  243,  244,  245,  246,  247,
        248,  249,  // 13
98                 252,  253,  254,  255,  256,  257,  258,  259,  260,  261,  262,  263,  264,  265,
        266,  267,  // 14
99                 270,  271,  272,  273,  274,  275,  276,  277,  278,  279,  280,  281,  282,  283,
        284,  285,  // 15
100                288,  289,  290,  291,  292,  293,  294,  295,  296,  297,  298,  299,  300,  301,
        302,  303   // 16
101             };
102
103
104
105 static const bool fSelect[] = {   false, false,  false, false, false, false, false, false, false,
        false, false, false, false, false, false, false, false, false,  // 1
106                false, false,  false, false, false, false, false, false, false, false, false,
        false, false, false, false, false, false, false,  // 2
107                false,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true
        ,  true,  true,  true,  true,  true,  false,  // 3
108                false,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true
        ,  true,  true,  true,  true,  true,  false,  // 4
109                false,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true
        ,  true,  true,  true,  true,  true,  false,  // 5
110                false,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true
        ,  true,  true,  true,  true,  true,  false,  // 6
111                false,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true
        ,  true,  true,  true,  true,  true,  false,  // 7
112                false,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true
        ,  true,  true,  true,  true,  true,  false,  // 8
113                false,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true
        ,  true,  true,  true,  true,  true,  false,  // 9
114                false,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true
        ,  true,  true,  true,  true,  true,  false,  // 10
115                false,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true
        ,  true,  true,  true,  true,  true,  false,  // 11
116                false,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true
        ,  true,  true,  true,  true,  true,  false,  // 12
117                false,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true
        ,  true,  true,  true,  true,  true,  false,  // 13
118                false,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true
        ,  true,  true,  true,  true,  true,  false,  // 14
119                false,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true
        ,  true,  true,  true,  true,  true,  false,  // 15
120                false,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true
        ,  true,  true,  true,  true,  true,  false,  // 16
121                false,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true
        ,  true,  true,  true,  true,  true,  false,  // 17
122                false,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true,  true
        ,  true,  true,  true,  true,  true,  false   // 18
123             };
```

```
124
125 #endif /* INC_FRAMESINDEXERS_H_ */
```

Anexo D

Librería para el desarrollo de la máquina de estados finitos implementada

D.1. Header

```
/*
 * eyes.h
 *
 *  Created on: 18 sept. 2020
 *      Author: deros
 */

#ifndef INC_EYES_H_
#define INC_EYES_H_

#include "usart.h"
#include "adns2610.h"
#include "opticalFlow.h"
#include "gimbalControl.h"

/** @defgroup Eyes constants
 *  @{
 */

/* Exported constants --------------------------------------------------------*/
#define FRAME_STUCT_LENGTH  681  // bytes
#define FRAME_HEADER    0xBFAABFAA
#define SECOND_SENSOR_IMPLEMENTED true
/**
 * @}
 */

/* Exported types ------------------------------------------------------------*/
typedef struct __attribute__((__packed__)){   /* payload ->     2*324 + 2*2*4 + 3*4 = 676 bytes  */
  const uint32_t header;            /* total length (logic + payload) ->  681 bytes */
  uint8_t seq;
  pixelTypeDef frame [2][PIXEL_QTY];
  optical2DFlowStruct oFRight;
  optical2DFlowStruct oFLeft;
  optical2DandRotateFlowStruct oFFused;
} frameStruct;
```

```

/* Exported variables ---------------------------------------------------------*/
extern frameStruct frames[2];
uint8_t currentFrameIdx;
uint8_t lastFrameIdx;

/* Exported functions ---------------------------------------------------------*/
void eyes_init();
void eyes_start();
void eyes_stop();

#endif /* INC_EYES_H_ */
```

D.2. Source

```
/*
 * eyes.c
 *
 *  Created on: 18 sept. 2020
 *      Author: deros
 */

/* Private includes ----------------------------------------------------------*/
#include "eyes.h"

/* Private macro -------------------------------------------------------------*/
#define SWITCH_FRAME_IDX(current, past) {uint8_t temp; temp = past; past = current; current = temp;}
#define TIME_TO_POSITION  410   // milliseconds

/* Private typedefs -----------------------------------------------*/
typedef enum adns2610_state{
  SENSOR_RESET,
  TRIGGER_FRAME,
  READING_FRAME,
  REQ_READING_FRAME,
  PROCESSING
} eyes_FSMstate_TypeDef;

/* Private variables ----------------------------------------------*/
static eyes_FSMstate_TypeDef FSMstate;
bool initialized = false;

/* Private functions ----------------------------------------------*/
void eyes_configureFSM_TIM(void);
void eyes_configureControl_TIM(void);
void eyes_FSM(void);
__STATIC_INLINE void eyes_waitIT(uint32_t Count250ns);
__STATIC_INLINE void eyes_stopWaitIT();
__STATIC_INLINE void eyes_waitControlTIM_IT(uint32_t millis);
__STATIC_INLINE void eyes_stopWaitControlTIM_IT();
bool eyes_computeIdxFromStatus(PixelStatus* status1, PixelStatus* status2, uint16_t* idx1,  uint16_t
    * idx2);

/* Exported variables ----------------------------------------------*/
frameStruct frames[2] = {{.header = FRAME_HEADER}, {.header = FRAME_HEADER}};

/**
 * @brief It initializes and it sets up the system.
 */
void eyes_init(){
  // Configure the timer to read the frames continuously
  eyes_configureFSM_TIM();
  eyes_configureControl_TIM();
```

```
48
49    // Initialize ADNS2610 sensor
50    adns2610_init(ADNS2610_RIGHT);
51  #if SECOND_SENSOR_IMPLEMENTED
52    adns2610_init(ADNS2610_LEFT);
53  #endif
54
55    // Configure DMA to transfer the frameStruct through DMA
56    configureDMA_USART_TX(USART2, BYTE, MEDIUM);
57
58    // Giving initial values to variables
59    currentFrameIdx = 0;
60    lastFrameIdx = 1;
61
62    // Initialization done
63    initialized = true;
64  }
65
66  /**
67   * @brief It starts the system operation.
68   */
69  void eyes_start(){
70
71    if(!initialized) eyes_init();
72
73    SET_BIT(TIM1->CR1, TIM_CR1_CEN);
74    FSMstate = TRIGGER_FRAME;
75  }
76
77  /**
78   * It stops the system operation.
79   */
80  void eyes_stop(){
81    FSMstate = RESET;
82  }
83
84  /* eye FSM states operation ----------------------------------------------------
85   *  -SENSOR_RESET:    The FSM stops. When it is restarted all is reseted.
86   *
87   *  -TRIGGER_FRAME:   The frame is taken. It's performed by a write operation
88   *            to PIXEL DATA register.
89   *
90   *  -REQ_READING_FRAME: The PIXEL DATA register address is sent. After that it's
91   *            needed to wait 100us to get the pixel data. This time is
92   *            used to check the last pixel received.
93   *
94   *  -READING_FRAME:   The pixel data is received and stored after the wait of
95   *            100us.
96   * ------------------------------------------------------------------------------ */
97  /**
98   *  @brief Compute the FSM (Finite State Machine) for image acquisition, optical flow computation
       and control loop.
99   */
100 void eyes_FSM(void){
101   static uint16_t pixelIdx[2] = { 0 };
102   static PixelStatus pixelStatus [2] = { 0 };
103   static bool firstPixelRead = true;
104   static bool firstFrameRead = true;
105
106   static uint8_t collisionFlag = 0;
107   static uint16_t errorCounter = 0;
108   static uint8_t seqTemp;
109
110   switch(FSMstate){
111   /* SENSOR_RESET state ------------------------------------------------------ */
```

```
112   case SENSOR_RESET:
113 //    if(collisionFlag) goto collisionError; else collisionFlag = 1;
114     pixelIdx[ADNS2610_RIGHT] = 0;
115 #if SECOND_SENSOR_IMPLEMENTED
116     pixelIdx[ADNS2610_LEFT] = 0;
117 #endif
118     /* Stop the interrupt timer and reset all the relevant values */
119     eyes_stopWaitIT();
120     eyes_stopWaitControlTIM_IT();
121 
122     pixelIdx[0] = pixelIdx[1] = 0;
123     pixelStatus[0] = pixelIdx[1] = 0;
124     firstPixelRead = true;
125     firstFrameRead = true;
126     seqTemp = 0;
127     initialized = false;
128     collisionFlag = 0;
129     return;
130   /* TRIGGER_FRAME state ------------------------------------------------------- */
131   case TRIGGER_FRAME:
132     eyes_stopWaitIT();
133     eyes_stopWaitControlTIM_IT();
134     if(collisionFlag) goto collisionError; else collisionFlag = 1;
135     /* Write pixel data register to reset the HW */
136     adns2610_writeRegister(ADNS2610_RIGHT, ADNS2610_PIXEL_DATA_REG, 0x01);
137 #if SECOND_SENSOR_IMPLEMENTED
138     adns2610_writeRegister(ADNS2610_LEFT, ADNS2610_PIXEL_DATA_REG, 0x01);
139 #endif
140     /* While it waits the needed delay it's performed some tasks:
141      *  - Increasing the SEQ number
142      *  - Transfer all data by means of DMA
143      *  */
144     eyes_waitIT(ADNS2610_TIM_BTW_WR);
145     firstPixelRead = true;
146     FSMstate = REQ_READING_FRAME;
147     pixelIdx[ADNS2610_RIGHT] = 0;
148 #if SECOND_SENSOR_IMPLEMENTED
149     pixelIdx[ADNS2610_LEFT] = 0;
150 #endif
151     if(!firstFrameRead){
152       frames[lastFrameIdx].seq = (seqTemp++) & 0x7F;
153       transferDMA_USART2_TX((uint32_t) &(frames[lastFrameIdx].header), FRAME_STUCT_LENGTH);
154       OF_ResetCoefficients();
155     }
156     collisionFlag = 0;
157     errorCounter = 0;
158     return;
159   /* REQ_READING_FRAME state --------------------------------------------------- */
160   case REQ_READING_FRAME:
161     eyes_stopWaitIT();
162     if(collisionFlag) goto collisionError; else collisionFlag = 1;
163     /* Send a pixel data read request if there are pixels to read*/
164     if(pixelIdx[ADNS2610_RIGHT] <= PIXEL_QTY-1) adns2610_sendByte(ADNS2610_RIGHT,
       ADNS2610_PIXEL_DATA_REG);
165 #if SECOND_SENSOR_IMPLEMENTED
166     if(pixelIdx[ADNS2610_LEFT] <= PIXEL_QTY-1)  adns2610_sendByte(ADNS2610_LEFT,
       ADNS2610_PIXEL_DATA_REG);
167 #endif
168     /* While it waits the needed delay it's performed some tasks:
169      *  - Check the last received pixel status and take decision related to it
170      *  - Compute OF coefficients when it was possible
171      *  */
172     eyes_waitIT(ADNS2610_TIM_TO_RD);
173     if(!firstPixelRead){
174       pixelStatus[ADNS2610_RIGHT] = adns2610_checkPixel(&frames[currentFrameIdx].frame[
```

