UNIVERSIDAD POLITÉCNICA DE MADRID

E.T.S. de Ingeniería de Sistemas Informáticos

Grado en Ingeniería de Computadores

Sistema de Visión en Primera Persona

Proyecto de Fin de Grado

Autor

Adrián Rodríguez García

Tutor

Vicente A. García Alcántara

Julio 2017

Índice

Índice de figuras

Índice de tablas

Objetivo

El objetivo del presente proyecto es desarrollar un sistema de visión en primera persona capaz de captar la información visual relativa al punto de vista de un sistema no tripulado controlado de forma remota y enviarla al operador del citado sistema para ofrecer un mayor control y precisión en el manejo. El sistema se basará en tecnologías y componentes de coste reducido de cara a que el gasto de desarrollo de la solución sea lo menor posible y, a poder ser, se optará por tecnologías y lenguajes distribuidos bajo licencias libres.

Se persigue analizar las distintas opciones disponibles en el mercado para elegir los componentes que mejor se adapten a la filosofía DIY (*Do It Yourself*) y conseguir así que el sistema pueda ser reproducido por alguien que no posea conocimientos avanzados de informática y electrónica.

A nivel personal se pretende afianzar los conocimientos adquiridos a lo largo de la carrera y seguir algunas de las metodologías estudiadas para desarrollar el proyecto. También se cuenta como un objetivo ganar experiencia en el análisis y estudio de *datasheets*, ya que son necesarios a la hora de entender el funcionamiento y los requisitos para trabajar con el hardware descrito en ellos.

Resumen

Con la proliferación de sistemas de realidad virtual y drones enfocados al ocio se han empezado a popularizar los sistemas de visión en primera persona. Estos sistemas suelen estar constituidos por componentes de un coste bastante elevado, por lo que se ha intentado enfocar las diferentes partes del prototipo de forma que fuesen de un coste asequible para el público general y que el sistema fuera del tipo DIY (*Do It Yourself* por sus siglas en inglés) para que cualquier persona con unos conocimientos básicos/medios de tecnología pudiera reproducirlo sin demasiado esfuerzo.

Se ha intentado construir el prototipo con varias configuraciones hardware y componentes distintos optándose finalmente por una Raspberry Pi como controlador para el sistema ya que ha resultado ser la mejor opción en cuanto a prestaciones con respecto a su ajustado precio. El software para el controlador de las distintas versiones se ha realizado en lenguaje C ya que permite realizar operaciones a bajo nivel con la sintaxis de un lenguaje de alto nivel por lo que el código desarrollado es lo suficientemente eficiente para no tener que recurrir a instrucciones en ensamblador. Para el software receptor ejecutado en un ordenador portátil se ha optado por desarrollarlo en Java, ya que se trata de un lenguaje licenciado en su mayor parte por una Licencia Pública General de GNU por lo que es libre parcialmente y además se trata de un lenguaje portable por lo que el software podría ejecutarse en cualquier ordenador que tenga una Máquina Virtual de Java.

Abstract

With the spread of Virtual Reality systems and leisure time focused drones, have begun to popularize First Person View systems (FPV). These systems are usually made up of a fairly high cost components, so that it has been attempted to focus the diferent parts of the system in order that they were affordable for the general public and that the system was of a DIY (*Do It Yourself*) type so that anyone with a basic-to-mid technology knowledge could reproduce it without too much effort.

It has been attempted to build up the prototype with many hardware configurations and different components, finally opting for using a Raspberry Pi as the controller of the system as it has turned out to be the best option in terms of benefits with respect to its low price. The controller software for the diferent versions has been made in C language as it allows to do low level operations with the syntax of a high level language so that the delveloped code is efficient enought to not have to resort to assembler instructions. For the receiver software executed in a laptop it has been chosen to develop in Java since it's an almost all GNU General Public License licensed language so that is partially free and also it's a portable language so the software could be executed in any computer that has a Java Virtual Machine installed.

Capítulo 1: Introducción

Ya sea como parte de un sistema de Visión Artificial o para manejar un vehículo controlado remotamente (RCV), los sistemas de Visión en Primera Persona (FPV por sus siglas en inglés) [Rce,www] son una parte esencial para su control. Los sistemas FPV suelen consistir en una cámara montada sobre el vehículo que se desea manejar de forma remota, una conexión para enviar las imágenes en tiempo real que está captando la cámara y un dispositivo en el que reproducir las imágenes. Para la persona que está manejando el vehículo, el punto de vista es idéntico a si estuviera a bordo del mismo. Este tipo de sistemas tienen multitud de aplicaciones, desde un pequeño coche R/C o un cuadricóptero de juguete hasta los robots que utilizan las fuerzas de seguridad para desactivación de explosivos.

Fig. 1: Coche R/C con cámara a bordo [Gee,www].

Fig. 2: Robot artificiero de la Guardia Civil [Lao,www].

La diferencia entre los sistemas destinados al ocio y los sistemas ideados para un uso crítico radica en la calidad del vídeo (menos resolución en los sistemas de ocio) y en la velocidad de la conexión de envío de datos (más lenta en equipos destinados al ocio).

En este tipo de sistemas, la conexión para el envío de datos de la cámara puede ser a través de cable o sin cables, en este segundo caso las opciones de comunicación son por radio, WiFi o Bluetooth.

1.1 Componentes

Dependiendo de la complejidad y del propósito del sistema de visión, los componentes de éste variaran en numero, calidad y precio. En este apartado se tratará de dar una visión general de los componentes básicos necesarios para implementar un sistema FPV.

- **Cámara**: Una cámara CMOS del tipo que se usa para seguridad puede servir si no es necesaria demasiada calidad de imagen. Es importante que el voltaje al que se alimenta la cámara sea compatible con la alimentación del vehículo sobre el que se monta el sistema.
- **Transmisor de vídeo**: El circuito emisor de vídeo coge la señal de la cámara y la envía a través de una antena. Este tipo de circuitos trabajan en frecuencias de envío de información que superan 1GHz. También puede usarse tecnología WiFi o Bluetooth.
- **Receptor de vídeo**: Un circuito receptor que trabaje en la misma frecuencia o mismo protocolo que el emisor.

- **Display:** Se puede usar cualquier tipo de monitor o, para una experiencia más inmersiva, unas gafas de realidad virtual [Vrs,www].

Fig. 3: Diagrama de componentes básicos [Eur,www].

1.2 Estado del arte

Con la proliferación de drones radiocontrol enfocados al ocio se ha popularizado mucho el uso de sistemas de visión en primera persona para controlarlos. En algunas asociaciones de pilotos de drones, como la Drone Sport Association americana, es obligatorio el uso de sistemas FPV para pilotar los aparatos [Dro,www]. Este tipo de drones utilizan cámaras de tamaño muy reducido, muchas de ellas consistiendo solamente en la placa de circuito y el sensor, para reducir el peso en el dron y el coste de la cámara. Suelen montar una única cámara por lo que no es posible tener visión en 3D.

Fig. 4: Dron personalizado de competición [Fpv,www].

Por otra parte, los robots de uso militar/policial como el Bimanual Dexterous Robotic Platform (BDRP, también llamado Sally) [Giz,www] desarrollado por el Departamento de Defensa estadounidense [Def,www], montan sistemas de visión mucho más complejos que suelen incluir varias cámaras para poder obtener visión estereoscópica, lo que ayuda al operador en la realización de tareas críticas para las que está diseñado este tipo de robots.

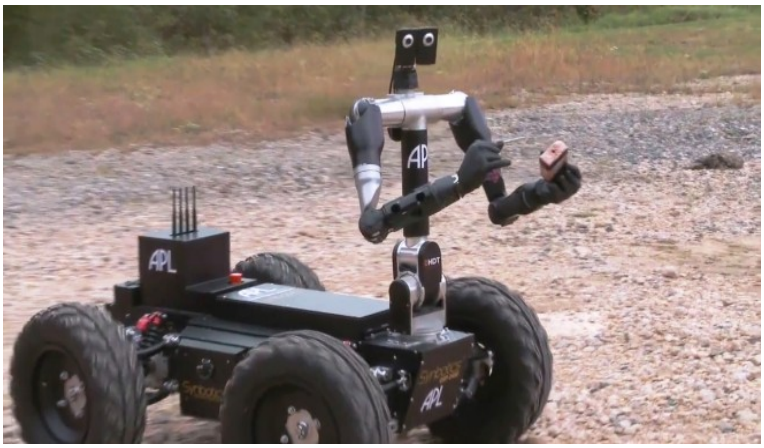


Fig. 5: Robot militar Robo Sally [Chw,www].

Capítulo 2: Análisis de requisitos y Especificaciones

Se pretende diseñar un sistema de visión en primera persona que cumpla una serie de requisitos.

- El sistema debe ser capaz de enviar la información recogida por la cámara.
- El sistema ha de poder comunicarse con la cámara y configurarla así como capturar la información que genere ésta.
- El sistema debe contar con un software capaz de mostrar la información que capte la cámara.
- El tamaño de la cámara y el circuito que la controle debe ser lo suficientemente pequeño para poder montarse en un vehículo radiocontrol o similar.
- La comunicación, a poder ser, debe realizarse de forma inalámbrica para que el sistema tenga la máxima movilidad posible.
- Los componentes elegidos para desarrollar el sistema deben ser lo más económicos posible para que el conjunto sea asequible y enfocar el conjunto a una filosofía *DIY* (*Do It Yourself* por sus siglas en inglés) [Cam,www].

Los objetivos a nivel personal del alumno se centran en su crecimiento como ingeniero así como la adquisición de conocimientos referidos al desarrollo de proyectos con diverso hardware involucrado.

- Seguir metodologías de desarrollo *top-down* a lo largo del proyecto.
- Adquirir soltura a la hora de analizar y comprender documentación y *datasheets* referentes a hardware de diferente tipo.
- Ser capaz de elegir la mejor opción para un componente teniendo en cuenta las diferentes opciones disponibles.
- Mejorar habilidades de programación y desarrollo de software controlador de dispositivos y programación de bajo nivel.

Capítulo 3: Diseño del sistema

En este capítulo se presentarán todos los aspectos relacionados con el diseño del sistema final. Como primer paso se analizarán las posibles alternativas en cuanto a hardware (cámara, controlador, conectividad). A continuación se explicará la configuración escogida para la implementación y el porqué de esas decisiones.

3.1 Análisis de alternativas

Existen multitud de posibilidades en cuanto a los componentes que pueden usarse para implementar un sistema de visión en primera persona, así como las posibles combinaciones distintas de estos componentes. Esta versatilidad es muy útil ya que dota de flexibilidad al proyecto pero tiene como contrapunto negativo que todas esas posibles opciones distintas deben ser evaluadas para encontrar la combinación que mejor se adecue a los requisitos del sistema.

3.1.1 Cámaras

Fig. 6: Cámara CMOS OV7670 [Ele,www].

Fig. 7: Cámara CCD Sony PZ0420M [Big,www].

Las cámaras usadas en estos sistemas deben ser lo más ligeras que sea posible ya que deben ir montadas en los vehículos desde los que se desee tener visión. Las cámaras CMOS, como la de la figura 6, cumplen con la premisa anterior, teniendo como ventaja su reducido precio y consumo de energía, aportando la suficiente calidad para ser una opción a tener en cuenta. Como otra opción tenemos las cámaras CCD (figura 7) que producen imágenes de más calidad pero consumen más energía en el proceso.

CCD	CMOS
Estructura simple del sensor	Celdas independientes
Necesidad de chip externo digitalizador	Digitalización en celda
Mayor consumo	Menor consumo
Mejor rango dinámico	Más barato
Menor ruido	Mayor velocidad

Tabla 1: Características de los sensores CMOS y CCD

3.1.2 Conectividad

Con respecto a la conectividad lo primero que cabe destacar es que predomina el uso de soluciones inalámbricas, ya que permiten más libertad de movimiento al vehículo. La característica a tener en cuenta a la hora de elegir un circuito emisor/receptor es la banda de frecuencia en la que trabaja, ya que dependiendo de cual sea puede tener mejor o peor penetración de la señal, puede variar el ruido en la banda debido a la popularidad de ésta o, incluso, puede que esa frecuencia esté reservada y sea ilegal usarla en un determinado país.

- 900 MHz: Ofrece gran penetración y muy buen rango, el problema es que es una banda reservada a telefonía móvil por lo que es ilegal en varios países. También tiene en contra el mayor tamaño de las antenas.
- 1,2 GHz: Buen rango y penetración aunque es una banda compartida por lo que es ilegal en varios países al igual que los 900 MHz.
- 1,3 GHz: Rango y penetración similares a 1,2 GHz y, al igual que ésta, es ilegal usarla en varios países.
- 2,4 GHz: Ofrece buen rango y los equipos son baratos pero desafortunadamente la mayoría de los equipos radiocontrol operan en esta frecuencia, así como los sistemas Bluetooth y WiFi, por lo que se experimentan problemas de ruido.
- 5,8 GHz: Buen rango si se monta correctamente, los equipos son pequeños y baratos y la banda está libre de interferencias, por lo que es la frecuencia más popular en las carreras de drones FPV.