```
    ADNS2610_RIGHT][pixelIdx[ADNS2610_RIGHT]]);
175   #if SECOND_SENSOR_IMPLEMENTED
176       pixelStatus[ADNS2610_LEFT] = adns2610_checkPixel(&frames[currentFrameIdx].frame[ADNS2610_LEFT
      ][pixelIdx[ADNS2610_LEFT]]);
177   #endif
178       if(eyes_computeIdxFromStatus(&pixelStatus[ADNS2610_RIGHT], &pixelStatus[ADNS2610_LEFT], &
      pixelIdx[ADNS2610_RIGHT], &pixelIdx[ADNS2610_LEFT])){
179          FSMstate = READING_FRAME;
180          if((pixelStatus[ADNS2610_RIGHT] == NON_VALID) || (pixelStatus[ADNS2610_RIGHT] ==
      NON_VALID_SOF)){
181            errorCounter++;
182          }
183          if(!firstFrameRead){
184            OF_ComputeCoefficients(ADNS2610_RIGHT, frames[currentFrameIdx].frame[ADNS2610_RIGHT],
      frames[lastFrameIdx].frame[ADNS2610_RIGHT], pixelIdx[ADNS2610_RIGHT]);
185 #if SECOND_SENSOR_IMPLEMENTED
186            OF_ComputeCoefficients(ADNS2610_LEFT, frames[currentFrameIdx].frame[ADNS2610_LEFT], frames
      [lastFrameIdx].frame[ADNS2610_LEFT], pixelIdx[ADNS2610_LEFT]);
187 #endif
188          }
189        }
190        else{
191          eyes_stopWaitIT();
192          FSMstate = TRIGGER_FRAME;
193          eyes_waitIT(ADNS2610_TIM_BTW_WR);
194        }
195      }
196      else{
197        firstPixelRead = false;
198        FSMstate = READING_FRAME;
199      }
200      collisionFlag = 0;
201      return;
202    /* READING_FRAME state ------------------------------------------------------------------ */
203    case READING_FRAME:
204      eyes_stopWaitIT();
205      if(collisionFlag) goto collisionError; else collisionFlag = 1;
206      /* Read pixel data register */
207      if(pixelIdx[ADNS2610_RIGHT] <= PIXEL_QTY-1) adns2610_receiveByte(ADNS2610_RIGHT, &frames[
       currentFrameIdx].frame[ADNS2610_RIGHT][pixelIdx[ADNS2610_RIGHT]]);
208      /* Check the last pixel status. This is done because if all is good, the next state is
       PROCESSING, not REQ_READING_FRAME state */
209 #if SECOND_SENSOR_IMPLEMENTED
210      if(pixelIdx[ADNS2610_LEFT] <= PIXEL_QTY-1)  adns2610_receiveByte(ADNS2610_LEFT, &frames[
       currentFrameIdx].frame[ADNS2610_LEFT][pixelIdx[ADNS2610_LEFT]]);
211
212      if((pixelIdx[ADNS2610_RIGHT] == PIXEL_QTY-1) && pixelIdx[ADNS2610_LEFT] == PIXEL_QTY-1){
213        if(eyes_computeIdxFromStatus(&pixelStatus[ADNS2610_RIGHT], &pixelStatus[ADNS2610_LEFT], &
       pixelIdx[ADNS2610_RIGHT], &pixelIdx[ADNS2610_LEFT])){
214          FSMstate = PROCESSING;
215        }
216      }
217      else{
218        FSMstate = REQ_READING_FRAME;
219        eyes_waitIT(ADNS2610_TIM_BTW_RD);
220        collisionFlag = 0;
221        return;
222      }
223 #else
224      if(pixelIdx[ADNS2610_RIGHT] == PIXEL_QTY-1){
225        if(eyes_computeIdxFromStatus(&pixelStatus[ADNS2610_RIGHT], &pixelStatus[ADNS2610_LEFT], &
       pixelIdx[ADNS2610_RIGHT], &pixelIdx[ADNS2610_LEFT])){
226          FSMstate = PROCESSING;
227        }
228      }
```

```
229     else{
230       // It only waits for the next state when it is REQ_READING_FRAME. If not, it continues
      directly to PROCESSING state.
231       FSMstate = REQ_READING_FRAME;
232       eyes_waitIT(ADNS2610_TIM_BTW_RD);
233       collisionFlag = 0;
234       return;
235     }
236 #endif
237     collisionFlag = 0;
238   /* PROCESSING state ------------------------------------------------------------- */
239   case PROCESSING:
240     /* Check if it's the first frame read */
241     if(firstFrameRead){
242       firstFrameRead = false;
243     }
244     else{
245       /* Compute the Optical Flow from the previous computed coefficients */
246       OF_Compute(ADNS2610_RIGHT, &(frames[currentFrameIdx].oFRight.x), &(frames[currentFrameIdx].
      oFRight.y));
247 #if SECOND_SENSOR_IMPLEMENTED
248       OF_Compute(ADNS2610_LEFT, &(frames[currentFrameIdx].oFLeft.x), &(frames[currentFrameIdx].
      oFLeft.y));
249       OF_ComputeFused(&frames[currentFrameIdx].oFRight, &frames[currentFrameIdx].oFLeft, &frames[
      currentFrameIdx].oFFused);
250 #endif
251     }
252     /* Switch the frame structures to store the new frame in the "oldest" data buffer */
253     SWITCH_FRAME_IDX(currentFrameIdx, lastFrameIdx);
254     FSMstate = TRIGGER_FRAME;
255 
256     if(IsTrackingEnable()){
257       applyControlLaw(frames[currentFrameIdx].oFFused.x, frames[currentFrameIdx].oFFused.y, frames[
      currentFrameIdx].oFFused.theta);
258       eyes_waitControlTIM_IT(TIME_TO_POSITION);
259     }
260     else{
261       eyes_waitIT(ADNS2610_TIM_BTW_RD);
262     }
263     return;
264   }
265 
266   // Check for collisions between interrupts callings
267   collisionError:
268     printf("COLISSION ERROR!!\r\n");
269     eyes_stopWaitIT();
270     while(1);
271 }
272 
273 /**
274  * It sets up a TIMER to wait the required times by the ADNS2610 sensor through an interrupt.
275  * The TIMER pre-scaler is configured to increase its count each 250ns.
276  */
277 void eyes_configureFSM_TIM(void){
278   // TIM1 prescalers has been configured to count microseconds
279   uint32_t temp = TIM1->CR1;
280 
281   // Disable update interrupt
282   CLEAR_BIT(TIM1->DIER, TIM_DIER_UIE);
283   // Modify CR1 register
284   MODIFY_REG(temp, ~(TIM_CR1_UDIS), TIM_CR1_URS);
285   TIM1->CR1 = temp;
286   // Set interrupt interval
287   TIM1->ARR = ADNS2610_TIM_TO_RD;
288   // Update the prescaler and counter registers
```

```
289   SET_BIT(TIM1->EGR, TIM_EGR_UG);
290   // Clear pending interrupt flag
291   CLEAR_BIT(TIM1->SR, TIM_SR_UIF);
292   // Enable update interrupt generation
293   CLEAR_BIT(TIM1->CR1, TIM_CR1_URS);
294   // Enable update interrupt
295   SET_BIT(TIM1->DIER, TIM_DIER_UIE);
296   // Configure NVIC to handle TIM1 update interrupt
297   NVIC_SetPriority(TIM1_UP_TIM16_IRQn, 1);
298   NVIC_EnableIRQ(TIM1_UP_TIM16_IRQn);
299 }
300
301 /**
302  * It sets up the TIMER interval and it starts the count until the interrupt launch.
303  * @param Count250ns  The interval to wait expressed as 250ns multiples
304  */
305 void eyes_waitIT(uint32_t Count250ns){
306   // Disable update interrupt generation
307   SET_BIT(TIM1->CR1, TIM_CR1_URS);
308   // Set time to wait
309   TIM1->ARR = Count250ns;
310   // Update the prescaler and counter registers
311   SET_BIT(TIM1->EGR, TIM_EGR_UG);
312   // Enable update interrupt generation
313   CLEAR_BIT(TIM1->CR1, TIM_CR1_URS);
314   // Enable and start timer
315   SET_BIT(TIM1->CR1, TIM_CR1_CEN);
316 }
317
318 /**
319  * It stops the TIMER count to avoid the TIMER continues launching interrupts each configured
        interval time.
320  */
321 void eyes_stopWaitIT(){
322   // Disable and start timer
323   CLEAR_BIT(TIM1->CR1, TIM_CR1_CEN);
324 }
325
326 /**
327  * It sets up a TIMER to wait the required time to move the platform to the new position through an
        interrupt.
328  * The TIMER pre-scaler is configured to increase its count each millisecond.
329  */
330 void eyes_configureControl_TIM(void){
331   // TIM1 prescalers has been configured to count microseconds
332   uint32_t temp = TIM4->CR1;
333
334   // Disable update interrupt
335   CLEAR_BIT(TIM4->DIER, TIM_DIER_UIE);
336   // Modify CR1 register
337   MODIFY_REG(temp, ~(TIM_CR1_UDIS), TIM_CR1_URS);
338   TIM4->CR1 = temp;
339   // Set interrupt interval
340   TIM4->ARR = 1;
341   // Update the prescaler and counter registers
342   SET_BIT(TIM4->EGR, TIM_EGR_UG);
343   // Clear pending interrupt flag
344   CLEAR_BIT(TIM4->SR, TIM_SR_UIF);
345   // Enable update interrupt generation
346   CLEAR_BIT(TIM4->CR1, TIM_CR1_URS);
347   // Enable update interrupt
348   SET_BIT(TIM4->DIER, TIM_DIER_UIE);
349   // Configure NVIC to handle TIM1 update interrupt
350   NVIC_SetPriority(TIM4_IRQn, 1);
351   NVIC_EnableIRQ(TIM4_IRQn);
```