3.1.3 Controlador

Si es necesario, se debe utilizar un controlador a modo de interfaz entre la cámara y el emisor de la señal de video. Como características principales se debe tener en cuenta el tamaño, cuanto más reducido mejor será ya que debe ir montado en el vehículo junto a la cámara, y el consumo, ya que deberá ser alimentado con una batería.

Las alternativas a este respecto son múltiples ya que existe una gran variedad de opciones a contemplar sobre todo gracias a la expansión de las plataformas Arduino [Ard4,www] y Raspberry Pi [Ras5,www] debido a su bajo coste, al buen numero de desarrolladores que tienen sus respectivas comunidades y a la gran cantidad de documentación que puede encontrarse de ambas plataformas. Otro factor a tener en cuenta es la creciente aparición de cámaras y módulos de comunicación compatibles con ambas, así como la variedad de componentes y bibliotecas desarrollados expresamente para trabajar con estos sistemas.

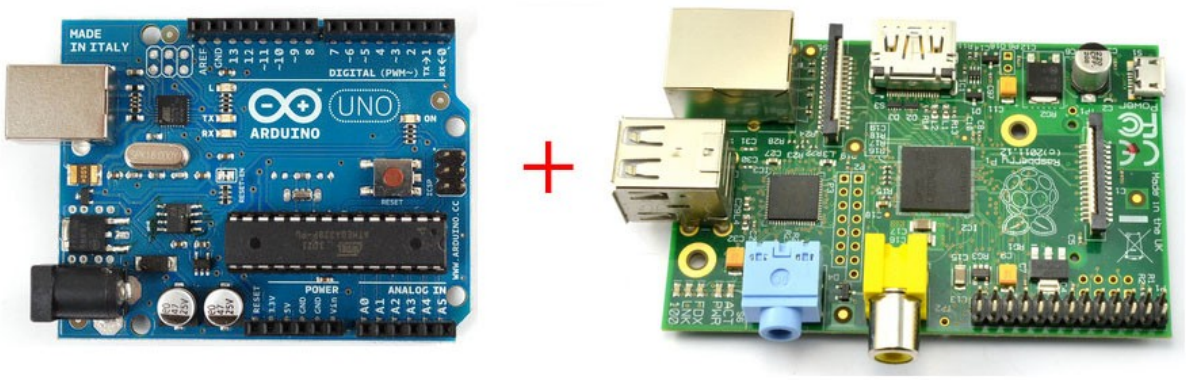


Fig. 8: Arduino UNO y Raspberry Pi 1 model B [Har,www].

También se han implementado algunos sistemas de este tipo con la familia de controladores PIC32 de Microchip [Mic10,www]. Con las familias de 8 y 16 bits es difícil conseguir resultados satisfactorios ya que no tienen la potencia suficiente para manejar una cantidad de información tan grande por segundo como la que produce una cámara capturando vídeo. Una gran ventaja que tiene utilizar microcontroladores de Microchip es que para un gran número modelos ofrecen la posibilidad de pedir muestras gratuitas de prueba, por lo que se reduce el coste si se desea hacer pruebas de rendimiento con varios chips o si por alguna cuestión es necesario cambiar el diseño. El gran inconveniente de estos microcontroladores es que es necesario montar el circuito completo para trabajar con él, es decir, lo que se ofrece es un chip como el de la figura 9 que debe ser montado en una placa con todos los componentes necesarios para su correcto funcionamiento (resistencias, condensadores, cristales de reloj, etc). Si se necesita un circuito completo montado en una placa, se puede optar por uno de los kits ya desarrollados de Microchip pero su precio es más alto que una Raspberry o Arduino ofreciendo características similares (Figura 10).

Fig. 9: Microcontrolador PIC32 [Fut,www].

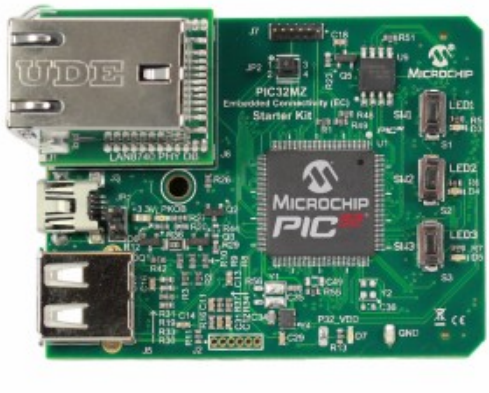


Fig. 10: PIC32MZ EC Starter Kit [Mic,www].

Como otra opción se puede usar como controlador una FPGA. Estos circuitos ofrecen la mayor versatilidad debido a su naturaleza reprogramable y a que están diseñados para ofrecer multitud de módulos distintos en el mismo dispositivo, así como mayor potencia a la hora de manejar varias tareas a la vez (por ejemplo, capturar vídeo de una cámara, almacenarlo y enviarlo a un dispositivo receptor vía emisor inalámbrico). Su principal inconveniente es su alto coste, por lo que para un proyecto con presupuesto reducido no es una opción demasiado viable.

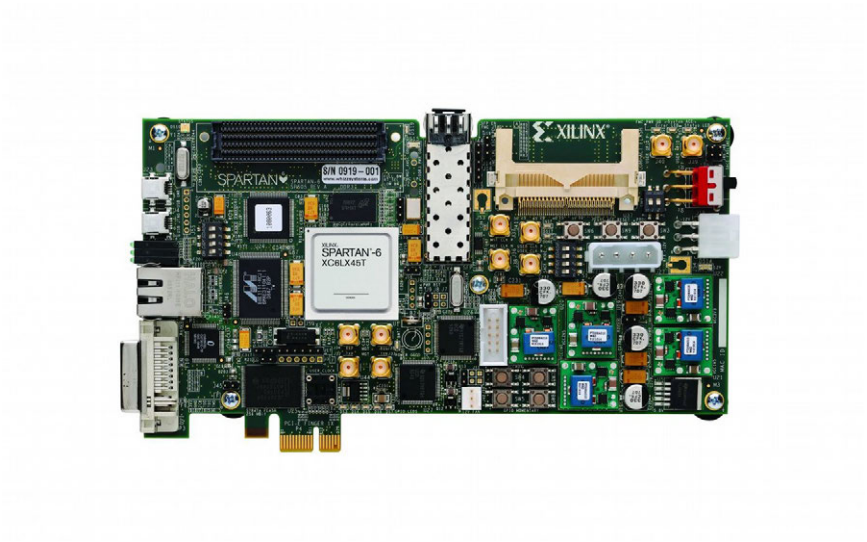


Fig. 11: Xilinx Spartan-6 FPGA Embedded Kit [Xil,www].

Capítulo 4: Hardware usado

En este apartado se presentarán los componentes hardware empleados para construir el sistema de visión, así como las características de cada uno. También se comentarán las razones de cada elección con respecto a las posibles alternativas disponibles. Por último, se indicarán las interconexiones entre los diferentes elementos.

4.1 Componentes

Gracias a la creciente popularidad de los drones destinados al ocio usados en carreras y a que éstos se manejan con sistemas de visión en primera persona debido a que facilitan su manejo, ha aumentado el número de alternativas distintas de componentes que pueden usarse para implementarlos por lo que el primer paso será un análisis de mercado y estudio de compatibilidad.

4.1.1 Cámara

El primer elemento a analizar será la cámara, ya que parte del resto de componentes dependerá de la cámara escogida y de su compatibilidad con éstos. La primera característica a tener en cuenta a la hora de elegir la cámara, como se indicó en el capítulo anterior, debe ser el tamaño, ya que cuanto más pequeña sea más fácil será montarla en el vehículo que lleve el sistema de visión. Otra característica importante a la hora de elegir la cámara es el interfaz de comunicación, ya que debe de ser compatible con los protocolos de comunicación que implemente el controlador que la maneje o, de lo contrario, será imposible configurar la cámara o recoger la información que esté captando.

En un primer momento se decidió usar el módulo OV7670 [Vot,www] de OmniVision [Ovt2,www] ya que sobre el papel sus especificaciones cumplían con las deseadas. Esta cámara es del tipo CMOS, su máxima resolución es VGA y tiene un precio que ronda los 7€. Tiene por dimensiones 3.5 cm x 3.4 cm x 2.6 cm por lo que cumple con la premisa de tener un tamaño reducido además de pesar solamente 13 gramos por lo que a priori es una buena opción. En el apartado de interfaces el módulo se comunica bajo el protocolo SCCB [Ovt,www] para recibir comandos de configuración. El protocolo SCCB es un protocolo de comunicación serie compatible con I^2C [I2c,www] desarrollado por OmniVision para controlar la mayoría de las funciones de sus cámaras. Para la salida de datos el módulo cuenta con 8 pines por los que de forma síncrona se envía la información byte a byte en paralelo de las imágenes que está recogiendo el sensor de la cámara.

Fig. 12: Cámara CMOS OV7670 [Ele,www].

El problema de esta cámara y por lo que no se continuó con ella fue la imposibilidad de configurarla correctamente, es decir, se consiguió realizar la comunicación por el protocolo SCCB con una implementación bitbang [Dna,www] de la librería I^2C siguiendo el cronograma de la figura 13. Esta técnica consiste en implementar el comportamiento de un hardware de comunicación serie a nivel software usando para ello pines de entrada/salida de propósito general del controlador. Pero aún pudiéndose cambiar los valores de los registros fue imposible hacer que la cámara capturase ninguna imagen, debido probablemente a la gran cantidad de valores distintos que se pueden dar a los registros lo que hace que las inicializaciones de la cámara sean difíciles de comprender.

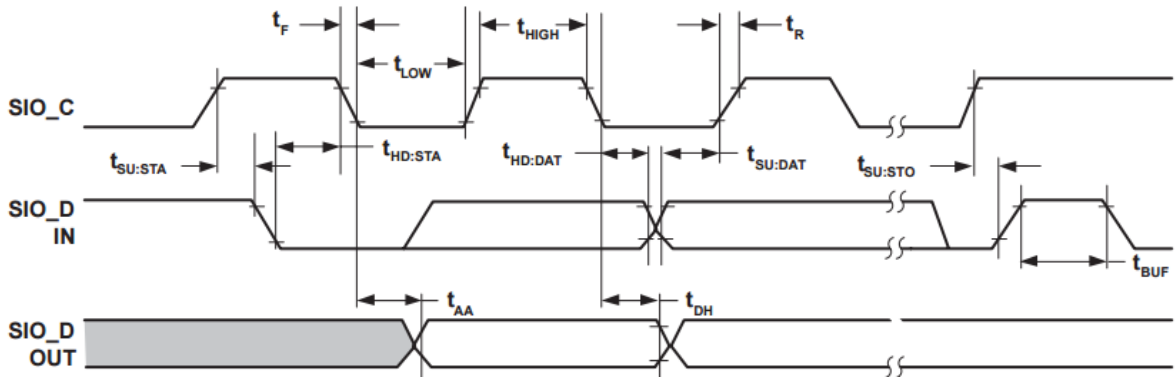


Fig. 13: Cronograma SCCB [Vot,www].

En el cronograma de la figura anterior puede observarse la forma en la que se lleva a cabo la comunicación serie, siendo necesaria una secuencia de inicio al principio del intercambio de información a la que siguen lecturas o escrituras que se llevan a cabo en los flancos altos del reloj.

Ante la imposibilidad de conseguir ningún resultado positivo, se decidió cambiar de módulo de cámara estudiándose diversas opciones para encontrar una que no presentara los problemas de configuración que tenía el modelo OV7670. Tras realizar una comparativa entre otras opciones disponibles se decidió utilizar el módulo ArduCAM Mini de 2MP [Ard3,www], esta cámara ofrece un sensor CMOS de 2 megapíxels y una conexión de datos vía SPI [Mct,www] por lo que se reduce drásticamente la complejidad del interfaz de la cámara (el módulo tiene 8 pines en comparación con los 18 del OV7670). Las dimensiones de la placa son 3.4cm x 2,4cm con un peso de 20 gramos y un precio de 26€ por lo que también es una opción asequible y con unas dimensiones buenas para montarlo en un vehículo. La mayor ventaja de esta cámara es la simplicidad de su interfaz hardware y la biblioteca de codigo abierto disponible para manejarla [Git,www]. Esta biblioteca es compatible con diversas plataformas como Arduino, Raspberry Pi, STM32 [Stm,www], Beaglebone Black [Bea,www], Chipkit [Chi,www] y, en principio, con cualquiera que tenga interfaz I^2C y SPI.

OV7670	ArduCAM Mini 2MP
Más barata	Más sencilla
Datos por pines paralelos (D0-D7)	Datos por SPI
Mayor número de pines a controlar	Más registros del sensor
Menor calidad de imagen	Más modos de captura
Existen desarrollos como ejemplo	Biblioteca de control disponible

Tabla 2: Comparativa entre OV7670 y ArduCAM Mini 2MP

Fig. 14: ArduCAM Mini 2MP [Ard,www].

4.1.2 Controlador

El controlador es empleado en este sistema para que actúe de interfaz entre la cámara y el módulo que se ocupe de la comunicación. En un primer momento se pensó usar un microcontrolador de Microchip que contase con interfaz I^2C y suficientes pines para interactuar con las salidas de datos de la cámara. Tras barajar varios modelos de distintas familias se optó por el microcontrolador PIC32MZ2048ECH064 [Mic8,www] (figura 15), uno de los chips de la familia que ofrece las características más avanzadas y mayor potencia de cálculo. El principal problema de este chip es que no existe una versión con un encapsulado compatible con el montaje en orificio pasante necesario para implementar el circuito sobre una placa de prototipado, por ello se adquirió un zócalo adaptador que permitiese sacar cables a la placa de prototipado (figura 16). El problema del zócalo es que aún siendo pequeño aumenta la complejidad y el tamaño del sistema que debe ir montado en el vehículo.