```
352 }
353
354 /**
355  * It sets up the TIMER interval and it starts the count until the interrupt launch
356  * @param millis  The interval to wait expressed in milliseconds
357  */
358 void eyes_waitControlTIM_IT(uint32_t millis){
359   // Disable update interrupt generation
360   SET_BIT(TIM4->CR1, TIM_CR1_URS);
361   // Set time to wait
362   TIM4->ARR = millis;
363   // Update the prescaler and counter registers
364   SET_BIT(TIM4->EGR, TIM_EGR_UG);
365   // Enable update interrupt generation
366   CLEAR_BIT(TIM4->CR1, TIM_CR1_URS);
367   // Enable and start timer
368   SET_BIT(TIM4->CR1, TIM_CR1_CEN);
369 }
370
371 /**
372  * It stops the TIMER count to avoid the TIMER continues launching interrupts each configured
      interval time.
373  */
374 void eyes_stopWaitControlTIM_IT(){
375   // Disable and start timer
376   CLEAR_BIT(TIM4->CR1, TIM_CR1_CEN);
377 }
378
379 /**
380  * It computes the next acquired pixel index:
381  *    - If the status is good, the index is increased by one.
382  *    - If global fault is detected the index is reset to zero.
383  *    - If local fault is detected the index isn't increased.
384  * @param status1 Pointer to pixel status type from one of the two devices
385  * @param status2 Pointer to pixel status type pixel status from one of the two devices
386  * @param idx1    Pointer to index for the next acquired pixel
387  * @param idx2    Poniter to index for the next acquired pixel
388  * @return  True if pixels status are good, false if not
389  */
390 bool eyes_computeIdxFromStatus(PixelStatus* status1, PixelStatus* status2, uint16_t* idx1,  uint16_t
      * idx2){
391
392   if((*status1 == VALID_SOF) && (*idx1 == 0)){
393     (*idx1)++;
394   }
395   else if((*status1 == VALID) && (*idx1 != 0) && (*idx1 < PIXEL_QTY-1)){
396     (*idx1)++;
397   }
398   else if ((*status1 == VALID_SOF) && (*idx1 != 0)){
399     *idx1 = *idx2 = 0;
400     return false;
401   }
402 #if SECOND_SENSOR_IMPLEMENTED
403   if((*status2 == VALID_SOF) && (*idx2 == 0)){
404     (*idx2)++;
405   }
406   else if((*status2 == VALID) && (*idx2 != 0) && (*idx2 < PIXEL_QTY-1)){
407     (*idx2)++;
408   }
409   else if((*status2 == VALID_SOF) && (*idx2 != 0)){
410     (*idx1) = (*idx2) = 0;
411     return false;
412   }
413 #endif
414   return true;
```

```
415 }
416
417 void TIM1_UP_TIM16_IRQHandler(void){
418   // If the interrupt flag is enabled
419   if(READ_BIT(TIM1->SR, TIM_SR_UIF)){
420     // Clear pending interrupt flag
421     CLEAR_BIT(TIM1->SR, TIM_SR_UIF);
422     // Process FSM
423     eyes_FSM();
424   }
425 }
426
427 void TIM4_IRQHandler(void){
428   // If the interrupt flag is enabled
429   if(READ_BIT(TIM4->SR, TIM_SR_UIF)){
430     // Clear pending interrupt flag
431     CLEAR_BIT(TIM4->SR, TIM_SR_UIF);
432     // Process FSM
433     eyes_FSM();
434   }
435 }
```

Anexo E

Librería para la implementación del control de la plataforma

E.1. Header

```
/*
 * gimbalControl.h
 *
 *  Created on: 27 oct. 2020
 *      Author: deros
 */

#ifndef INC_GIMBALCONTROL_H_
#define INC_GIMBALCONTROL_H_

/* Includes ------------------------------------------------------------------*/
#include "usart.h"
#include "string.h"
#include <stdlib.h>

/* Exported functions ------------------------------------------------------*/
void gimbalControlInit(void);
void applyControlLaw(int x, int y, int rotation);
bool IsTrackingEnable();

#endif /* INC_GIMBALCONTROL_H_ */
```

E.2. Source

```
/*
 * gimbalControl.c
 *
 *  Created on: 27 oct. 2020
 *      Author: deros
 */

/* Includes ------------------------------------------------------------------*/
#include "gimbalControl.h"

/* Private macro -------------------------------------------------------------*/
#define TAIL_CHAR '\n'
#define BUFFER_SIZE 10

```

```
15 /* DC PWM default and range values in CNT format*/
16 #define MIN_POS   3199    // 1 ms
17 #define CENTER_POS  4799    // 1.5 ms
18 #define MAX_POS   6399    // 2 ms
19 #define DELTA_POS 50
20
21 /* PID parameters*/
22 // PITCH
23 #define PITCH_P 0.45
24 #define PITCH_I 0.001
25 #define PITCH_D 0.001
26 // ROLL
27 #define ROLL_P  0.42
28 #define ROLL_I  0.001
29 #define ROLL_D  0.001
30 // YAW
31 #define YAW_P 0.55
32 #define YAW_I 0.001
33 #define YAW_D 0.001
34
35 #define DELTALIMIT 8
36
37 #define PITCH_WINDUP  500
38 #define ROLL_WINDUP   500
39 #define YAW_WINDUP    500
40
41 /* Private typedefs -----------------------------------------*/
42 typedef enum commandEnum{
43   UP,
44   DOWN,
45   LEFT,
46   RIGHT,
47   ROTATE_LEFT,
48   ROTATE_RIGHT,
49   CENTER,
50   TRACKING_ON,
51   TRACKING_OFF,
52   NA
53 } cmdTypeDef;
54
55 typedef struct{
56   uint16_t pitchPos;
57   uint16_t rollPos;
58   uint16_t yawPos;
59 }motorPosTypeDef;
60
61 /* Private variables ----------------------------------------*/
62 motorPosTypeDef motorPos = { .pitchPos = CENTER_POS, .rollPos = CENTER_POS, .yawPos = CENTER_POS};
63 bool pwmEn;
64 bool trackingEn;
65
66 static int xSum = 0, ySum = 0, rotationSum = 0;
67 static int xLast = 0, yLast = 0, rotationLast = 0;
68
69 /* Private functions ----------------------------------------*/
70 cmdTypeDef decodeCmd(char const * cmdString, int length);
71 void enablePWM();
72 void disablePWM();
73 __STATIC_INLINE void NormalizeRange(int value, int MaxRange, int MinRange);
74
75 /**
76  * @brief Setting up all the peripherals (UART and TIMER) needed
77  * to control the gimbal position
78  */
79 void gimbalControlInit(void){
```

```
80    // Configure UART2 interrupt to receive data from PC
81    configure_IRQ_USART_RX();
82
83    // Flag to know PWM signal state
84    pwmEn = false;
85
86    // Flag to know if tracking function is enable/disable
87    trackingEn = false;
88  }
89
90  /* ------------------------------------------------
91   * PWM OUTPUTS:
92   *    - RC-1: Pith  -->   TIM3->CH1 --> PA7
93   *    - RC-2: Roll  -->   TIM3->CH2 --> PA6
94   *    - RC-3: Yaw   -->   TIM3->CH4 --> PB1
95   * ------------------------------------------------
96   */
97  /**
98   * @brief Receive a string and decode the command type related to it
99   * @param cmdString The command in string format
100  * @param length  The length of the command
101  * @return  The command type in cmdTypeDef format
102  */
103 cmdTypeDef decodeCmd(char const * cmdString, int length){
104
105   // Enable PWM if it was disabled
106   if(!pwmEn)  enablePWM();
107
108   // Tracking enable command
109   if(strncmp(cmdString, "TRON\n", length) == 0){
110     trackingEn = true;
111     return TRACKING_ON;
112   }
113   if(strncmp(cmdString, "TROFF\n", length) == 0){
114     trackingEn = false;
115
116     return TRACKING_OFF;
117   }
118
119   // Tracking enable so It isn't able to perform any command
120   if(trackingEn) return NA;
121
122   // Center command
123   if(strncmp(cmdString, "CN\n", length) == 0){
124
125     motorPos.pitchPos = CENTER_POS;
126     motorPos.rollPos = CENTER_POS;
127     motorPos.yawPos = CENTER_POS;
128
129     TIM3->CCR1 = motorPos.pitchPos;
130     TIM3->CCR2 = motorPos.rollPos;
131     TIM3->CCR4 = motorPos.yawPos;
132
133     return CENTER;
134   }
135
136   // Up command
137   if(strncmp(cmdString, "UP\n", length) == 0){
138     motorPos.pitchPos -= DELTA_POS;
139     NormalizeRange(motorPos.pitchPos, MAX_POS, MIN_POS);
140     TIM3->CCR2 = motorPos.pitchPos;
141     return UP;
142   }
143   // Down command
144   if(strncmp(cmdString, "DW\n", length) == 0){
```

```
145     motorPos.pitchPos += DELTA_POS;
146     NormalizeRange(motorPos.pitchPos, MAX_POS, MIN_POS);
147     TIM3->CCR2 = motorPos.pitchPos;
148     return DOWN;
149   }
150   // Left command
151   if(strncmp(cmdString, "LF\n", length) == 0){
152     motorPos.yawPos -= DELTA_POS;
153     NormalizeRange(motorPos.yawPos, MAX_POS, MIN_POS);
154     TIM3->CCR4 = motorPos.yawPos;
155     return LEFT;
156   }
157   // Right command
158   if(strncmp(cmdString, "RH\n", length) == 0){
159     motorPos.yawPos += DELTA_POS;
160     NormalizeRange(motorPos.yawPos, MAX_POS, MIN_POS);
161     TIM3->CCR4 = motorPos.yawPos;
162     return RIGHT;
163   }
164
165   // Rotate left command
166   if(strncmp(cmdString, "RLF\n", length) == 0){
167     motorPos.rollPos += DELTA_POS;
168     NormalizeRange(motorPos.rollPos, MAX_POS, MIN_POS);
169     TIM3->CCR1 = motorPos.rollPos;
170     return ROTATE_LEFT;
171   }
172   // Rotate right command
173   if(strncmp(cmdString, "RRH\n", length) == 0){
174     motorPos.rollPos -= DELTA_POS;
175     NormalizeRange(motorPos.rollPos, MAX_POS, MIN_POS);
176     TIM3->CCR1 = motorPos.rollPos;
177     return ROTATE_RIGHT;
178   }
179   return NA;
180 }
181
182 /**
183  * \brief Implements the control law from the optical flow received as
184  * arguments
185  * @param x Optical flow value in horizontal direction
186  * @param y Optical flow value in vertical direction
187  * @param rotation  Optical flow value which indicates the rotation
188  */
189 void applyControlLaw(int x, int y, int rotation){
190   int deltaPitch, deltaRoll, deltaYaw;
191
192   if(!trackingEn) return;
193
194   // Integrate values
195   xSum += x;
196   ySum += y;
197   rotationSum += rotation;
198
199   // Integral windup
200   if(abs(ySum) > PITCH_WINDUP) ySum = 0;
201   if(abs(rotationSum) > ROLL_WINDUP) rotationSum = 0;
202   if(abs(xSum) > YAW_WINDUP) xSum = 0;
203
204   // PID implementation
205   deltaYaw = (float)(x * YAW_P) + (float)(xSum * YAW_I) + (float)((x - xLast) * YAW_D);
206   deltaPitch = y * PITCH_P + ySum * PITCH_I + (y - yLast) * PITCH_D;
207   deltaRoll = rotation * ROLL_P + rotationSum *  ROLL_I +
208       (rotation - rotationLast) * ROLL_D;
209
```