Fig. 15: PIC32MZ2018ECH064 con encapsulado TQFP [All,www].

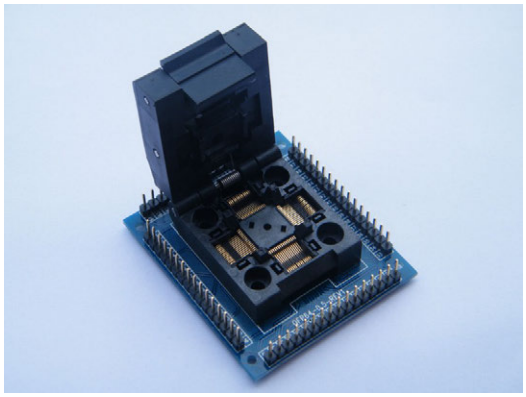


Fig. 16: Zócalo adaptador TQFP [Ali,www].

Como características principales el chip tiene una velocidad máxima de 200 MHz, 512KB de RAM, 53 pines configurables como entrada/salida, el rango de tensión de funcionamiento

está entre 2,2V y 3,6V, USART, I^2C, SPI y 8 canales DMA que ayudan a trabajar con la información que genera la cámara.

Al final se decidió cambiar de controlador por una Raspberry Pi ya que se intentó adaptar la biblioteca de ArduCAM [Git2,www] al PIC32 y fue imposible porque la inicialización del sensor de la cámara no se pudo adaptar al microcontrolador, teniendo el resto de la biblioteca ya implementada para esa plataforma. La Raspberry Pi 1 Model B utilizada en este caso ofrece la funcionalidad de un ordenador en una placa de tamaño muy reducido con 512MB de memoria RAM, un procesador ARM de un solo núcleo a 700 MHz, un buen número de E/S configurables con diversas bibliotecas para poder trabajar con módulos hardware a través de múltiples interfaces como SPI, I^2C, USART, i^2S, diversos puertos USB, HDMI, CSI para cámaras, DSI para pantallas, Ethernet, jack de 3,5mm para audio y vídeo compuesto. Un factor a tener también en cuenta es su bajo precio ya que cuesta solamente 30€ por lo que junto a Arduino es de las plataformas más asequibles.

Fig. 17: Raspberry Pi 1 Model B [Ves,www].

Otra ventaja de la Raspberry Pi es su sistema operativo oficial, Raspbian [Ras6,www], que es una versión de Debian preparada para poder ser cargada en la Raspberry desde una tarjeta SD como las utilizadas en las cámaras digitales. Este sistema basado en Linux es libre y de código abierto y tiene disponibles alrededor de 35.000 paquetes de software precompilado para una fácil instalación. Como alternativas existen Ubuntu Mate [Ubu,www], Snappy Ubuntu Core [Ubu2,www], Windows 10 IoT [Mic11,www], OSMC [Osm,www], LibreELEC [Lib,www], PiNet [Pin2,www] y RISC OS [Ris,www].

4.1.3 Comunicación

Como se indicó en el capítulo anterior, la mejor opción y la más viable es la comunicación inalámbrica ya que es el tipo de comunicación que dota al vehículo de más autonomía y libertad de movimiento. Tras tener en cuenta todas las opciones anteriormente citadas a este respecto, se decidió trabajar con tecnología WiFi ya que a pesar de ser una opción poco utilizada en este tipo de sistemas ofrece ventajas como la amplia variedad de opciones en cuanto al equipamiento a usar, la posibilidad de usar ordenadores con tarjeta de red inalámbrica o smartphones como receptor, el hecho de ser una tecnología estándar y el

bajo coste de los componentes. En un primer momento se optó por usar el módulo RN171XV [Mic4,www] que ofrece conectividad WiFi bajo el estándar 802.11 b/g.

El módulo RN171XV se basa en el RN171 de Roving Networks e incorpora una radio 802.11 b/g, un procesador SPARC de 32 bits, una pila TCP/IP, un reloj en tiempo real, un acelerador criptográfico, una unidad de control de potencia y una interfaz de sensores analógicos. El módulo ofrece una UART TTL como interfaz hardware que puede soportar una tasa de datos de hasta 464Kbps, puede trabajar como cliente en una red doméstica o como AP creando su propia red dedicada, dispone de 8 pines destinados a E/S de propósito general, 3 pines destinados a entrada de datos de sensores analógicos y viene con un firmware precargado para simplificar la integración con los demás componentes, ya que para funcionar de forma básica sólo necesita cuatro conexiones (PWR, TX, RX y GND). La placa sobre la que se monta el módulo esta basada en el diseño del estándar 802.15.4 que utiliza Digi para sus placas XBee [Dig,www] por lo que aún siendo un circuito preparado para el montaje en orificio pasante se tuvo que comprar una placa de adaptación y unos sockets compatibles con ambos ya que el paso del RN171XV es más pequeño que el de la breadboard usada para montar el sistema.

Fig. 18: Módulo RN171XV [Rso,www].

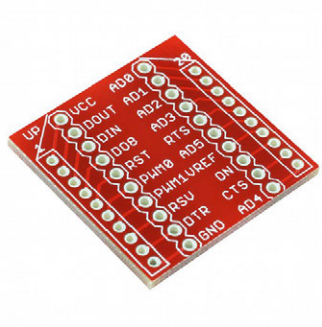


Fig. 19: XBee Breakout Board [Spa,www].

El módulo funcionaba correctamente y se consiguió trabajar a una velocidad cercana a 1Mbaud, pero al cambiar el controlador por una Raspberry Pi, que dispone de puertos USB en cualquiera de sus versiones, se decidió cambiar también el módulo de comunicaciones por un *dongle USB*. Este tipo de adaptadores inalámbricos tienen la gran ventaja de ocupar un espacio ínfimo y de estar preparados para usarse nada más ser conectados a un puerto USB libre del sistema al que se necesite dar conectividad (*plug and play*), por lo que se consideró mejor opción ya que limitaba el número de E/S a usar de la Raspberry, reducía la complejidad del software controlador del sistema y reducía el tamaño total del mismo. Se disponía de un dongle D-Link [Dli2,www] de tamaño muy pequeño, compatible con el sistema operativo de la Raspberry Pi y que ofrece conectividad 802.11 b/g/n, es compatible con los algoritmos de cifrado WEP, WPA y WPA2.

Fig. 20: Microadaptador Wireless USB D-Link [Dli,www].

4.1.4 Alimentación

Respecto a la alimentación del sistema, hay distintas opciones que se pueden considerar dependiendo del resto de componentes elegidos. Para la primera implementación en la que se usaba como controlador un PIC32 se decidió utilizar una pila de 9V o, si el vehículo en el que se montase después disponía de ella, la fuente de alimentación del propio vehículo. Esta versatilidad a la hora de compatibilizar el sistema con distintas opciones de alimentación se consigue utilizando un regulador de voltaje. Estos circuitos aceptan un rango mas o menos amplio de voltaje de entrada dependiendo de cada modelo y como salida proporcionan un voltaje fijo regulado. El dispositivo elegido fue el LM2937 de Texas Instruments [Tin,www] que acepta como entrada una tensión en el rango entre 4,75V como mínimo y 26V como máximo, y como voltaje de salida suministra 3,3V con un máximo de 500mA como corriente de salida. Tiene protección contra cortocircuitos, sobrecarga térmica, colocación inversa de batería y transitorio de entrada de hasta 60V.

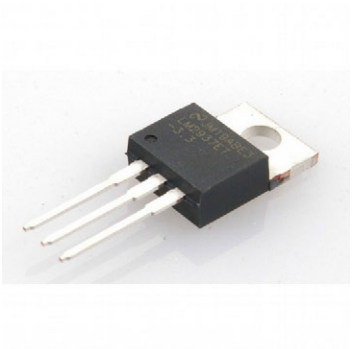


Fig. 21: Regulador LM2937 [Els,www].

Al cambiar el microcontrolador por una Raspberry Pi también fue necesario cambiar la forma en la que se alimentaba el circuito. La Raspberry se alimenta a través de un conector micro USB y necesita una tensión estable de 5V y una corriente de 700mA para funcionar. Una ventaja es que un cargador de teléfono móvil basta para alimentar la Raspberry Pi, habiendo sido lo usado para alimentarla en este caso.

4.1.5 Receptor

Como receptor de la información suministrada por la cámara se ha elegido un ordenador portátil con tarjeta inalámbrica, ya que es la opción que aúna receptor WiFi, interfaz para mostrar la información (pantalla) y herramientas para desarrollar el software que la mostrará. El portátil tiene Windows 7 como sistema operativo, aunque el software se ha desarrollado en Java por lo que sería portable a otros sistemas que dispongan de una máquina virtual de Java (JVM). Las características del ordenador portátil son las siguientes:

Fig. 22: Ordenador que actúa como receptor [Opt,www].

- Procesador Intel Core i7-3612QM
- Memoria RAM 8GB DDR3
- Disco duro 500GB SATA
- Pantalla 15,6" LED HD (1366 x 768) 16:9
- Tarjeta gráfica AMD Radeon HD 8570M 2GB GDDR3
- Conectividad
  - 802.11 b/g/n
  - LAN 10/100
  - Bluetooth V4.0 High Speed

4.2 Conexiones

Las conexiones del sistema se realizarán entre los distintos componentes con la Raspberry Pi ya que ésta realiza las tareas de controlador principal. A un puerto USB se debe conectar el dongle WiFi mientras que la cámara se conectará a los pines de entrada/salida de los que dispone la Raspberry. La cantidad de pines y la función de cada uno dependerá de la versión de la Raspberry Pi que se esté usando, en este caso se trata del primer modelo desarrollado que presenta el menor número de pines de entrada/salida de propósito general siendo en este caso suficientes para controlar la cámara.

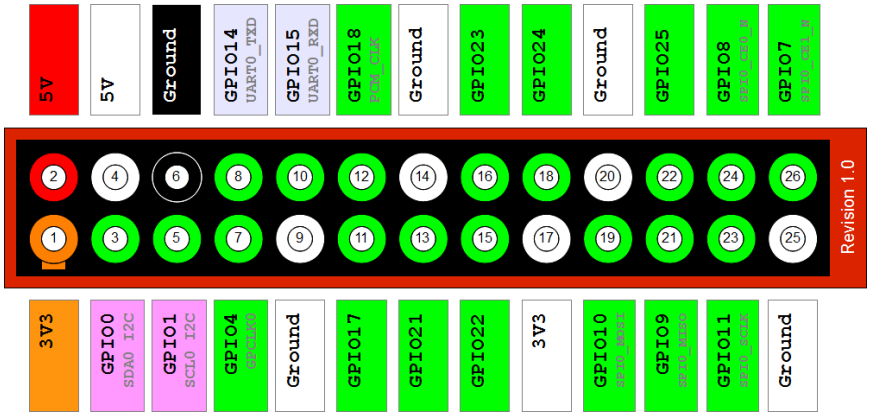


Fig. 23: Raspberry Pi 1 Model B GPIO [Ras,www].

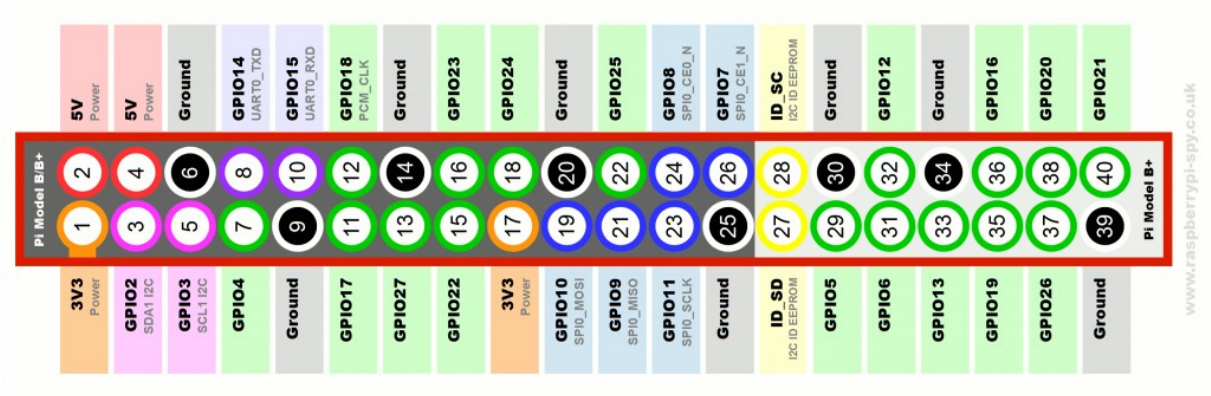


Fig. 24: Raspberry Pi 2 y 3 GPIO [Ras2,www].

Las imágenes anteriores (figura 23 y 24) muestran la numeración y función de los distintos pines en cada modelo de Raspberry Pi desarrollado. Como puede observarse, en los modelos actuales se amplió el número de pines respetándose la funcionalidad de los 26 primeros por compatibilidad con desarrollos en placas anteriores.

A la hora de trabajar con las entradas/salidas en la Raspberry es común utilizar una biblioteca para hacer más comprensible el uso de las GPIO y poder manejar de forma

cómoda las diferentes funcionalidades que son capaces de realizar algunos pines (SPI, UART,...).

En este caso se ha usado para manejar los pines entrada/salida la biblioteca WiringPi [Wir,www], la cual es un intento de portar la simplicidad de las E/S en Arduino. Su objetivo es tener una única plataforma y un conjunto de funciones para acceder a las GPIO de la Raspberry desde diferentes lenguajes. WiringPi es una biblioteca escrita en C pero está disponible para usuarios de Python [Pyt,www] y Ruby [Rub,www].

Se ha optado por utilizar esta plataforma al ser necesaria para trabajar con la biblioteca de libre distribución disponible para manejar el módulo de ArduCAM. Es necesario ya que esta biblioteca hace uso de funciones de la plataforma WiringPi para manejar los diferentes pines de la cámara así como utilizar las diferentes funcionalidades necesarias para el funcionamiento de la cámara como son el protocolo SPI y el I^2C. La figura 25 muestra la equivalencia entre la nomenclatura de los pines utilizada por la plataforma WiringPi y la numeración original de Raspberry Pi.

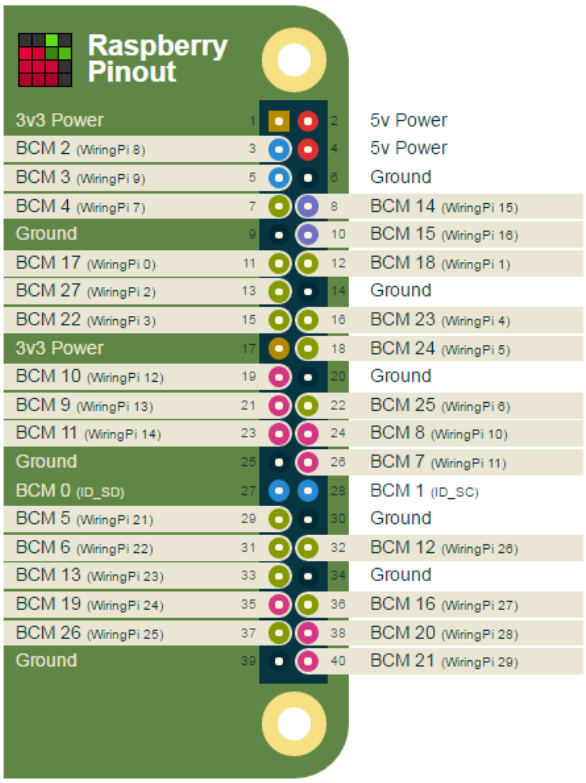


Fig. 25: Diagrama de pines WiringPi [Pin,www].

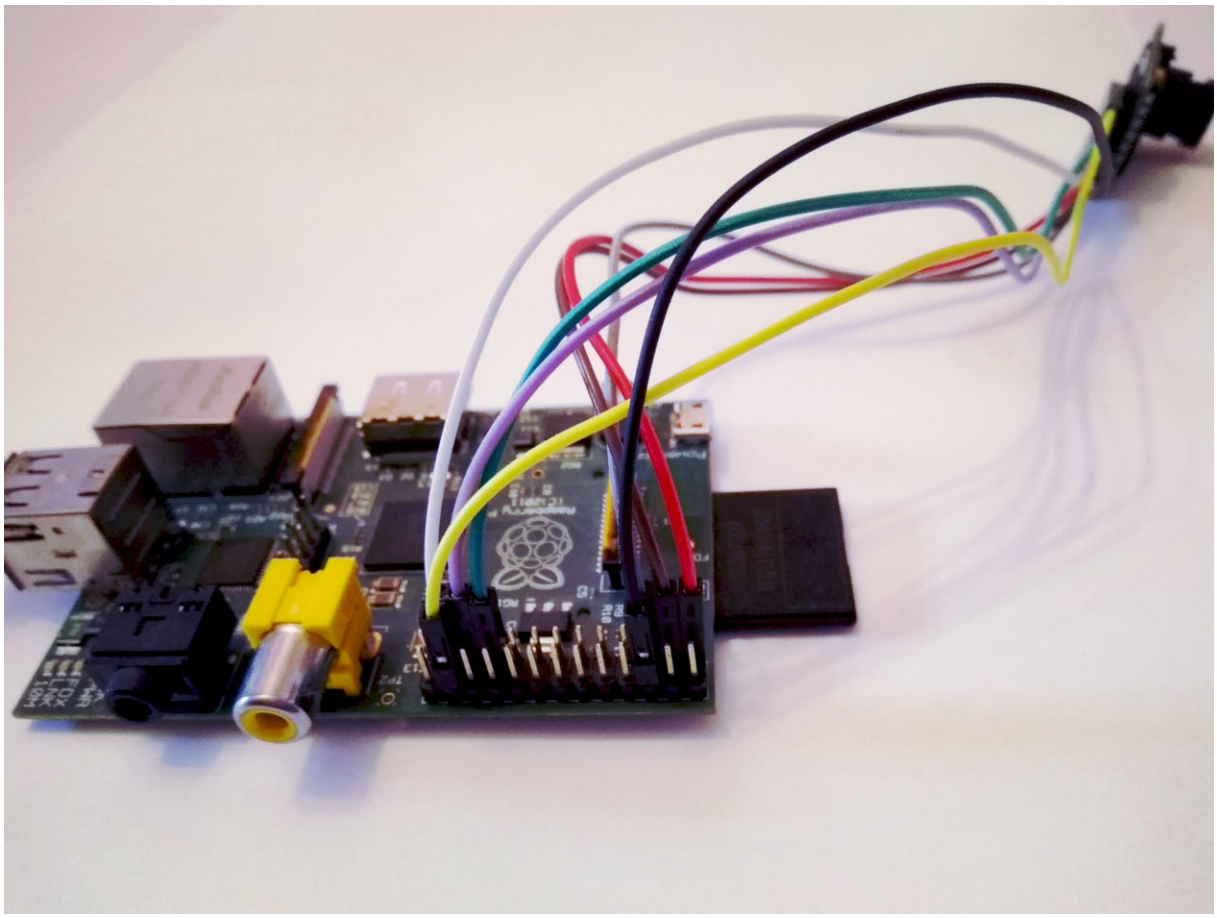
A continuación se detallan los pines que se utilizan para conectar la cámara con la Raspberry y la función de cada uno. Como puede observarse solo es necesario el uso de 8 pines para alimentar y manejar el módulo de ArduCAM.

Fig. 26: GPIO Raspberry Pi

Pin Raspberry GPIO	Señal ArduCAM
1	VCC
3	SDA
5	SCL
6	GND
19	MOSI
21	MISO
23	SCK
24	CS

Tabla 3: Conexiones entre Raspberry Pi y ArduCAM

Como puede observarse en la imagen, la conexión es muy simple en comparación con otro tipo de cámaras que necesitan más señales de salida y de control. En la figura 27 puede verse como queda el cableado de la cámara al conectarlo todo en la Raspberry.



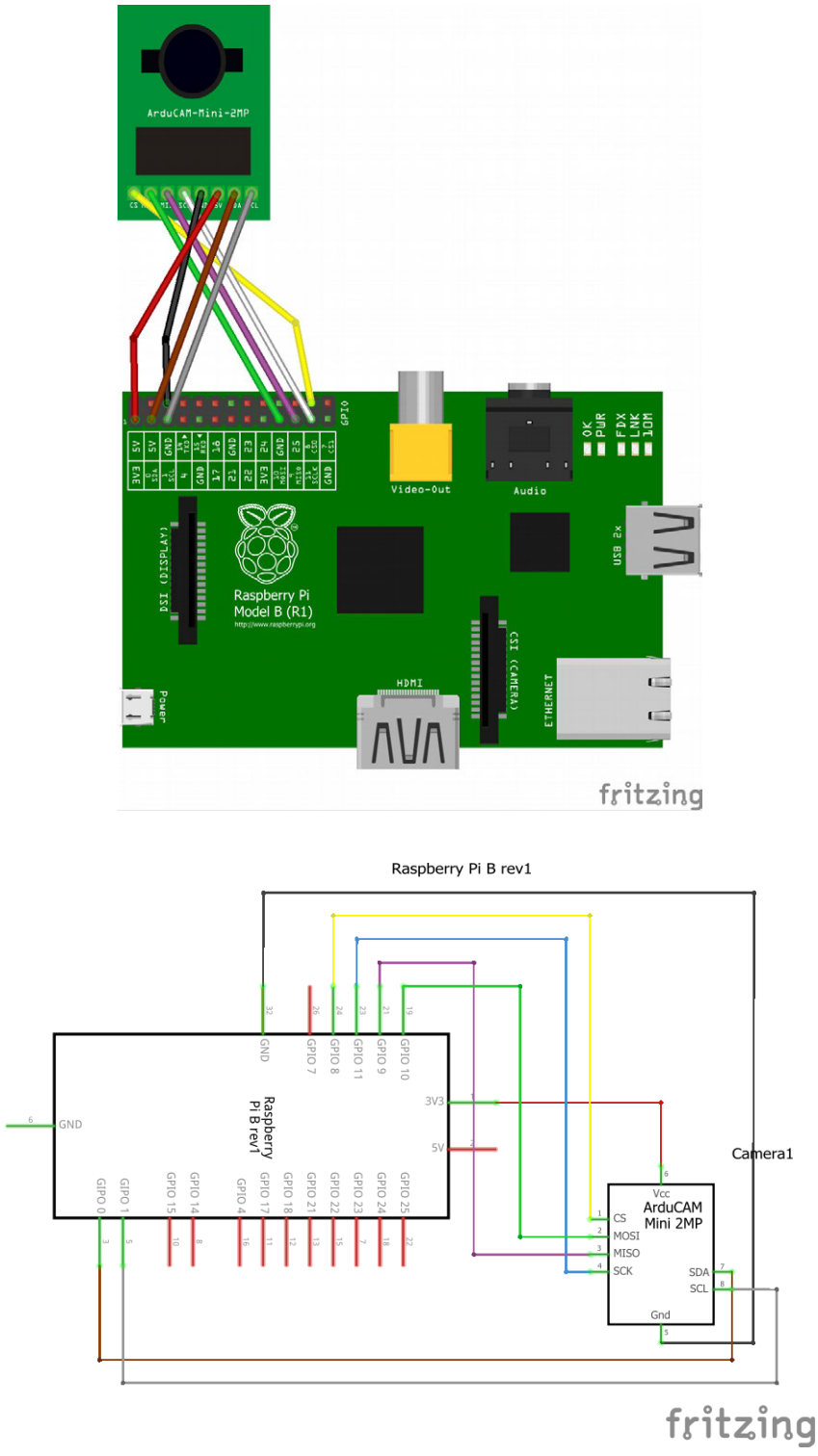


Fig. 27: Cableado de la cámara.

Capítulo 5: Implementación del sistema

A lo largo de este capítulo se detallará cómo se ha llevado a cabo el trabajo. Se comenzará con la configuración de los elementos que lo componen para, a continuación, explicar las pruebas individuales que se han realizado con cada elemento.

5.1 Raspberry Pi

En este apartado se detallarán las configuraciones realizadas a la Raspberry Pi así como las herramientas instaladas.

Como se indicó anteriormente, el sistema operativo escogido para la Raspberry fue Raspbian. Este sistema operativo es un *port* no oficial de Debian con opciones de compilación ajustadas para producir código optimizado para realizar cálculos en coma flotante en hardware, lo que proporciona un rendimiento significativamente mayor en aplicaciones que realizan una gran cantidad de cálculos de este tipo.

5.1.1 Instalación del sistema operativo

Para la instalación del sistema operativo, Raspberry proporciona dos métodos para realizarlo:

- A través de la herramienta NOOBS (New Out Of the Box Software) [Ras7,www], una herramienta desarrollada por la comunidad que permite instalar varios sistemas

operativos, entre ellos Raspbian, de forma sencilla y sin necesidad de estar conectado a Internet.

- Descargando la imagen de Raspbian desde la sección de descargas de la web de Raspberry Pi e instalándola en una tarjeta SD.

En este caso se decidió optar por el segundo método ya que para el primero era necesario contar con periféricos para seleccionar las opciones a la hora de instalar y no se disponía de un monitor HDMI al que conectar la Raspberry y no suponía ningún problema utilizar el otro método menos sencillo. Los pasos a seguir para realizar la instalación de la manera elegida son los siguientes:

1. Descargar la imagen del sistema que se desea y extraerla del fichero zip.
2. Introducir la tarjeta SD en el lector de tarjetas y comprobar la etiqueta de unidad que se le asigna (la letra asociada al dispositivo, C: en el caso del disco duro).
3. Descargar la aplicación Win32DiskImager de su página en Sourceforge [Sou,www] como archivo zip y extraer el ejecutable.
4. Ejecutar Win32DiskImager como administrador.
5. Seleccionar la imagen extraída anteriormente.
6. Seleccionar la etiqueta de volumen asignada a la tarjeta SD, teniendo cuidado de que sea la letra correcta ya que si ocurre una equivocación puede eliminarse todo el contenido del disco duro.
7. Pulsar el botón "Write" y esperar a que termine el proceso.
8. Cuando finalice se cierra el programa y se extrae la tarjeta, que ya está lista para ser introducida en la Raspberry Pi.