```
210   // Avoid small changes in computed values
211   if(abs(deltaYaw)>=DELTALIMIT) motorPos.yawPos +=deltaYaw;
212   if(abs(deltaPitch)>=DELTALIMIT) motorPos.pitchPos +=deltaPitch;
213   if(abs(deltaRoll)>=DELTALIMIT) motorPos.rollPos +=deltaRoll;
214
215   // Check the signals are in the proper range
216   NormalizeRange(motorPos.pitchPos, MAX_POS, MIN_POS);
217   NormalizeRange(motorPos.rollPos, MAX_POS, MIN_POS);
218   NormalizeRange(motorPos.yawPos, MAX_POS, MIN_POS);
219
220   // RC control signals generation
221   TIM3->CCR1 = motorPos.rollPos;
222   TIM3->CCR2 = motorPos.pitchPos;
223   TIM3->CCR4 = motorPos.yawPos;
224
225   // Save values to differentiation
226   xLast = x;
227   yLast = y;
228   rotationLast = rotation;
229 }
230
231 /**
232  * @brief Check if tracking function is enable/disable
233  * @return  True if the tracking function is enable, False if it is disable
234  */
235 bool IsTrackingEnable(){
236   return trackingEn;
237 }
238
239 /**
240  * @brief Enable RC (PWM) signal generation
241  */
242 void enablePWM(){
243   // Enable output compare OCx channels
244   //SET_BIT(TIM3->CCER, TIM_CCER_CC4E);
245   MODIFY_REG(TIM3->CCER, ~(TIM_CCER_CC1NE | TIM_CCER_CC2NE),
246       (TIM_CCER_CC1E | TIM_CCER_CC2E | TIM_CCER_CC4E));
247
248   // Enable master output
249   MODIFY_REG(TIM3->BDTR, ~(TIM_BDTR_OSSI | TIM_BDTR_OSSR), TIM_BDTR_MOE);
250
251   // Enable counter
252   SET_BIT(TIM3->CR1, TIM_CR1_CEN);
253
254   pwmEn = true;
255 }
256
257 /**
258  * @brief Disable RC (PWM) signal generation
259  */
260 void disablePWM(){
261   // Disable output
262   //CLEAR_BIT(TIM3->CCER, TIM_CCER_CC4E);
263   CLEAR_BIT(TIM3->CCER, (TIM_CCER_CC1E | TIM_CCER_CC2E | TIM_CCER_CC4E));
264
265   // Disable master output
266   CLEAR_BIT(TIM3->BDTR, TIM_BDTR_MOE);
267
268   // Disable counter
269   CLEAR_BIT(TIM3->CR1, TIM_CR1_CEN);
270
271   // Reset motor position struct to initial values
272   motorPos.pitchPos = CENTER_POS;
273   motorPos.rollPos = CENTER_POS;
274   motorPos.yawPos = CENTER_POS;
```

```
275
276   pwmEn = false;
277 }
278
279 /**
280  * @brief Avoid out of range values in RC signals
281  * @param value Current value of signal
282  * @param MaxRange  Maximum value for signal
283  * @param MinRange  Minimum value for signal
284  */
285 void NormalizeRange(int value, int MaxRange, int MinRange){
286   if(value > MaxRange){
287     value = MaxRange;
288   }
289   else if(value < MinRange){
290     value = MinRange;
291   }
292 }
293
294 void USART2_IRQHandler(void){
295   static char bufferIn[BUFFER_SIZE];
296   static int i = 0;
297
298   if(READ_BIT(USART2->ISR, USART_ISR_ORE)){
299     SET_BIT(USART2->ICR, USART_ICR_ORECF);
300     // Flush all data in USART RX
301     SET_BIT(USART2->RQR, USART_RQR_RXFRQ);
302   }
303   if(READ_BIT(USART2->ISR, USART_ISR_RXNE)){
304     bufferIn[i] = READ_REG(USART2->RDR);
305
306     if(bufferIn[i] == TAIL_CHAR){
307       i++;
308       decodeCmd(bufferIn, i);
309       i = 0;
310     }
311     else if(i ==  BUFFER_SIZE){
312       i = 0;
313     }
314     else{
315       i++;
316     }
317   }
318
319 }
```

Anexo F

Descripción en XAML de la interfaz

```
<Window x:Class="OFGimbalPlatform.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:i="clr-namespace:System.Windows.Interactivity;assembly=System.Windows.Interactivity"
        xmlns:materialDesign="http://materialdesigninxaml.net/winfx/xaml/themes"
        xmlns:vm="clr-namespace:OFGimbalPlatform.VM"
        xmlns:views="clr-namespace:OF_gimbal_platform.View"
        xmlns:ports="clr-namespace:System.IO.Ports;assembly=System"
        xmlns:tools="clr-namespace:Tools"
        xmlns:local="clr-namespace:OFGimbalPlatform"
        mc:Ignorable="d"
        Name="Window1"
        Title="Eyes OF Gimbal Platform" Height="768" Width="1024"
        WindowStartupLocation="CenterScreen"
        TextElement.Foreground="{DynamicResource MaterialDesignBody}"
        TextElement.FontWeight="Regular"
        TextElement.FontSize="13"
        TextOptions.TextFormattingMode="Ideal"
        TextOptions.TextRenderingMode="Auto"
        Background="{DynamicResource MaterialDesignPaper}"
        FontFamily="{DynamicResource MaterialDesignFont}">

    <Window.DataContext>
        <vm:ViewModel/>
```

```
27      </Window.DataContext>
28
29      <Window.Resources>
30          <ObjectDataProvider ObjectType="{x:Type ports:SerialPort}" MethodName="GetPortNames" x:Key="portNames" />
31          <tools:BoolToSelectedSlide x:Key="BoolToSelectedSlide"/>
32          <tools:BooleanToBadge x:Key="BooleanToBadge"/>
33      </Window.Resources>
34
35      <i:Interaction.Triggers>
36          <i:EventTrigger EventName="Closing">
37              <i:InvokeCommandAction Command="{Binding ExitCommand}" />
38          </i:EventTrigger>
39      </i:Interaction.Triggers>
40
41      <Grid>
42          <Grid.ColumnDefinitions>
43              <ColumnDefinition Width="*"/>
44              <ColumnDefinition Width="*"/>
45              <ColumnDefinition Width="*"/>
46              <ColumnDefinition Width="*"/>
47              <ColumnDefinition Width="*"/>
48          </Grid.ColumnDefinitions>
49          <Grid.RowDefinitions>
50              <RowDefinition Height="0.6*"/>
51              <RowDefinition Height="2*"/>
52              <RowDefinition Height="2*"/>
53              <RowDefinition Height="*"/>
54          </Grid.RowDefinitions>
55
56          <materialDesign:ColorZone Grid.ColumnSpan="5" Mode="PrimaryDark" Padding="16" Margin="1 1 1 0" CornerRadius="10" ClipToBounds="False"
57                                    materialDesign:ShadowAssist.ShadowDepth="Depth3">
58              <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
59                  <TextBlock Style="{StaticResource MaterialDesignHeadline4TextBlock}" VerticalAlignment="Center" Margin="16 0 0 0">
60                      Eyes OF Gimbal Platform</TextBlock>
61              </StackPanel>
62          </materialDesign:ColorZone>
63          <views:ADNS2610_monitoring Grid.Column="2" Grid.Row="1" Grid.RowSpan="2" Grid.ColumnSpan="2" ImageSource="{Binding SensorData.ImageRight}"
64                                     Title="Right eye" OFX="{Binding SensorData.OFRight.X}" OFY="{Binding SensorData.OFRight.Y}"
65                                     Visibility="{Binding IsStarted, Converter={StaticResource BooleanToVisibilityConverter}}" OFEnable="{Binding IsOFEnable}"
66                                     OFXFiltered="{Binding SensorData.OFRightFiltered.X}" OFYFiltered="{Binding SensorData.OFRightFiltered.Y}"/>
67          <views:ADNS2610_monitoring Grid.Column="0" Grid.Row="1" Grid.RowSpan="2" Grid.ColumnSpan="2" ImageSource="{Binding SensorData.ImageLeft}"
68                                     Title="Left eye" OFX="{Binding SensorData.OFLeft.X}" OFY="{Binding SensorData.OFLeft.Y}"
69                                     Visibility="{Binding IsStarted, Converter={StaticResource BooleanToVisibilityConverter}}" OFEnable="{Binding IsOFEnable}"
70                                     OFXFiltered="{Binding SensorData.OFLeftFiltered.X}" OFYFiltered="{Binding SensorData.OFLeftFiltered.Y}"/>
71
72          <Grid Grid.Column="5" Grid.Row="1" Grid.RowSpan="3">
73              <Grid.RowDefinitions>
```