Con estos pasos se ha llevado a cabo la instalación básica del sistema, ahora queda configurar la Raspberry para poder conectarse a ella y manejar sus funciones en remoto para así no depender de periféricos.

5.1.2 Configuración de la comunicación WiFi

A continuación se explica cómo conectar y configurar el WiFi para crear una red *ad hoc* a la que conectarse con el portátil. Una red *ad hoc* es una red inalámbrica descentralizada que no depende de infraestructuras como routers o puntos de acceso para funcionar, el encaminamiento se realiza mediante el reenvío de datos hacia otros nodos.

Lo primero será conectar un cable Ethernet entre la Raspberry y el ordenador ya que, al tener que configurar el dongle previamente a su uso, la conexión se realizará a través de cable. Seguidamente será necesario conocer la dirección IP asignada automáticamente a la Raspberry, que debería estar en el rango entre 169.254.0.0 y 169.254.255.254 [Pih,www]. Para encontrar la dirección exacta se utilizó una aplicación de búsqueda de direcciones IP en red llamada Angry IP Scanner [Ang,www] a la que se indica el interfaz de red sobre el que se

quiere realizar el escaneo, la máscara de red para que el programa sepa el rango de direcciones en el que debe buscar, en este caso la máscara sería 255.255.0.0 y se presiona el botón Comenzar para que el escáner realice la búsqueda. Pasado un tiempo, que en este caso será más largo del habitual ya que la red es de tipo B y contiene más direcciones que una red doméstica, el programa muestra una lista de direcciones indicando cuales están activas y el nombre del equipo que la está usando (figura 28).

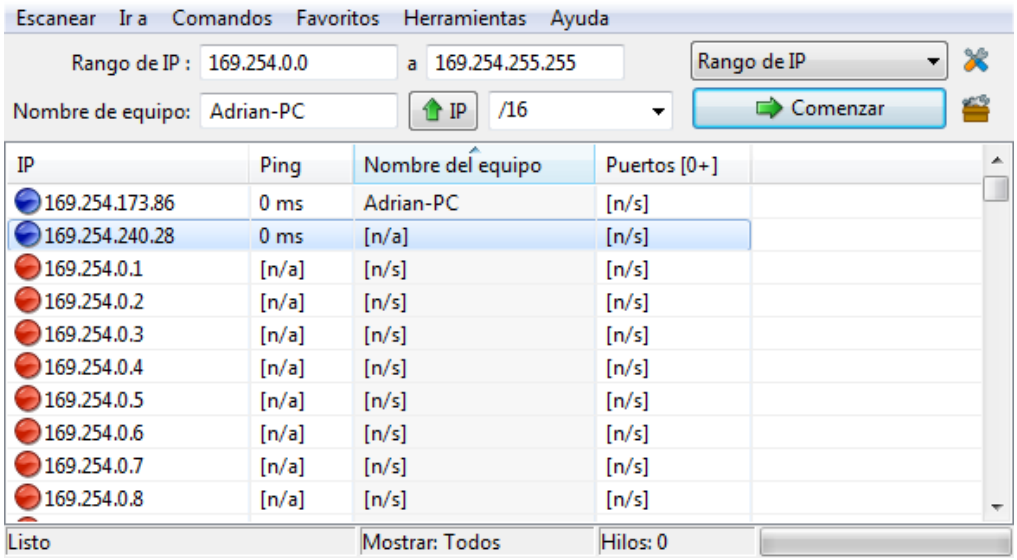


Fig. 28: Escaneo de Angry IP Scanner

Una vez conseguida la IP ya es posible conectarse a la Raspberry a través de SSH [Mit,www]. El servidor SSH está preinstalado por defecto en Raspian y se arranca como demonio al arrancar el sistema, así que solo es necesario introducir la dirección anteriormente conseguida en un cliente SSH, como por ejemplo PuTTY [Put,www], para conectarse (figura 29).

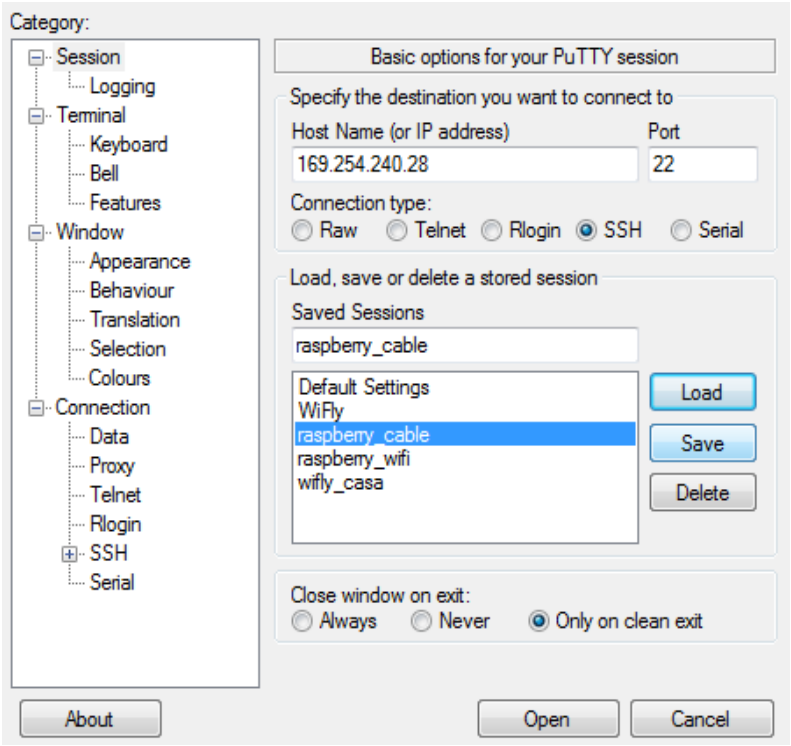


Fig. 29: Herramienta PuTTY para conexiones SSH

Si la conexión se realiza correctamente, aparecerá un mensaje indicando que el host al que se está intentando conectar es un host desconocido y pedirá confirmación para realizar la conexión y guardar la clave RSA del servidor a la caché de PuTTY (esto solo ocurre en la primera conexión, el mensaje no volverá a aparecer a no ser que se cambie la IP).

Una vez se ha conectado con el servidor SSH, se pedirá el usuario y la contraseña para acceder al sistema, que por defecto son "pi" para el usuario y "raspberry" para la contraseña. Si se desea cambiar la contraseña por seguridad puede realizarse con el comando:

```
~$ sudo passwd pi
```

El siguiente paso a realizar es la instalación del WiFi y su configuración para conectarse a la red doméstica. Para el adaptador WiFi elegido no fue necesario instalar ningún driver ya que una vez conectado apareció el interfaz virtual de red inalámbrica por defecto (wlan0) y fue posible realizar un escaneo de las redes cercanas. Para confirmar que el adaptador WiFi es reconocido y la interfaz inalámbrica esta activa se usa el comando:

```
~$ iwconfig
```

Si aparece la interfaz "wlan0" se puede realizar un escaneo para buscar las redes locales cercanas y comprobar que el adaptador funciona correctamente

```
~$ sudo iwlist wlan0 scan
```

En la figura 30 se observa el resultado de realizar el escaneo de redes que indica que el adaptador está listo para configurar la red.

```
pi@raspberrypi:~ $ sudo iwlist wlan0 scan
wlan0     Scan completed :
          Cell 01 - Address: 88:03:55:AF:52:D8
                    ESSID:"Orange-52D6"
                    Protocol:IEEE 802.11bgn
                    Mode:Master
                    Frequency:2.412 GHz (Channel 1)
                    Encryption key:on
                    Bit Rates:144 Mb/s
                    Extra:wpa_ie=dd1a0050f20101000050f20202000
                    IE: WPA Version 1
                        Group Cipher : TKIP
                        Pairwise Ciphers (2) : CCMP TKIP
                        Authentication Suites (1) : PSK
                    Extra:rsn_ie=30180100000fac020200000fac040
                    IE: IEEE 802.11i/WPA2 Version 1
                        Group Cipher : TKIP
                        Pairwise Ciphers (2) : CCMP TKIP
                        Authentication Suites (1) : PSK
                       Preauthentication Supported
                    IE: Unknown: DD8F0050F204104A0001101044000
637353139525732321024000930302E39362E3332301042000A4A333334313
                    Quality=100/100  Signal level=44/100
          Cell 02 - Address: FA:8F:CA:5A:BC:37
                    ESSID:""
                    Protocol:IEEE 802.11bgn
                    Mode:Master
                    Frequency:2.422 GHz (Channel 3)
                    Encryption key:off
                    Bit Rates:72 Mb/s
```

Fig. 30: Resultado del escaneo de redes.

Una vez comprobado que el adaptador funciona se pasará a configurar la red. En Linux esto se realiza editando el archivo "interfaces" que se encuentra en /etc/networks/ aunque, al tratarse de una red con seguridad WPA2-PSK, el sistema delega la tarea al demonio "wpa_supplicant" que implementa negociación de claves y autenticación WPA, y controla el roaming y la asociación/autenticación IEEE 802.11 del controlador wlan. La configuración de este demonio se realiza editando el fichero "wpa_supplicant.conf" que se encuentra en /etc/wpa_supplicant/ y añadiendo la red como en la figura 31.

```
network={
        ssid="nombre_de_red"
        psk="contraseña"
}
```

Fig. 31: Datos de la red en wpa_supplicant.conf

Una vez editado el fichero y añadidos los datos de la red, se deberá reiniciar la interfaz inalámbrica de la siguiente forma:

```
~$ sudo ifdown wlan0
~$ sudo ifup wlan0
```

Tras realizar estos pasos la conexión con la red local ya estaría lista para usarse. Esta conexión se utilizó durante buena parte del desarrollo del proyecto hasta cambiarla finalmente por una conexión *ad hoc*.

Es recomendable actualizar el registro de paquetes de software y los paquetes preinstalados en el sistema, para llevarlo a cabo se ejecuta el comando:

```
~$ sudo apt-get update
~$ sudo apt-get upgrade
```

En este proyecto se ha utilizado la red *ad hoc* para comunicar la Raspberry Pi con el ordenador portátil que actúa como receptor de la información. A continuación se detallará la configuración necesaria para que la Raspberry cree la red, la parte de configuración necesaria para el portátil se detallará en un apartado posterior ya que este se centra en las configuraciones relativas a la Raspberry Pi.

Para configurar la red que se desea crear, se debe editar el fichero "interfaces" mencionado anteriormente y definir la red como se indica en la figura 32:

```
allow-hotplug wlan0
iface wlan0 inet static
    address 192.168.1.1
    mask 255.255.255.0
    wireless-channel 1
    wireless-essid Red-Raspberry
    wireless-mode ad-hoc
```

Fig. 32: Configuración de la red ad hoc

La primera línea indica la posibilidad de desconexión "en caliente" del interfaz (sin apagar la máquina), a continuación se indica que la dirección será estática y después se establece la dirección que tendrá la Raspberry en la red, la máscara de red, el canal en el que estará, el SSID o identificador de la red y el modo, en este caso *ad-hoc*.

5.1.3 Configuración de servidores remotos

Para la conexión en remoto con la Raspberry Pi se utilizarán servidores SSH y VNC [Cam2,www]. Como ya se dijo anteriormente, el servidor SSH viene instalado por defecto en el sistema operativo [Ras3,www] y se arranca como demonio al encender la máquina, por lo que no se comentará nada relativo a su instalación o configuración. En caso de que no esté habilitado, se hará uso de la herramienta de configuración del sistema raspi-config. Esta herramienta se ejecuta mediante el comando:

```
~$ sudo raspi-config
```

Tras ejecutar el comando aparecerá el menú de la herramienta (figura 33) y se deberá entrar en la opción 9 (Advanced Options). Ahí aparecerá la opción SSH desde la que será posible elegir si se desea que el servidor SSH esté habilitado o no.

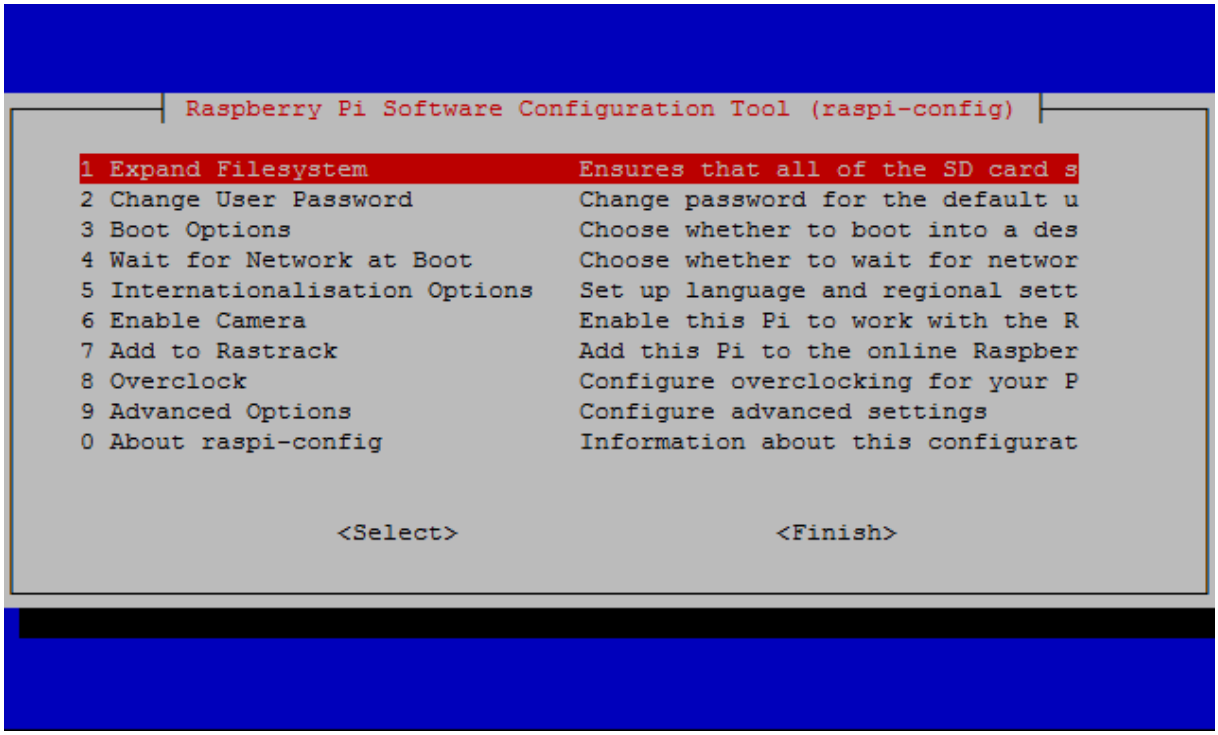


Fig. 33: Menú de la herramienta raspi-config

También es recomendable hacer uso del servidor VNC (Virtual Network Computing) que es un sistema multiplataforma para compartir escritorio gráfico y que permite transmitir los eventos del teclado y del ratón. Este sistema resultó muy útil para comprobar las imágenes que estaba capturando la cámara, ya que ésta está directamente conectada a la Raspberry. El

servidor VNC, al igual que el SSH, está también instalado por defecto y habilitado en el sistema. La forma para habilitarlo en caso de que no lo estuviera es idéntica a como se habilita el servidor SSH mediante la herramienta raspi-config [Ras4,www].

Una vez se ha verificado que el servicio VNC está habilitado, se deberá arrancar el servidor y crear un escritorio virtual al que conectarse mediante el programa cliente. La forma más simple de arrancar el servidor será mediante el comando:

```
~$ vncserver
```

Este comando pedirá una contraseña para el acceso la primera vez que sea ejecutado y a continuación realizará otro tipo de inicializaciones. Para crear un escritorio virtual se ejecuta el mismo comando que lo crea con las opciones por defecto que se encuentran en el archivo /home/pi/.vnc/xstartup. Hay que prestar atención a la salida que devuelve el comando al terminar ya que se indicará la IP del servidor (se conoce previamente ya que es a la que se ha conectado por SSH) y el número de escritorio virtual creado, el cual será necesario conocer a la hora de conectarse desde el cliente. El comando también permite parametrización tal como las dimensiones del escritorio, el número de escritorio a crear, la profundidad de colores y varias más que pueden ser consultadas mediante el comando:

```
~$ man vncserver
```

De las diversas opciones de parametrización, la más relevante es la opción -kill que dado un número de escritorio virtual fuerza su eliminación, lo cual es de utilidad para eliminar un escritorio virtual una vez ha terminado de usarse.

En la parte del cliente, en este caso un ordenador portátil, se ha descargado la herramienta TightVNC Viewer.

5.1.4 Descarga e instalación de bibliotecas

La primera biblioteca necesaria para el desarrollo del proyecto es WiringPi. Esta biblioteca es usada por el software proporcionado por ArduCAM para interactuar con los pines de entrada/salida de la Raspberry. Para su instalación se ha hecho uso del comando para instalar paquetes de los repositorios de Raspbian:

```
~$ sudo apt-get install wiringpi
```

Tras esto será necesario descargar la biblioteca que ArduCAM provee para interactuar con la cámara. Puede descargarse directamente desde el repositorio en GitHub de ArduCAM:

```
~$ sudo wget
https://github.com/ArduCAM/RaspberryPi/archive/master.zip
```

Para descomprimir el archivo se debe usar una herramienta de descompresión compatible con ZIP, en este caso se usó unzip. Si no está instalado se puede instalar de los repositorios de Raspbian de la siguiente manera:

```
~$ sudo apt-get install unzip
```

Para realizar la descompresión bastará con usar el comando:

```
~$ sudo unzip master.zip
```

5.2 Portátil

Aquí se explicarán las configuraciones y aplicaciones necesarias para que el ordenador portátil actúe como receptor.

En el portátil usado se utilizó como sistema operativo Windows 7 ya que varias herramientas de las usadas solamente eran compatibles con sistemas Windows. Aún así el proyecto es multiplataforma y debería funcionar sin problema en cualquier sistema operativo que se utilice en el ordenador receptor mientras tenga una tarjeta de red inalámbrica bien configurada.

En este apartado se presupondrá que el sistema operativo del ordenador ya está instalado y configurado para usarse, ya que no es el objeto de este capítulo la explicación de como se lleva a cabo.

5.2.1 Configuración de la comunicación

Como ya se indicó en el apartado referido a la Raspberry, el primer paso para configurar la comunicación inalámbrica es conectarse por cable con ella y conseguir la dirección IP mediante un escáner de red. A continuación una vez conseguida la dirección ya es posible conectarse a través de SSH, para ello se utiliza la herramienta PuTTY que es un cliente SSH y telnet para Windows de código abierto desarrollado y mantenido por un grupo de voluntarios [Gre,www].

Tras configurar la red *ad hoc* en la Raspberry, es necesario cambiar los ajustes de red del portátil para que se conecte correctamente a la red creada.

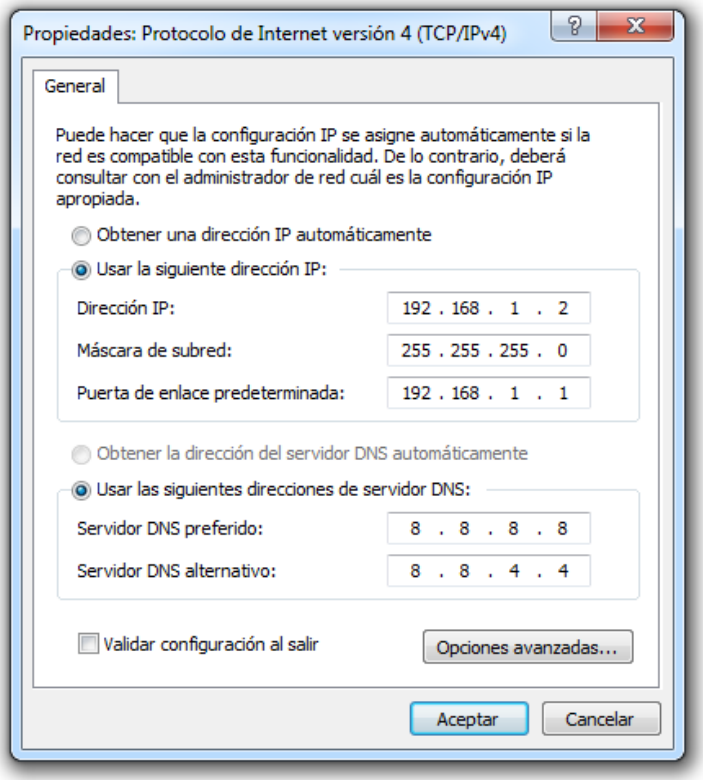


Fig. 34: Ajustes de red en el portátil

Como puede observarse, la dirección pertenece a la red que se ha creado en la Raspberry y la máscara de red indica que se trata de una red privada. La dirección indicada en la puerta de enlace es la de la Raspberry ya que es la misma usada por el router doméstico por lo que resulta más cómodo cambiar entre una red y otra sin tener que cambiar constantemente los ajustes de red. La dirección de los servidores DNS usada es la de los de Google ya que las opciones de red de Windows no permiten dejar esos campos en blanco aunque no sean necesarios.

Para el acceso en modo gráfico se descargó TightVNC, un cliente VNC gratuito ligero compatible con el sistema operativo Windows 7 utilizado en el portátil usado como receptor. La herramienta puede descargarse desde la web de TightVNC [Tig,www]. En el cuadro en el que se debe indicar la dirección del host remoto se indica también el número de escritorio virtual separado por dos puntos (:).

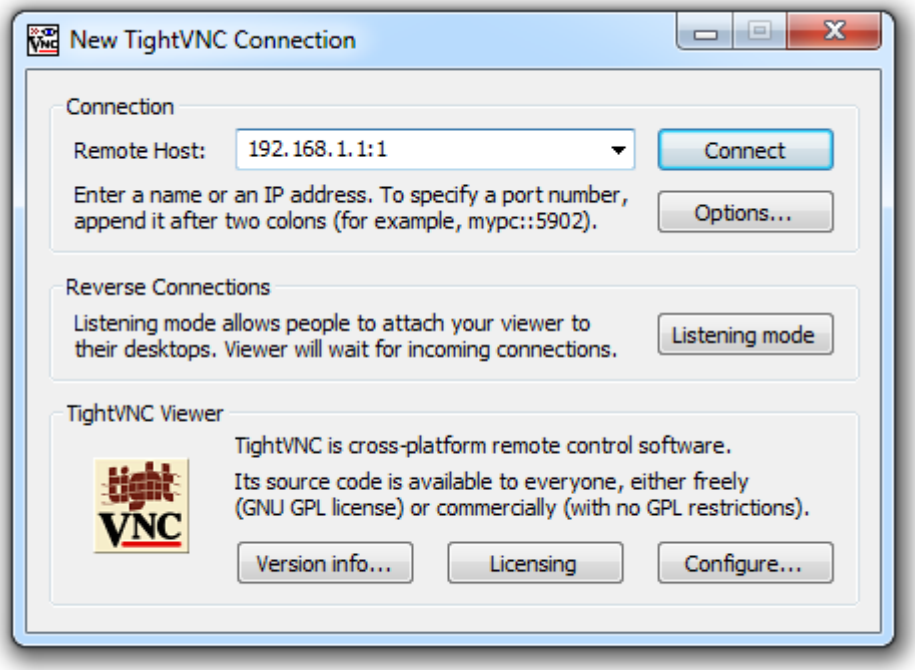


Fig. 35: Cliente TightVNC Viewer

5.2.2 IDE

Para desarrollar el software encargado de recibir la información y mostrarla en el lado del cliente se decidió utilizar un lenguaje que permitiera la portabilidad del código. Por eso se barajaron lenguajes semi-interpretados tales como Java [Jav,www] o Python [Pyt,www]. Entre esos dos se decidió usar Java por estar más familiarizado con ese lenguaje.

Como entorno de desarrollo se ha utilizado Eclipse [Ecl3,www], que ofrece distintos IDEs de código abierto para diversos lenguajes de programación, aunque su plataforma principal (orientada al desarrollo en Java) se puede modificar y extender por medio de plugins que pueden descargarse de Eclipse Marketplace [Ecl2,www] para, por ejemplo, desarrollar aplicaciones Android, desarrollar en lenguaje Python, añadir herramientas de control de versiones como Git o SVN y multitud de opciones más.

Java	Python
Tipado estático	Tipado dinámico
Bloques de código con llaves	Bloques de código con indentación
Recolector de basura	Conteo de referencias

Tabla 4: Diferencias entre Java y Python.

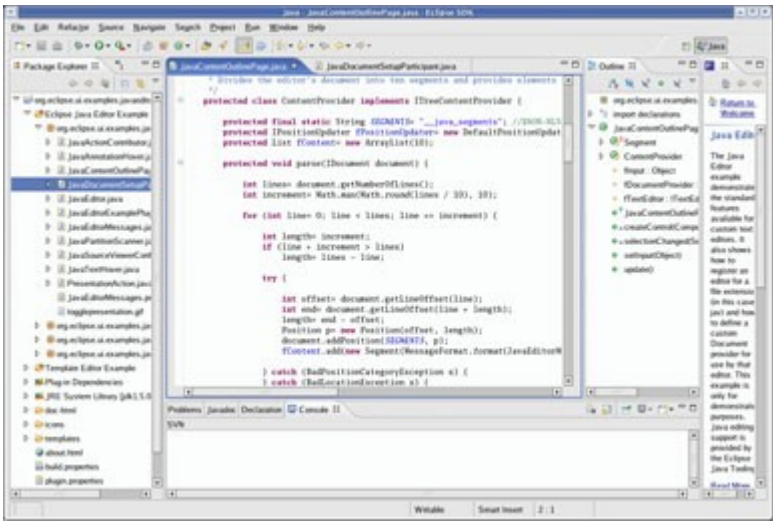


Fig. 36: IDE de escritorio de Eclipse [Ecl,www]

En la versión desarrollada usando un PIC32 como controlador se usó como entorno de desarrollo MPLAB X [Mic12,www], que es un entorno de desarrollo multiplataforma basado en el entorno de código abierto NetBeans [Net,www] de Oracle [Ora2,www] y al que se añadió la plataforma MPLAB Harmony [Mic13,www] que permite configurar y desarrollar código para PIC32 de forma rápida ya que esta orientado a que sus drivers y bibliotecas funcionen juntas con el menor esfuerzo posible dada la naturaleza modular de la plataforma.