```
74              <RowDefinition Height="1.3*"/>
75              <RowDefinition Height="*"/>
76              <RowDefinition Height="*"/>
77              <RowDefinition Height="1.7*"/>
78              <RowDefinition Height="*"/>
79          </Grid.RowDefinitions>
80          <Grid>
81              <Grid.RowDefinitions>
82                  <RowDefinition Height="1.1*"/>
83                  <RowDefinition Height="*"/>
84              </Grid.RowDefinitions>
85              <materialDesign:Transitioner SelectedIndex="{Binding IsStarted, Converter={StaticResource BoolToSelectedSlide}}"
86                                           AutoApplyTransitionOrigins="True" IsEnabled="{Binding StartIsEnable}">
87                  <Button Style="{StaticResource MaterialDesignRaisedAccentButton}" Height="40"
88                          materialDesign:ButtonAssist.CornerRadius="10"  HorizontalAlignment="Center" VerticalAlignment="Center"
89                          ToolTip="START the sensors monitor." Margin="0 20 0 0" IsEnabled="{Binding StartIsEnable}"
90                          Command="{Binding StartCmd}">
91                      <TextBlock Text="START" Style="{StaticResource MaterialDesignHeadline5TextBlock}"/>
92                  </Button>
93                  <Button Style="{StaticResource MaterialDesignRaisedAccentButton}" Height="40"
94                          materialDesign:ButtonAssist.CornerRadius="10" HorizontalAlignment="Center" VerticalAlignment="Center"
95                          ToolTip="STOP the sensors monitor." Margin="0 20 0 0" Command="{Binding StopCmd}"
96                          IsEnabled="{Binding StartIsEnable}">
97                      <TextBlock Text="STOP" Style="{StaticResource MaterialDesignHeadline5TextBlock}"/>
98                  </Button>
99              </materialDesign:Transitioner>
100             <ComboBox Grid.Row="1" Style="{StaticResource MaterialDesignFloatingHintComboBox}" materialDesign:HintAssist.Hint="COM"
101                       materialDesign:ColorZoneAssist.Mode="Light" Margin="40 0 40 0" materialDesign:HintAssist.HelperText="Select one Serial Port"
102                       materialDesign:TextFieldAssist.UnderlineBrush="{DynamicResource SecondaryHueMidBrush}"
103                       ItemsSource="{Binding Source={StaticResource portNames}}" SelectedItem="{Binding COM_sel}">
104             </ComboBox>
105         </Grid>
106         <materialDesign:Badged Grid.Row="1" BadgeColorZoneMode="PrimaryMid" HorizontalAlignment="Center" Margin="0"
107                                VerticalAlignment="Bottom" Badge="{Binding IsOFEnable, Converter={StaticResource BooleanToBadge}}" IsEnabled="{Binding IsStarted
    }">
108             <Button Style="{StaticResource MaterialDesignRaisedAccentButton}" Height="60" materialDesign:ButtonAssist.CornerRadius="15"
109                     ToolTip="Show ON/OFF optical flow" IsEnabled="{Binding IsStarted}" Command="{Binding OFEnableCmd}">
110                 <TextBlock Text="OPTICAL FLOW" Style="{StaticResource MaterialDesignHeadline6TextBlock}"/>
111             </Button>
112         </materialDesign:Badged>
113         <materialDesign:Badged Grid.Row="2" BadgeColorZoneMode="PrimaryMid" HorizontalAlignment="Center" Margin="20 20 20 0"
114                                VerticalAlignment="Top" Badge="{Binding IsCalibrationMode, Converter={StaticResource BooleanToBadge}}"
115                                IsEnabled="{Binding IsOFEnable}">
116             <Button Style="{StaticResource MaterialDesignRaisedAccentButton}" Height="50" materialDesign:ButtonAssist.CornerRadius="15"
117                     ToolTip="Set Reference Frame" IsEnabled="{Binding IsOFEnable}" Command="{Binding CalibrationCmd}">
118                 <TextBlock Text="CALIBRATION" Style="{StaticResource MaterialDesignHeadline6TextBlock}"/>
119             </Button>
```

```
120             </materialDesign:Badged>
121             <views:Joystick Grid.Row="3" IsEnable="{Binding IsStarted}" Height="180" Width="180" Margin="0 0 0 10" VerticalAlignment="Bottom"
122                             UpCommand="{Binding UpGimbalCmd}" DownCommand="{Binding DownGimbalCmd}"
123                             LeftCommand="{Binding LeftGimbalCmd}" RightCommand="{Binding RightGimbalCmd}"
124                             RotateLeftCommand="{Binding RotateLeftGimbalCmd}" RotateRightCommand="{Binding RotateRightGimbalCmd}"
125                             CenterCommand="{Binding CenterGimbalCmd}"/>
126             <materialDesign:Badged x:Name="CountingBadge" Grid.Row="4" BadgeColorZoneMode="PrimaryMid" HorizontalAlignment="Center" Margin="0 15 0 0"
127                                    VerticalAlignment="Top" Badge="{Binding IsTrackingEnable, Converter={StaticResource BooleanToBadge}}" IsEnabled="{Binding
        IsStarted}">
128                 <Button Style="{StaticResource MaterialDesignRaisedAccentButton}" Height="60" materialDesign:ButtonAssist.CornerRadius="10"
129                         ToolTip="Command ON/OFF trancking mode." IsEnabled="{Binding IsStarted}" Command="{Binding TrackingEnableCmd}">
130                     <TextBlock Text="TRACKING" Style="{StaticResource MaterialDesignHeadline5TextBlock}"/>
131                 </Button>
132             </materialDesign:Badged>
133         </Grid>
134 
135         <views:OFFusedSection Grid.Row="3" Grid.ColumnSpan="4" OFX="{Binding SensorData.OFFussed.X}"
136                               OFY="{Binding SensorData.OFFussed.Y}" Rotation="{Binding SensorData.OFFussed.Theta}"
137                               InheritedVisibility="{Binding IsOFEnable, Converter={StaticResource BooleanToVisibilityConverter}}"/>
138     </Grid>
139 </Window>
```

Anexo G

Descripción en XAML de la vista para monitorización de la imagen y el flujo óptico


```
<UserControl x:Class="OF_gimbal_platform.View.ADNS2610_monitoring"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:materialDesign="http://materialdesigninxaml.net/winfx/xaml/themes"
             xmlns:tools="clr-namespace:Tools"
             xmlns:petzold="clr-namespace:Petzold.Media2D"
             xmlns:local="clr-namespace:OF_gimbal_platform.View"
             mc:Ignorable="d"
             Name="ADNS2610_monitor"
             d:DesignHeight="450" d:DesignWidth="400">

    <UserControl.Resources>
        <tools:DimensionToCenter x:Key="DimensionToCenter"/>
        <tools:OpticalFlowToForwardPixels x:Key="OpticalFlowToForwardPixels"/>
        <tools:OpticalFlowToReversePixels x:Key="OpticalFlowToReversePixels"/>
        <tools:BooleanToVisibility x:Key="BooleanToVisibility"/>
    </UserControl.Resources>

    <Grid DataContext="{Binding ElementName=ADNS2610_monitor}">
        <GroupBox  Header="{Binding Title}"  Style="{DynamicResource MaterialDesignCardGroupBox}"  Margin="5"
                   materialDesign:ShadowAssist.ShadowDepth="Depth2" Padding="2" BorderThickness="0" materialDesign:ColorZoneAssist.Mode="PrimaryMid">
```

```
24              <GroupBox.HeaderTemplate>
25                  <DataTemplate>
26                      <StackPanel Orientation="Horizontal">
27                          <materialDesign:PackIcon Kind="Eye" Height="32" Width="32" VerticalAlignment="Center" Margin="20 0 0 0"/>
28                          <TextBlock Margin="8,15,0,15" VerticalAlignment="Center" Style="{StaticResource MaterialDesignSubtitle1TextBlock}" Text="{Binding}" />
29                      </StackPanel>
30                  </DataTemplate>
31              </GroupBox.HeaderTemplate>
32              <Grid>
33                  <Grid.RowDefinitions>
34                      <RowDefinition Height="4*"/>
35                      <RowDefinition Height="*"/>
36                  </Grid.RowDefinitions>
37                  <Grid.ColumnDefinitions>
38                      <ColumnDefinition Width="*"/>
39                      <ColumnDefinition Width="*"/>
40                  </Grid.ColumnDefinitions>
41                  <Image Grid.Row="0" Grid.ColumnSpan="2" Source="{Binding ImageSource.Source}"
42                         Stretch="Uniform" Margin="3">
43                  </Image>
44                  <Canvas x:Name="Canvas"  Grid.Row="0" Grid.ColumnSpan="2" Visibility="{Binding OFEnable, Converter={StaticResource BooleanToVisibility}}">
45                      <petzold:ArrowLine x:Name="Arrow"
46                                         X1="{Binding ElementName=Canvas, Path=ActualWidth, Converter={StaticResource DimensionToCenter}}"
47                                         Y1="{Binding ElementName=Canvas, Path=ActualHeight, Converter={StaticResource DimensionToCenter}}"
48                                         ArrowEnds="End" StrokeThickness="3" Stroke="Red">
49                          <petzold:ArrowLine.X2>
50                              <MultiBinding Converter="{StaticResource OpticalFlowToForwardPixels}" ConverterParameter="X">
51                                  <Binding ElementName="Arrow" Path="X1"/>
52                                  <Binding ElementName="Arrow" Path="Y1"/>
53                                  <Binding Path="OFXFiltered"/>
54                                  <Binding Path="OFYFiltered"/>
55                              </MultiBinding>
56                          </petzold:ArrowLine.X2>
57                          <petzold:ArrowLine.Y2>
58                              <MultiBinding Converter="{StaticResource OpticalFlowToForwardPixels}" ConverterParameter="Y">
59                                  <Binding ElementName="Arrow" Path="X1"/>
60                                  <Binding ElementName="Arrow" Path="Y1"/>
61                                  <Binding Path="OFXFiltered"/>
62                                  <Binding Path="OFYFiltered"/>
63                              </MultiBinding>
64                          </petzold:ArrowLine.Y2>
65                      </petzold:ArrowLine>
66                      <petzold:ArrowLine X1="{Binding ElementName=Canvas, Path=ActualWidth, Converter={StaticResource DimensionToCenter}}"
67                                         Y1="{Binding ElementName=Canvas, Path=ActualHeight, Converter={StaticResource DimensionToCenter}}"
68                                         ArrowEnds="End" StrokeThickness="3" Stroke="Green">
69                          <petzold:ArrowLine.X2>
70                              <MultiBinding Converter="{StaticResource OpticalFlowToReversePixels}" ConverterParameter="X">
```

```
71                              <Binding ElementName="Arrow" Path="X1"/>
72                              <Binding ElementName="Arrow" Path="Y1"/>
73                              <Binding Path="OFXFiltered"/>
74                              <Binding Path="OFYFiltered"/>
75                          </MultiBinding>
76                      </petzold:ArrowLine.X2>
77                      <petzold:ArrowLine.Y2>
78                          <MultiBinding Converter="{StaticResource OpticalFlowToReversePixels}" ConverterParameter="Y">
79                              <Binding ElementName="Arrow" Path="X1"/>
80                              <Binding ElementName="Arrow" Path="Y1"/>
81                              <Binding Path="OFXFiltered"/>
82                              <Binding Path="OFYFiltered"/>
83                          </MultiBinding>
84                      </petzold:ArrowLine.Y2>
85                  </petzold:ArrowLine>
86              </Canvas>
87              <GroupBox Grid.Row="1" Style="{DynamicResource MaterialDesignCardGroupBox}" Margin="10" UseLayoutRounding="False" SnapsToDevicePixels="True"
88                        BorderThickness="0" materialDesign:ShadowAssist.ShadowDepth="Depth3" Padding="2" materialDesign:ColorZoneAssist.Mode="PrimaryLight"
89                        Visibility="{Binding OFEnable, Converter={StaticResource BooleanToVisibility}}">
90                  <GroupBox.HeaderTemplate>
91                      <DataTemplate>
92                          <TextBlock Margin="5" VerticalAlignment="Center" HorizontalAlignment="Center"
93                                     Style="{StaticResource MaterialDesignSubtitle2TextBlock}" Text="Optical Flow in X" />
94                      </DataTemplate>
95                  </GroupBox.HeaderTemplate>
96                  <TextBlock Style="{StaticResource MaterialDesignSubtitle2TextBlock}" Text="{Binding OFX}" VerticalAlignment="Center"
97                             HorizontalAlignment="Center"/>
98              </GroupBox>
99              <GroupBox Grid.Row="1" Grid.Column="1" Style="{DynamicResource MaterialDesignCardGroupBox}" Margin="10" UseLayoutRounding="False"
    SnapsToDevicePixels="True"
100                       BorderThickness="0" materialDesign:ShadowAssist.ShadowDepth="Depth3" Padding="2" materialDesign:ColorZoneAssist.Mode="PrimaryLight"
101                       Visibility="{Binding OFEnable, Converter={StaticResource BooleanToVisibility}}">
102                 <GroupBox.HeaderTemplate>
103                     <DataTemplate>
104                         <TextBlock Margin="5" VerticalAlignment="Center" HorizontalAlignment="Center"
105                                    Style="{StaticResource MaterialDesignSubtitle2TextBlock}" Text="Optical Flow in Y" />
106                     </DataTemplate>
107                 </GroupBox.HeaderTemplate>
108                 <TextBlock Style="{StaticResource MaterialDesignSubtitle2TextBlock}" Text="{Binding OFY}" VerticalAlignment="Center"
109                            HorizontalAlignment="Center"/>
110             </GroupBox>
111         </Grid>
112       </GroupBox>
113   </Grid>
114 </UserControl>
```

Anexo H

'View Model' de la aplicación 'Eyes OF Gimbal Platform'

```
using System;
using System.ComponentModel;
using System.Diagnostics;
using System.IO.Ports;
using System.Threading;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Media.Imaging;
using FrameWrapper;
using FrameWrapper.Wrappers;
using OF_gimbal_platform.Model;
using OFGimbalPlatform.Base;
using OFGimbalPlatform.Model;

namespace OFGimbalPlatform.VM
{
    public class ViewModel : BindableBase
    {
        /* Private data ------------------------------------------------------- */
        private SerialPort COM;
        private readonly ADNS2610_wrapper<ADNS2610_Packet> adns2610Provider;
        private SerialSTM32GimbalCommander commander;

        /* Bindable data ------------------------------------------------------ */
        private BindableSensorData _sensorData;
        public BindableSensorData SensorData
        {
            get { return _sensorData; }
            set { SetProperty(ref _sensorData, value); }
        }
        private string _COM_sel;
        public string COM_sel
        {
            get { return _COM_sel; }
            set {
                SetProperty(ref _COM_sel, value);
                OnCOMSelectedChanged();
            }
        }
        private bool _StartIsEnable;
        public bool StartIsEnable
        {
```

```
43              get { return _StartIsEnable; }
44              set { SetProperty(ref _StartIsEnable, value); }
45          }
46          private bool _IsStarted;
47          public bool IsStarted
48          {
49              get { return _IsStarted; }
50              set { SetProperty(ref _IsStarted, value); }
51          }
52          private bool _IsOFEnable;
53          public bool IsOFEnable
54          {
55              get { return _IsOFEnable; }
56              set { SetProperty(ref _IsOFEnable, value); }
57          }
58          private bool _IsCalibrationMode;
59          public bool IsCalibrationMode
60          {
61              get { return _IsCalibrationMode; }
62              set { SetProperty(ref _IsCalibrationMode, value); }
63          }
64          private bool _IsTackingEnable;
65          public bool IsTrackingEnable
66          {
67              get { return _IsTackingEnable; }
68              set { SetProperty(ref _IsTackingEnable, value); }
69          }
70
71          /* Logic and tasks ----------------------------------------------------*/
72          readonly CancellationTokenSource exitAppSource;
73          readonly CancellationToken exitAppToken;
74          readonly Task FrameUpdateTask;
75          readonly Task OFFilteringTask;
76          readonly SemaphoreSlim FrameUpdateSem;
77
78          /* UI commands --------------------------------------------------------*/
79          private DelegateCommand _exitCommand;
80          public DelegateCommand ExitCommand
81          {
82              get { return _exitCommand; }
83              set { SetProperty(ref _exitCommand, value); }
84          }
85          private DelegateCommand _StartCmd;
86          public DelegateCommand StartCmd
87          {
88              get { return _StartCmd; }
89              set { SetProperty(ref _StartCmd, value); }
90          }
91          private DelegateCommand _StopCmd;
92          public DelegateCommand StopCmd
93          {
94              get { return _StopCmd; }
95              set { SetProperty(ref _StopCmd, value); }
96          }
97          private DelegateCommand _OFEnableCmd;
98          public DelegateCommand OFEnableCmd
99          {
100             get { return _OFEnableCmd; }
101             set { SetProperty(ref _OFEnableCmd, value); }
102         }
103         private DelegateCommand _calibrationCmd;
104         public DelegateCommand CalibrationCmd
105         {
106             get { return _calibrationCmd; }
107             set { SetProperty(ref _calibrationCmd, value); }
```

```
108         }
109         private DelegateCommand _TrackingEnableCmd;
110         public DelegateCommand TrackingEnableCmd
111         {
112             get { return _TrackingEnableCmd; }
113             set { SetProperty(ref _TrackingEnableCmd, value); }
114         }
115         private DelegateCommand _upGimbalCmd;
116         public DelegateCommand UpGimbalCmd
117         {
118             get { return _upGimbalCmd; }
119             set { SetProperty(ref _upGimbalCmd, value); }
120         }
121         private DelegateCommand _downGimbalCmd;
122         public DelegateCommand DownGimbalCmd
123         {
124             get { return _downGimbalCmd; }
125             set { SetProperty(ref _downGimbalCmd, value); }
126         }
127         private DelegateCommand _leftGimbalCmd;
128         public DelegateCommand LeftGimbalCmd
129         {
130             get { return _leftGimbalCmd; }
131             set { SetProperty(ref _leftGimbalCmd, value); }
132         }
133         private DelegateCommand _rightGimbalCmd;
134         public DelegateCommand RightGimbalCmd
135         {
136             get { return _rightGimbalCmd; }
137             set { SetProperty(ref _rightGimbalCmd, value); }
138         }
139         private DelegateCommand _rotateLeftGimbalCmd;
140         public DelegateCommand RotateLeftGimbalCmd
141         {
142             get { return _rotateLeftGimbalCmd; }
143             set { SetProperty(ref _rotateLeftGimbalCmd, value); }
144         }
145         private DelegateCommand _rotateRightGimbalCmd;
146         public DelegateCommand RotateRightGimbalCmd
147         {
148             get { return _rotateRightGimbalCmd; }
149             set { SetProperty(ref _rotateRightGimbalCmd, value); }
150         }
151         private DelegateCommand _centerGimbalCmd;
152         public DelegateCommand CenterGimbalCmd
153         {
154             get { return _centerGimbalCmd; }
155             set { SetProperty(ref _centerGimbalCmd, value); }
156         }
157 
158         /// <summary>
159         /// View Model constructor
160         /// </summary>
161         public ViewModel()
162         {
163 #if DEBUG
164             /* Avoid the use of this class from code designer. This resolves the COM denied access
        problem. */
165             if (DesignerProperties.GetIsInDesignMode(new DependencyObject())) return;
166 #endif
167             /* Data declaration ------------- */
168             SensorData = new BindableSensorData();
169             COM_sel = "";
170             StartIsEnable = false;
171             IsStarted = false;
```

```
172             IsOFEnable = false;
173             IsCalibrationMode = false;
174             IsTrackingEnable = false;
175 
176             adns2610Provider = new ADNS2610_wrapper<ADNS2610_Packet>(ADNS2610_Packet.PacketSize *
    100, ADNS2610_Packet.Header, ADNS2610_Packet.PacketSize, true);
177             adns2610Provider.FrameAvailableEvent += ImageSource_FrameAvailableEvent;
178 
179             /* Logic and tasks declaration --------- */
180             exitAppSource = new CancellationTokenSource();
181             exitAppToken = exitAppSource.Token;
182             FrameUpdateSem = new SemaphoreSlim(0, 10);
183 
184             /* UI commands declaration  ------------ */
185             AssignDelegateCmd();
186 
187             /* Starting processes ------------------ */
188             FrameUpdateTask = Task.Run(() => FrameUpdateTaskMethod(), exitAppToken);
189             OFFilteringTask = Task.Run(() => OFFilteringTaskMethod(), exitAppToken);
190 
191             adns2610Provider.Start();
192         }
193 
194         #region Tasks
195         /// <summary>
196         /// In this task the frames detected in the input stream are updated in the GUI
197         /// </summary>
198         /// <returns>Task class to be managed by the app</returns>
199         private async Task FrameUpdateTaskMethod()
200         {
201             while (!exitAppToken.IsCancellationRequested)
202             {
203                 await FrameUpdateSem.WaitAsync().ConfigureAwait(true);
204 
205                 if (exitAppToken.IsCancellationRequested) break;
206 
207                 await Application.Current.Dispatcher.BeginInvoke(new Action(() =>
208                 {
209                     try
210                     {
211                         SensorData.GetRigthImageSource().Lock();
212                         SensorData.GetLeftImageSource().Lock();
213 
214                         IntPtr pBackBufferRight = SensorData.GetRigthImageSource().BackBuffer;
215                         IntPtr pBackBufferLeft = SensorData.GetLeftImageSource().BackBuffer;
216 
217                         unsafe
218                         {
219                             FillFrame(SensorData.GetRigthImageSource(), SensorData.GetRightBuffer())
    ;
220                             FillFrame(SensorData.GetLeftImageSource(), SensorData.GetLeftBuffer());
221                         }
222 
223                         SensorData.GetRigthImageSource().AddDirtyRect(new System.Windows.Int32Rect
    (0, 0, ADNS2610_Packet.FrameWidth, ADNS2610_Packet.FrameHeight));
224                         SensorData.GetLeftImageSource().AddDirtyRect(new System.Windows.Int32Rect(0,
     0, ADNS2610_Packet.FrameWidth, ADNS2610_Packet.FrameHeight));
225                     }
226                     finally
227                     {
228                         SensorData.GetRigthImageSource().Unlock();
229                         SensorData.GetLeftImageSource().Unlock();
230                     }
231                 }));
232             }
```