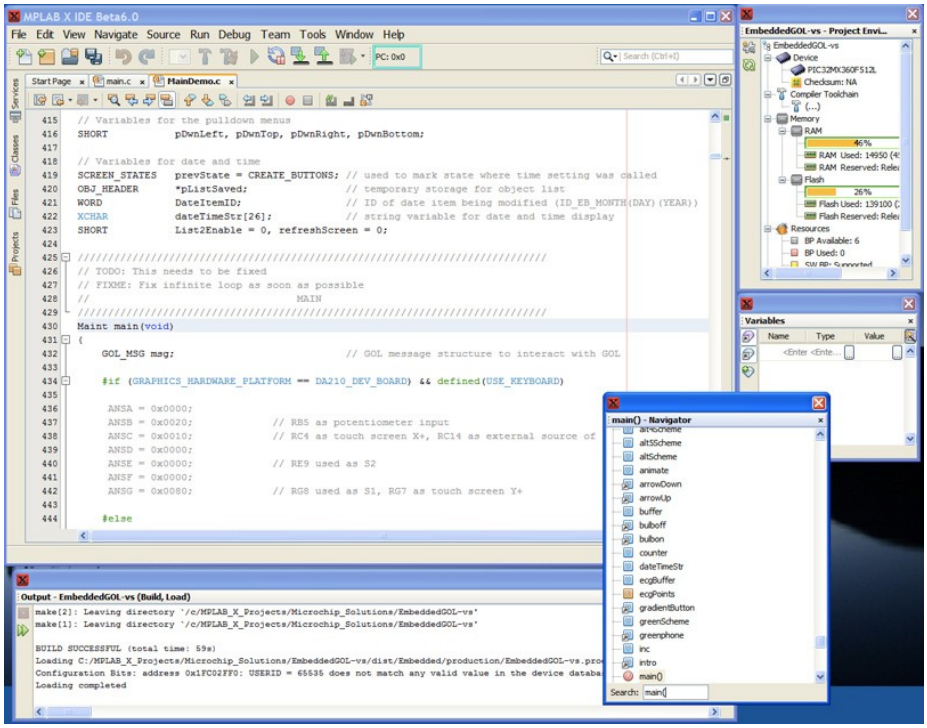


Fig. 37: MPLAB X IDE en su versión Beta [Mic2,www].

MPLAB X permite también la inclusión de plugins que ayudan a agilizar el desarrollo de código para PIC32 como por ejemplo el MPLAB Harmony Configurator, un plugin que permite añadir y configurar módulos tales como UART, buses, timers, ADCs, etc. de forma gráfica y genera el código de configuración del hardware deseado, genera el código relacionado con el middleware y actualiza automáticamente el proyecto de MPLAB X con todos los archivos necesarios.

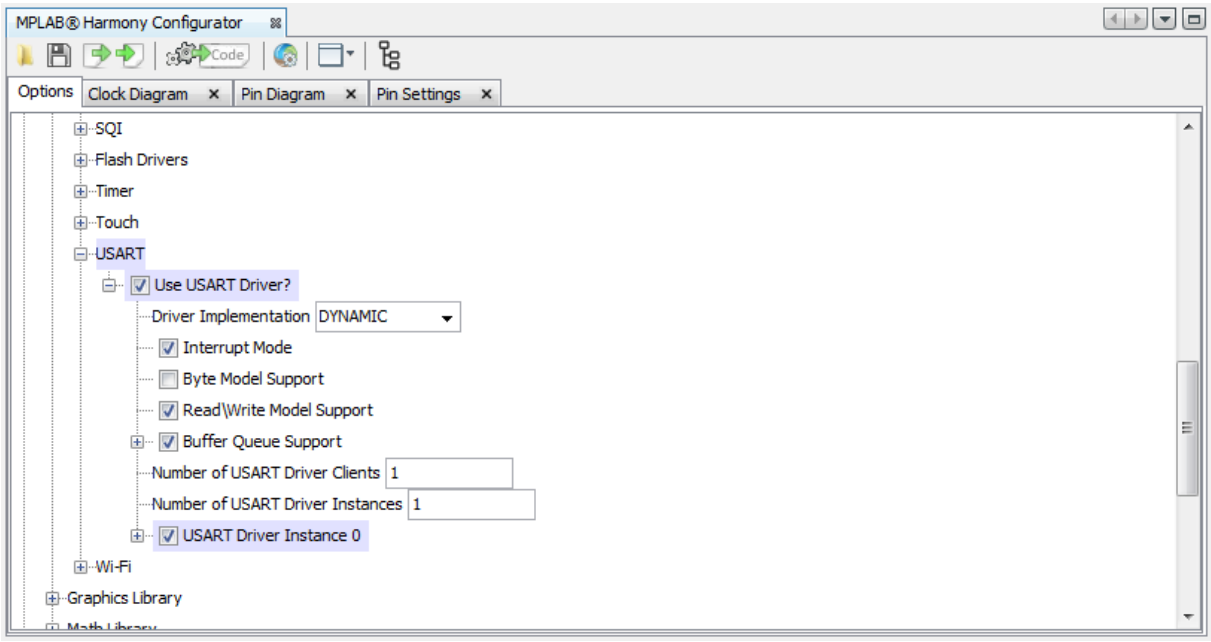


Fig. 38: Pestaña de MPLAB Harmony Configurator

5.3 Pruebas realizadas

Una vez se ha terminado de instalar y configurar el software necesario, se pasará a realizar las pruebas pertinentes de los componentes del proyecto.

Prueba básica de ArduCAM

La primera prueba realizada fue la prueba de captura que ArduCAM ofrece junto con su biblioteca en su repositorio en GitHub [Git2,www]. Esta prueba se descarga al descargar el archivo zip que contiene la biblioteca y que se mencionó en un apartado anterior.

La biblioteca de control de la cámara para Raspberry Pi está escrita en el lenguaje C [Rit,93], por lo que será necesario contar con un compilador para poder compilar la biblioteca y los programas que hagan uso de ella. En Raspbian, como en la mayoría de los sistemas GNU/Linux, el compilador que viene por defecto es GCC. GCC (GNU Compiler

Collection) [Gnu,www] es una colección de compiladores desarrollada por Richard Stallman [Sta,www] para el sistema GNU que ofrece interfaces para C, C++ [Str,97], Objective C [App,www], Fortran [Ibi,www], Java, Ada [Ada,www] y Go [Gol,www].

Para compilar la biblioteca y los programas de ejemplo se utiliza el comando *make* [Gnu2,www] que es una utilidad de GNU usada para mantener grupos de programas. Para ello el comando sigue las instrucciones indicadas en un fichero llamado Makefile (figura 39) en el que se detallan los pasos a seguir. Una vez detalladas las relaciones entre los diferentes ficheros que componen el proyecto, el comando que se ejecuta para llevar a cabo las tareas será el siguiente:

```
~$ sudo make
```

```
all :  ov2640_capture ov2640_4cams_capture ov5642_capture ov5642_4cams_capture

objects = arducam.o arducam_arch_raspberrypi.o


ov2640_capture : $(objects) arducam_ov2640_capture.o
	gcc -o ov2640_capture $(objects) arducam_ov2640_capture.o -lwiringPi -Wall
ov2640_4cams_capture : $(objects) arducam_ov2640_4cams_capture.o
	gcc -o ov2640_4cams_capture $(objects) arducam_ov2640_4cams_capture.o -lwiringPi -Wall
ov5642_capture : $(objects) arducam_ov5642_capture.o
	gcc -o ov5642_capture $(objects) arducam_ov5642_capture.o -lwiringPi -Wall
ov5642_4cams_capture : $(objects) arducam_ov5642_4cams_capture.o
	gcc -o ov5642_4cams_capture $(objects) arducam_ov5642_4cams_capture.o -lwiringPi -Wall
capture_video : $(objects) capture_video.o
	gcc -o capture_video $(objects) capture_video.o -lwiringPi -Wall

arducam.o : arducam.c
	gcc -c arducam.c -lwiringPi -Wall
arducam_arch_raspberrypi.o : arducam_arch_raspberrypi.c
	gcc -c arducam_arch_raspberrypi.c -lwiringPi -Wall


arducam_ov2640_capture.o : arducam_ov2640_capture.c
	gcc -c arducam_ov2640_capture.c -lwiringPi -Wall
arducam_ov2640_4cams_capture.o : arducam_ov2640_4cams_capture.c
	gcc -c arducam_ov2640_4cams_capture.c -lwiringPi -Wall
arducam_ov5642_capture.o : arducam_ov5642_capture.c
	gcc -c arducam_ov5642_capture.c -lwiringPi -Wall
arducam_ov5642_4cams_capture.o : arducam_ov5642_4cams_capture.c
	gcc -c arducam_ov5642_4cams_capture.c -lwiringPi -Wall
capture_video.o : capture_video.c
	gcc -c capture_video.c -lwiringPi -Wall

clean :
	rm -f  ov2640_capture ov2640_4cams_capture ov5642_capture ov5642_4cams_capture capture_
video  $(objects) *.o
```

Fig. 39: Contenido del fichero Makefile

Si la compilación es correcta, en el directorio actual deberían aparecer los archivos con extensión ".o", que son ficheros objeto generados por el compilador y usados por el enlazador para generar el fichero ejecutable, y los ficheros ejecutables sin extensión. Para comprobar que están los ficheros se usa el comando *ls* que lista el contenido del directorio actual o que se pasa como parámetro:

```
~$ ls -l
```

Con el parámetro "-l" la salida del comando aparece más ordenada y se muestra información adicional que puede resultar de utilidad como los permisos de los ficheros, el usuario propietario del fichero o la fecha de modificación, que puede resultar útil para saber si un fichero ha sido modificado o actualizado.

Para ejecutar el programa, basta invocarlo desde la línea de comandos acompañándolo de los parámetros necesarios para su ejecución, los cuales pueden consultarse invocando el programa sin parámetros:

```
~$ sudo ./ov2640_capture
```

```
Usage: ./ov2640_capture [-s <resolution>] | [-c <filename] -s <resolution>
 Set resolution, valid resolutions are:
                  160x120
                  176x144
                  320x240
                  352x288
                  640x480
                  800x600
                  1024x768
                  1280x1024
                  1600x1200
 -c <filename>   Capture image
```

Fig. 40: Ayuda del programa de captura

La salida del comando indica la forma de usarlo (figura 40). Como puede observarse deben usarse los parámetros "-c" para indicar que se va a especificar la resolución de la imagen capturada, el nombre del archivo que va a generar el programa y la resolución de la imagen.

Con el comando *ls* también aparecen los permisos del fichero. Para que pueda ejecutarse, en los permisos debe aparecer la letra "x" que hace referencia al permiso de ejecución del fichero. Si se desean cambiar los permisos para añadir permisos de ejecución puede realizarse con el siguiente comando:

```
~$ sudo chmod a+w ov2640_capture
```

Una vez se ha comprobado que el fichero puede ejecutarse y se conoce el modo de uso del mismo, se procede a ejecutarlo desde la línea de comandos:

```
~$ sudo ./ov2640_capture -c prueba.jpg 320x240
```

Si el programa se ejecuta de forma correcta, la salida generada debería ser como la de la figura 41.

```
SPI interface OK!
OV2640 detected
Changed resolution to 320x240
Start capture
CAM1 Capture Done
Reading FIFO
```

Fig. 41: Salida del programa de ejemplo

Una vez el programa termina se puede ejecutar el comando *ls* para comprobar que se ha creado el archivo "prueba.jpg", si está creado se arranca el servidor VNC con el comando *vncserver*, como fue indicado anteriormente, para poder ver la imagen resultante en el entorno de ventanas.