```
233         }
234         /// <summary>
235         /// This task does the filtering processing of the optical flow values received
236         /// </summary>
237         /// <returns></returns>
238         private async Task OFFilteringTaskMethod()
239         {
240             while (!exitAppToken.IsCancellationRequested)
241             {
242                 if(IsOFEnable)
243                     SensorData.FilteringOF(0.1);
244                 await Task.Delay(10);
245             }
246         }
247         #endregion
248 
249         #region Delegates
250         /// <summary>
251         /// This function is called when there are data in the COM port created by the platform in
    the PC
252         /// </summary>
253         /// <param name="sender">Who is the caller of the event</param>
254         /// <param name="e">Some information about the data received</param>
255         private void OnCOMDataReceived(object sender, SerialDataReceivedEventArgs e)
256         {
257             int n = COM.BytesToRead;
258             byte[] temp = new byte[n];
259 
260             n = COM.Read(temp, 0, n);
261 
262             adns2610Provider.AddBytes(temp, 0, n);
263         }
264         /// <summary>
265         /// This event is called when a dataset has been detected by the frame wrapper
266         /// </summary>
267         /// <param name="payload">The dataset detected</param>
268         private void ImageSource_FrameAvailableEvent(ADNS2610_Packet payload)
269         {
270             SensorData.Copy(payload);
271 
272             FrameUpdateSem.Release();
273         }
274         /// <summary>
275         /// It is executed when is detected a change in the COM box at the GUI
276         /// </summary>
277         private void OnCOMSelectedChanged()
278         {
279             if(!StartIsEnable) StartIsEnable = true;
280         }
281         /// <summary>
282         /// Manage the start command
283         /// </summary>
284         /// <param name="sender">Information about the event caller</param>
285         private void OnStartCmd(object sender)
286         {
287             if ((COM == null))
288             {
289                 COM = new SerialPort(COM_sel, 921600, Parity.None, 8, StopBits.One)
290                 {
291                     ReceivedBytesThreshold = ADNS2610_Packet.PacketSize
292                 };
293 
294                 commander = new SerialSTM32GimbalCommander(COM);
295             }
296             if (!COM.IsOpen)
```

```
297             {
298                 COM.Open();
299                 COM.DiscardInBuffer();
300                 COM.DataReceived += OnCOMDataReceived;
301                 IsStarted = true;
302             }
303         }
304         /// <summary>
305         /// Manage the stop command
306         /// </summary>
307         /// <param name="sender">Information about the event caller</param>
308         private void OnStopCmd(object sender)
309         {
310             if ((COM != null) && COM.IsOpen)
311             {
312                 COM.Close();
313                 COM.DataReceived -= OnCOMDataReceived;
314                 IsStarted = false;
315                 IsOFEnable = false;
316                 IsTrackingEnable = false;
317             }
318         }
319         /// <summary>
320         /// Manage the enable optical flow command
321         /// </summary>
322         /// <param name="sender">Information about the event caller</param>
323         private void OnOFEnable(object sender)
324         {
325             IsOFEnable ^= true;
326         }
327         /// <summary>
328         /// Manage the calibration function mode command
329         /// </summary>
330         /// <param name="sender">Information about the event caller</param>
331         private void OnCalibration(object sender)
332         {
333             IsCalibrationMode ^= true;
334             SensorData.SetCalibrationMode(IsCalibrationMode);
335         }
336         /// <summary>
337         /// Manage the tracking function mode command
338         /// </summary>
339         /// <param name="sender">Information about the event caller</param>
340         private void OnTrackingEnable(object sender)
341         {
342             IsTrackingEnable ^= true;
343 
344             if (IsTrackingEnable) commander.SendCommand(CommandType.TRACKING_ON);
345             else commander.SendCommand(CommandType.TRACKING_OFF);
346         }
347         #region Gimbal Delegates
348         /// <summary>
349         /// Manage the UP command to move the platform
350         /// </summary>
351         /// <param name="sender">Information about the event caller</param>
352         private void OnUpGimbalCmd(object sender)
353         {
354             Debug.WriteLine("OnUpGimbalCmd");
355             commander.SendCommand(CommandType.UP);
356         }
357         /// <summary>
358         /// Manage the DOWN command to move the platform
359         /// </summary>
360         /// <param name="sender">Information about the event caller</param>
361         private void OnDownGimbalCmd(object sender)
```

```
362         {
363             Debug.WriteLine("OnDownGimbalCmd");
364             commander.SendCommand(CommandType.DOWN);
365         }
366         /// <summary>
367         /// Manage the LEFT command to move the platform
368         /// </summary>
369         /// <param name="sender">Information about the event caller</param>
370         private void OnLeftGimbalCmd(object sender)
371         {
372             Debug.WriteLine("OnLeftGimbalCmd");
373             commander.SendCommand(CommandType.LEFT);
374         }
375         /// <summary>
376         /// Manage the RIGHT command to move the platform
377         /// </summary>
378         /// <param name="sender">Information about the event caller</param>
379         private void OnRightGimbalCmd(object sender)
380         {
381             Debug.WriteLine("OnRightGimbalCmd");
382             commander.SendCommand(CommandType.RIGHT);
383         }
384         /// <summary>
385         /// Manage the ROTATE LEFT command to move the platform
386         /// </summary>
387         /// <param name="sender">Information about the event caller</param>
388         private void OnRotateLeftGimbalCmd(object sender)
389         {
390             Debug.WriteLine("OnRotateLeftGimbalCmd");
391             commander.SendCommand(CommandType.ROTATE_LEFT);
392         }
393         /// <summary>
394         /// Manage the ROTATE RIGHT command to move the platform
395         /// </summary>
396         /// <param name="sender">Information about the event caller</param>
397         private void OnRotateRightGimbalCmd(object sender)
398         {
399             Debug.WriteLine("OnRotateRightGimbalCmd");
400             commander.SendCommand(CommandType.ROTATE_RIGHT);
401         }
402         /// <summary>
403         /// Manage the CENTER command to move the platform
404         /// </summary>
405         /// <param name="sender">Information about the event caller</param>
406         private void OnCenterGimbalCmd(object sender)
407         {
408             Debug.WriteLine("OnCenterGimbalCmd");
409             commander.SendCommand(CommandType.CENTER);
410         }
411         #endregion
412         /// <summary>
413         /// Manage the EXIT command
414         /// </summary>
415         /// <param name="obj">Information about the event caller</param>
416         private void OnExit(object obj)
417         {
418             adns2610Provider.Stop();
419             Debug.WriteLine("Image source stopped.");
420             COM?.Close();
421             Debug.WriteLine("COM closed.");
422         }
423         #endregion
424
425         #region Auxiliar methods
426         /// <summary>
```

```
427         /// Take the pixels values of the frame and convert it to create a WriteableBitmap class
428         /// </summary>
429         /// <param name="bitmap">The WriteableBitmap class where is going to be stored the pixel
    values</param>
430         /// <param name="array">The pixel values</param>
431         private void FillFrame(WriteableBitmap bitmap, byte[] array)
432         {
433             int bitmapIdx = 0, rowIdx = ADNS2610_Packet.FrameHeight - 1, columnIdx = 0, arrayIdx;
434             int stride = bitmap.BackBufferStride;
435             int arraySize = (int) (stride * bitmap.Height);
436             IntPtr bitmapPtr = bitmap.BackBuffer;
437 
438 
439             while (bitmapIdx < arraySize)
440             {
441                 if ((bitmapIdx % stride) == 0 && (bitmapIdx > 0))
442                 {
443                     columnIdx = 0;
444                     rowIdx--;
445                 }
446 
447                 if(columnIdx < bitmap.PixelWidth)
448                 {
449                     unsafe
450                     {
451                         arrayIdx = (columnIdx * ADNS2610_Packet.FrameHeight) + rowIdx;
452                         *((byte*)bitmapPtr) = (byte)Math.Round((array[arrayIdx] & 0x3F) * (255.0 /
    63.0), 0);
453                     }
454                     columnIdx++;
455                 }
456                 bitmapIdx++;
457                 bitmapPtr += 1;
458             }
459         }
460         /// <summary>
461         /// Assign the function-event relationship
462         /// </summary>
463         private void AssignDelegateCmd()
464         {
465             StartCmd = new DelegateCommand(OnStartCmd);
466             StopCmd = new DelegateCommand(OnStopCmd);
467 
468             OFEnableCmd = new DelegateCommand(OnOFEnable);
469             CalibrationCmd = new DelegateCommand(OnCalibration);
470             TrackingEnableCmd = new DelegateCommand(OnTrackingEnable);
471 
472             UpGimbalCmd = new DelegateCommand(OnUpGimbalCmd);
473             DownGimbalCmd = new DelegateCommand(OnDownGimbalCmd);
474             LeftGimbalCmd = new DelegateCommand(OnLeftGimbalCmd);
475             RightGimbalCmd = new DelegateCommand(OnRightGimbalCmd);
476             RotateLeftGimbalCmd = new DelegateCommand(OnRotateLeftGimbalCmd);
477             RotateRightGimbalCmd = new DelegateCommand(OnRotateRightGimbalCmd);
478             CenterGimbalCmd = new DelegateCommand(OnCenterGimbalCmd);
479 
480             ExitCommand = new DelegateCommand(OnExit);
481         }
482         #endregion
483     }
484 }
```

Anexo I

‘Librería para recibir datos desde la plataforma en la aplicación ‘Eyes OF Gimbal Platform’

```
using System;
using System.Diagnostics;
using System.Threading;
using System.Threading.Tasks;

namespace FrameWrapper
{
    public abstract class SerialFrameWrapperBase<T>
    {
        protected CircularBuffer<byte> _buffer;
        protected uint _header;
        protected int _frameLength;
        protected byte[] _internalFrameAnalyzerBuffer;

        protected Task AnalyzerTask;
        protected Semaphore LevelTriggeredSem;
        protected bool AnalyzerTaskFlag;

        public delegate void FrameAvailableDelegate(T payload);
        public event FrameAvailableDelegate FrameAvailableEvent;
        /// <summary>
        /// Initialize and setting up the wrapper
        /// </summary>
        /// <param name="bufferCapacity">The buffer size used to store the data which are going to
    be processed</param>
        /// <param name="header">The character which is the header of a dataset</param>
        /// <param name="frameLength">The length of the dataset</param>
        /// <param name="eventEnable">Flag to enable or disable the event throwing</param>
        protected abstract void Init(int bufferCapacity, uint header, int frameLength, bool
    eventEnable);
        /// <summary>
        /// Starting the wrapper operation
        /// </summary>
        public void Start()
        {
            AnalyzerTask = Task.Run(() => AnalyzerLoop());
        }
        /// <summary>
        /// Stopping the wrapper operation
```

```
38          /// </summary>
39          public void Stop()
40          {
41              AnalyzerTaskFlag = false;
42              LevelTriggeredSem.Release();
43              LevelTriggeredSem.Close();
44              LevelTriggeredSem.Dispose();
45              AnalyzerTask.Wait();
46          }
47          /// <summary>
48          /// Add data to the wrapper for processing
49          /// </summary>
50          /// <param name="buffer">Buffer where the data is allocated</param>
51          /// <param name="startIndex">Index in the input buffer where is the data we are interested</
        param>
52          /// <param name="length">Number of bytes to be imported</param>
53          public void AddBytes(byte[] buffer, int startIndex, int length)
54          {
55              if(AnalyzerTaskFlag)
56                  _buffer.Add(buffer, startIndex, length);
57          }
58          /// <summary>
59          /// Decode a dataset detected. This function is defined at each particular case
60          /// </summary>
61          /// <param name="buffer">Buffer where the data is allocated</param>
62          /// <param name="length">Number of bytes to be processed</param>
63          public abstract void DecodeFrame(byte[] buffer, int length);
64          /// <summary>
65          /// Launch a event when a number of bytes in the internal buffer has reached
66          /// </summary>
67          public void OnThresholdEvent()
68          {
69              LevelTriggeredSem.Release();
70          }
71          /// <summary>
72          /// Analyze the available data in internal buffer continuously
73          /// </summary>
74          public void AnalyzerLoop()
75          {
76              int idxFrame = 0, idxTemp;
77              int length;
78              byte[] internalBuffer = new byte[_frameLength * 10];
79
80              byte[] headerBytes = BitConverter.GetBytes(_header);
81              int headerIdx = 0;
82
83              while (AnalyzerTaskFlag)
84              {
85                  if (_buffer.IsEventEnable()) try{ LevelTriggeredSem.WaitOne(); } catch{ }
86
87                  if (!AnalyzerTaskFlag) break;
88
89                  length = _buffer.Get(internalBuffer, 0, _frameLength * 3);
90                  idxTemp = 0;
91
92                  while (length > idxTemp)
93                  {
94                      if (!AnalyzerTaskFlag) break;
95
96                      _internalFrameAnalyzerBuffer[idxFrame] = internalBuffer[idxTemp];
97
98                      if(_internalFrameAnalyzerBuffer[idxFrame] == headerBytes[headerIdx])
99                      {
100                         if(idxFrame == _frameLength - 1)
101                         {
```

```
102                             DecodeFrame(_internalFrameAnalyzerBuffer, _frameLength);
103                         }
104
105                         if(headerIdx == 3)
106                         {
107                             idxFrame = -1;
108                             headerIdx = 0;
109                         }
110                         else
111                         {
112                             headerIdx++;
113                         }
114                     }
115                     else
116                     {
117                         if(headerIdx > 0) headerIdx = 0;
118                     }
119
120                     idxTemp++;
121                     idxFrame++;
122                     if(idxFrame >= _internalFrameAnalyzerBuffer.Length)
123                     {
124                         idxFrame = _frameLength-1;
125                     }
126                 }
127
128                 if (!_buffer.IsEventEnable()) Thread.Sleep(10);
129             }
130         }
131         /// <summary>
132         /// Throw a event to indicate that a dataset has been detected
133         /// </summary>
134         /// <param name="payload">The dataset detected</param>
135         public void FireEvent(T payload)
136         {
137             FrameAvailableEvent?.Invoke(payload);
138         }
139     }
140 }
```

Anexo J

Modelo de datos para el módulo EyeOF

```
using System;
using System.Collections.Generic;
using System.Text;
using OFGimbalPlatform.Base;
using FrameWrapper;
using System.Windows.Controls;
using System.Windows.Media.Imaging;
using System.Windows.Media;
using static OF_gimbal_platform.Tools.opticalFlowTool;

namespace OFGimbalPlatform.Model
{
    public class BindableOF2DandRotation : BindableBase
    {
        private int _X;
        public int X
        {
            get { return _X; }
            set { SetProperty(ref _X, value); }
        }

        private int _Y;
        public int Y
        {
            get { return _Y; }
            set { SetProperty(ref _Y, value); }
        }

        private int _theta;
        public int Theta
        {
            get { return _theta; }
            set { SetProperty(ref _theta, value); }
        }

        public BindableOF2DandRotation()
        {
            X = Y = Theta = 0;
        }

        public void Copy(OF2DandRotation data)
        {
```

```
43              X = data.X;
44              Y = data.Y;
45              Theta = data.theta;
46          }
47      }
48      public class BindableOF2D : BindableBase
49      {
50          private int _X;
51          public int X
52          {
53              get { return _X; }
54              set { SetProperty(ref _X, value); }
55          }
56
57          private int _Y;
58          public int Y
59          {
60              get { return _Y; }
61              set { SetProperty(ref _Y, value); }
62          }
63
64          public BindableOF2D()
65          {
66              X = Y = 0;
67          }
68
69          public void Copy(OF2D data)
70          {
71              X = data.X;
72              Y = data.Y;
73          }
74      }
75      public class BindableOF2DDouble : BindableBase
76      {
77          private double _X;
78          public double X
79          {
80              get { return _X; }
81              set { SetProperty(ref _X, value); }
82          }
83
84          private double _Y;
85          public double Y
86          {
87              get { return _Y; }
88              set { SetProperty(ref _Y, value); }
89          }
90
91          public BindableOF2DDouble()
92          {
93              X = Y = 0;
94          }
95
96          public void Copy(OF2D data)
97          {
98              X = data.X;
99              Y = data.Y;
100         }
101     }
102
103     public class BindableSensorData : BindableBase
104     {
105         private BindableOF2D _OFRight;
106         public BindableOF2D OFRight
107         {
```

```
108             get { return _OFRight; }
109             set { SetProperty(ref _OFRight, value); }
110         }
111         private BindableOF2D _OFLeft;
112         public BindableOF2D OFLeft
113         {
114             get { return _OFLeft; }
115             set { SetProperty(ref _OFLeft, value); }
116         }
117         private BindableOF2DandRotation _OFFussed;
118         public BindableOF2DandRotation OFFussed
119         {
120             get { return _OFFussed; }
121             set { SetProperty(ref _OFFussed, value); }
122         }
123         private BindableOF2DDouble _OFRightFiltered;
124         public BindableOF2DDouble OFRightFiltered
125         {
126             get { return _OFRightFiltered; }
127             set { SetProperty(ref _OFRightFiltered, value); }
128         }
129         private BindableOF2DDouble _OFLeftFiltered;
130         public BindableOF2DDouble OFLeftFiltered
131         {
132             get { return _OFLeftFiltered; }
133             set { SetProperty(ref _OFLeftFiltered, value); }
134         }
135         private int _SEQ;
136         public int SEQ
137         {
138             get { return _SEQ; }
139             set { SetProperty(ref _SEQ, value); }
140         }
141         private Image _imageRight;
142         public Image ImageRight
143         {
144             get { return _imageRight; }
145             set { SetProperty(ref _imageRight, value); }
146         }
147         private Image _imageLeft;
148         public Image ImageLeft
149         {
150             get { return _imageLeft; }
151             set { SetProperty(ref _imageLeft, value); }
152         }
153         private readonly WriteableBitmap ImageSourceRight;
154         private readonly WriteableBitmap ImageSourceLeft;
155         private readonly byte[] FrameRight;
156         private readonly byte[] FrameLeft;
157         private byte[] FrameRightRef;
158         private byte[] FrameLeftRef;
159         private readonly object locker;
160         private bool CalibrationModeEnabled;
161
162         #region Public methods
163         public BindableSensorData()
164         {
165             locker = new object();
166
167             OFRight = new BindableOF2D();
168             OFLeft = new BindableOF2D();
169             OFFussed = new BindableOF2DandRotation();
170
171             OFRightFiltered = new BindableOF2DDouble();
172             OFLeftFiltered = new BindableOF2DDouble();
```

```
173
174             FrameRight = new byte[ADNS2610_Packet.FrameSize];
175             FrameLeft = new byte[ADNS2610_Packet.FrameSize];
176             FrameRightRef = new byte[ADNS2610_Packet.FrameSize];
177             FrameLeftRef = new byte[ADNS2610_Packet.FrameSize];
178
179             ImageSourceRight = new WriteableBitmap(ADNS2610_Packet.FrameWidth, ADNS2610_Packet.
    FrameHeight, 96d, 96d, PixelFormats.Gray8, null);
180             ImageRight = new Image
181             {
182                 Source = ImageSourceRight
183             };
184
185             ImageSourceLeft = new WriteableBitmap(ADNS2610_Packet.FrameWidth, ADNS2610_Packet.
    FrameHeight, 96d, 96d, PixelFormats.Gray8, null);
186             ImageLeft = new Image
187             {
188                 Source = ImageSourceLeft
189             };
190
191             CalibrationModeEnabled = false;
192         }
193         public byte[] GetRightBuffer()
194         {
195             return FrameRight;
196         }
197         public byte[] GetLeftBuffer()
198         {
199             return FrameLeft;
200         }
201         public WriteableBitmap GetRigthImageSource()
202         {
203             return ImageSourceRight;
204         }
205         public WriteableBitmap GetLeftImageSource()
206         {
207             return ImageSourceLeft;
208         }
209         public void Copy(ADNS2610_Packet data)
210         {
211             if (data == null) return;
212
213             SEQ = data.SEQ;
214
215             lock (locker)
216             {
217                 if (!CalibrationModeEnabled)
218                 {
219                     OFRight.Copy(data.OFRight);
220                     OFLeft.Copy(data.OFLeft);
221                     OFFussed.Copy(data.OFFused);
222                 }
223                 else
224                 {
225                     ComputeOFInCalibrationMode(data.FrameRight, data.FrameLeft);
226                 }
227
228                 Array.Copy(data.FrameRight, 0, FrameRight, 0, ADNS2610_Packet.FrameSize);
229                 Array.Copy(data.FrameLeft, 0, FrameLeft, 0, ADNS2610_Packet.FrameSize);
230             }
231         }
232         public void FilteringOF(double alpha)
233         {
234             lock(locker){
235                 OFLeftFiltered.X = OFLeftFiltered.X + alpha * (OFLeft.X - OFLeftFiltered.X);
```

```
236                 OFLeftFiltered.Y = OFLeftFiltered.Y + alpha * (OFLeft.Y - OFLeftFiltered.Y);
237
238                 OFRightFiltered.X = OFRightFiltered.X + alpha * (OFRight.X - OFRightFiltered.X);
239                 OFRightFiltered.Y = OFRightFiltered.Y + alpha * (OFRight.Y - OFRightFiltered.Y);
240             }
241         }
242         public void SetCalibrationMode(bool state)
243         {
244             lock (locker)
245             {
246                 CalibrationModeEnabled = state;
247
248                 Array.Copy(FrameRight, 0, FrameRightRef, 0, ADNS2610_Packet.FrameSize);
249                 Array.Copy(FrameLeft, 0, FrameLeftRef, 0, ADNS2610_Packet.FrameSize);
250             }
251         }
252         #endregion
253         #region Auxiliar methods
254         private void ComputeOFInCalibrationMode(byte[] frameRight, byte[] frameLeft)
255         {
256             ComputeOFTool(frameRight, frameLeft, FrameRightRef, FrameLeftRef, OFRight, OFLeft,
    OFFussed);
257         }
258         #endregion
259     }
260 }
```