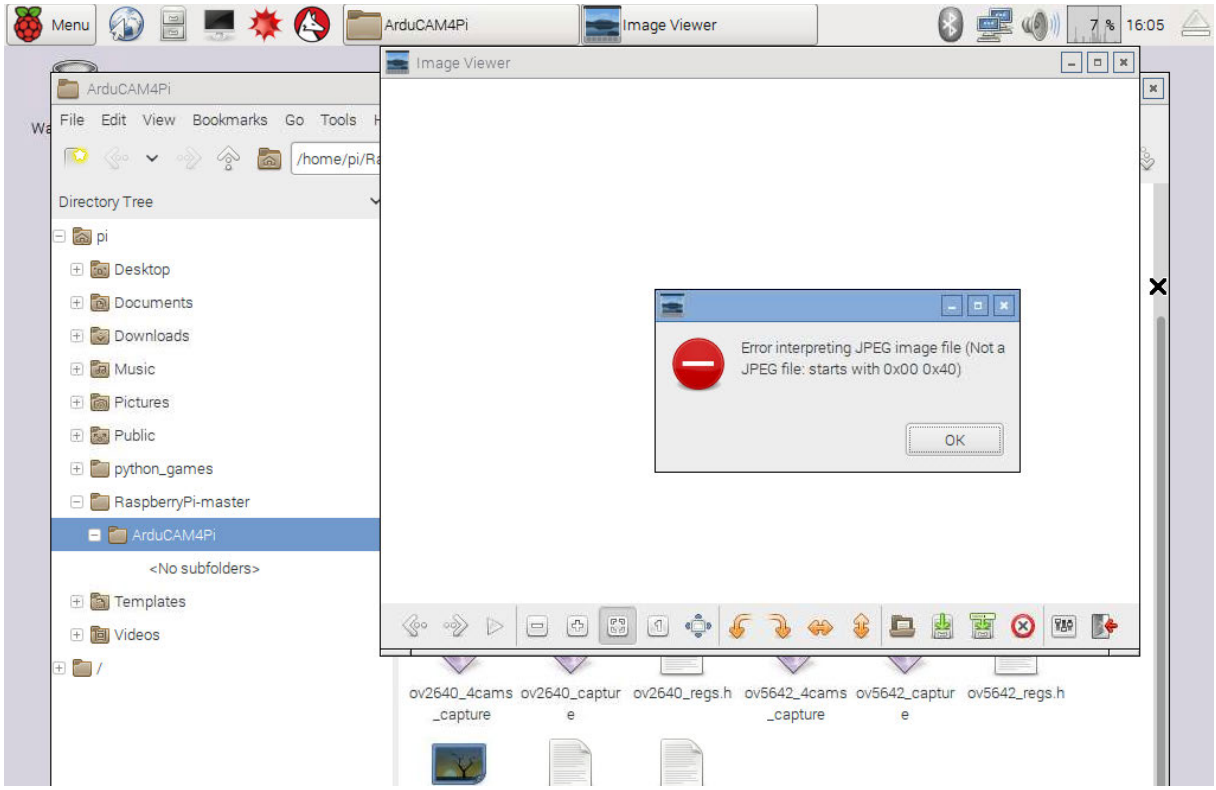


Fig. 42: Resultado erroneo de la prueba de ArduCAM

Al intentar abrir el fichero generado por el programa de ejemplo suministrado por ArduCAM, aparece el mensaje de error mostrado en la figura 42. Se analizó el ejemplo para encontrar el posible error y se cambió el modo de lectura del buffer FIFO de la cámara del modo "*burst*" al modo "normal", tras este cambio fue posible realizar la captura de imágenes de forma correcta por lo que, a partir de este punto, el modo de lectura usado durante todo el desarrollo fue el modo "normal". La diferencia entre ambos modos radica en el número de

operaciones de lectura que se deben realizar a través del bus SPI para leer datos. Mientras que en el modo "normal" se debe realizar una operación de lectura por cada dato leído, en el modo "ráfaga" ("*burst*" en inglés) sólo se necesita enviar un comando de lectura para leer múltiples datos de la cámara en un solo ciclo de lectura, por lo que el modo recomendado por el fabricante para leer múltiples imágenes es el modo ráfaga para conseguir un mejor rendimiento. En las figuras 43 y 44 puede observarse la diferencia entre ambos modos de lectura por medio de un cronograma.

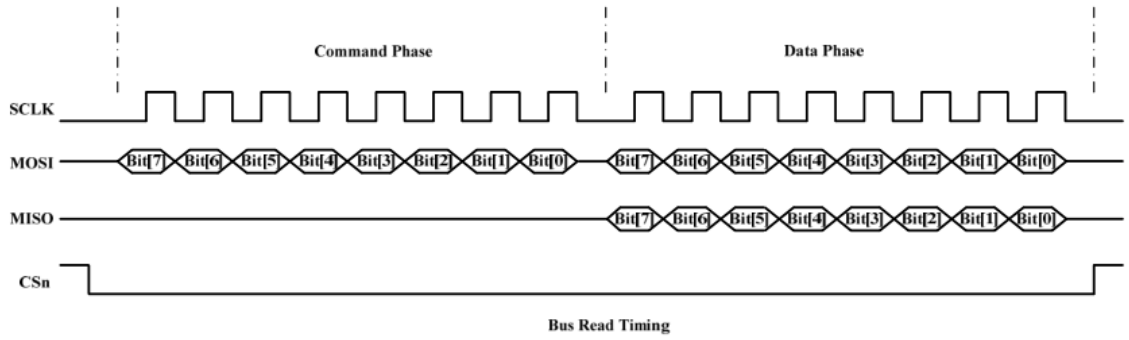


Fig. 43: Cronograma de una lectura sencilla [Ard2,www].

La lectura sencilla mostrada en el cronograma anterior se utiliza para leer valores de los registros internos del chip que controla la cámara (ArduChip) y para realizar lecturas individuales del buffer FIFO utilizadas en el modo "normal" de lectura. Esta operación de lectura se compone de dos fases, la fase de comando y la fase de datos, que se ejecutan cuando la señal de selección de componente (CSn o *chip select* en inglés) está a nivel bajo. Los primeros 8 bits pertenecen al byte comando que se decodifican como la dirección de un registro, los siguientes 8 bits son un byte ficticio que se escribe en la señal MOSI del bus SPI y el contenido del registro aparece en la señal MISO del bus.

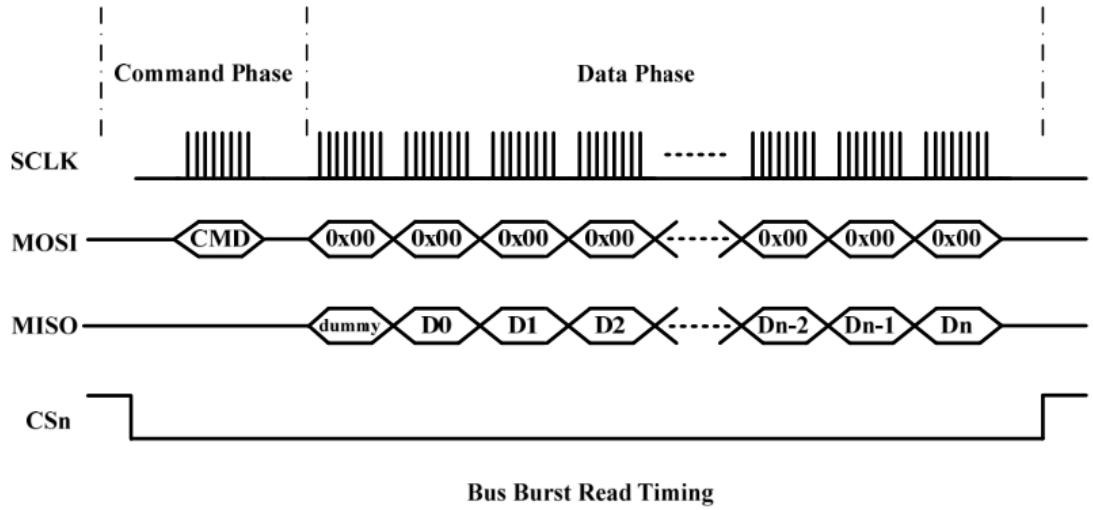


Fig. 44: Cronograma de una lectura en ráfaga [Ard2,www].

La lectura en "ráfaga" mostrada en el cronograma anterior es usada solamente en el modo de lectura en "ráfaga" del buffer FIFO [Ard2,www]. Se compone de una fase de comando en la que se envía el comando correspondiente a este tipo de lectura, seguida de múltiples fases de datos con el fin de obtener el doble de rendimiento con respecto al modo de lectura normal.

En cuanto al código de la prueba, únicamente se realizaron cambios en la forma en la que son leídos los bytes de la cámara. En la versión original (figura 45) se leen bloques de 4KB a través del bus SPI y se almacenan en un buffer que posteriormente se escribe en un fichero creado para almacenar la imagen capturada y cuyo nombre se envía como parámetro.

```
digitalWrite(CAM1_CS,LOW);  //Set CS low
set_fifo_burst(BURST_FIFO_READ);
arducam_spi_transfers(buffer,1);//dummy read
uint32_t i=0;
while(len>4096)
{
  arducam_spi_transfers(buffer+i, 4096);
  len -= 4096;
  i += 4096;
}
arducam_spi_transfers(buffer+i, len);

fwrite(buffer, sizeof(uint8_t), i+len, fp1);
digitalWrite(CAM1_CS,HIGH);  //Set CS HIGH
```

Fig. 45: Código de lectura en modo ráfaga

En la versión en la que se utiliza el modo normal de lectura (figura 46) se realiza una lectura byte a byte de la información de la cámara y se almacenan en un buffer, al igual que en la versión original, para luego escribir el buffer en un fichero.

```
uint32_t i=0;
while(len>0)
{
  buffer[i] = arducam_read_fifo(CAM1_CS);
  len--;
  i++;
}

fwrite(buffer, sizeof(uint8_t), i, fp1);
```

Fig. 46: Código de lectura byte a byte

Con ese cambio fue posible capturar imágenes y el ejemplo proporcionado por ArduCAM funcionó correctamente, como puede observarse en la figura 47.

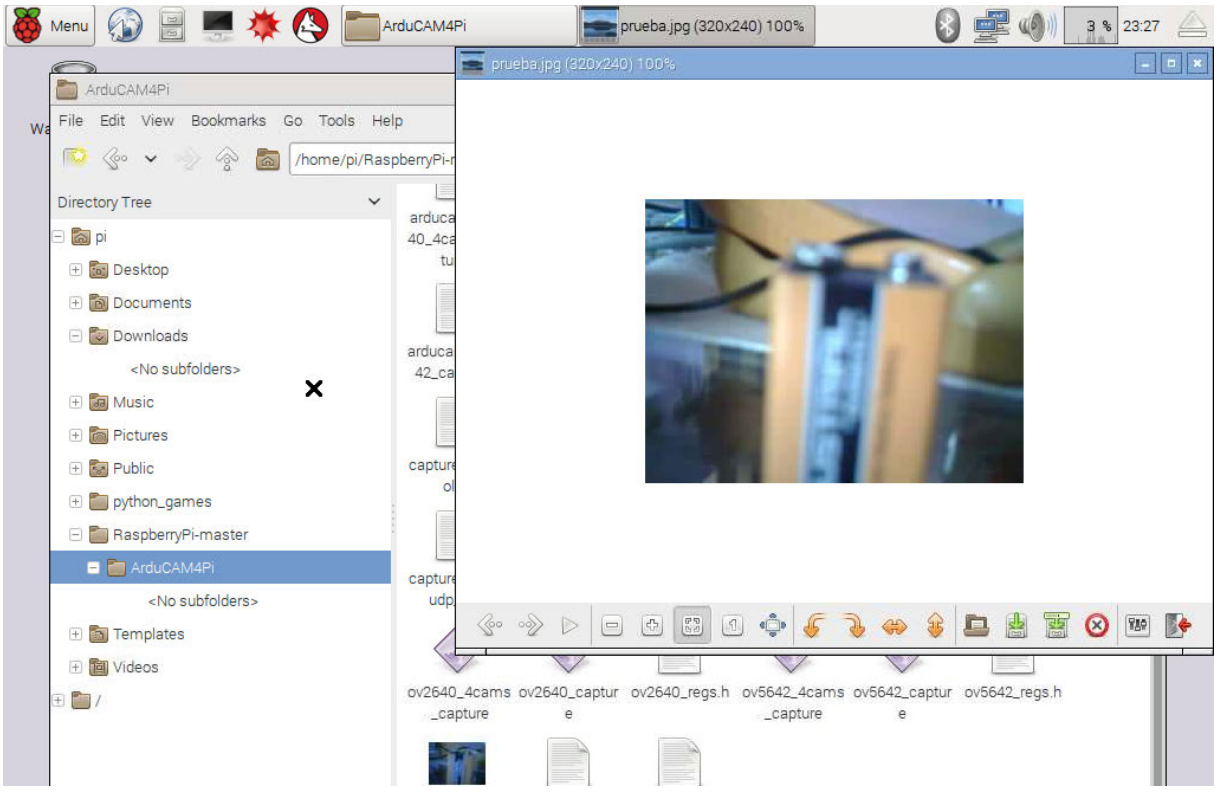


Fig. 47: Captura generada por el programa de prueba

Como paso previo a la captura de vídeo, se implementó otro programa que realizaba un número determinado de capturas y creaba un fichero por cada una de ellas. Para ello se

ejecutaba el código mostrado en la figura 46 en bucle, habiéndose creado previamente un descriptor de fichero por cada imagen a crear.

Prueba con comunicación inalámbrica

Una vez se ha probado la captura de imágenes múltiples, lo siguiente que se ha de realizar es el programa de captura de vídeo, para lo que se tomará como base el programa llamado "multicaptura" adjuntado en el CD. Para que fuese posible la comunicación, el programa hace uso de sockets de red. Un socket es un punto final de un enlace de comunicación bidireccional entre dos programas que se ejecutan en red. Están unidos a un número de puerto de manera que la capa TCP puede identificar la aplicación a la que los datos van destinados [Ora,www]. Ese punto final está formado por una dirección IP y un número de puerto. Cada conexión TCP puede ser identificada inequívocamente por sus dos puntos finales.

En sistemas GNU/Linux, la biblioteca que provee las funciones para manejar sockets forma parte de la biblioteca *glibc* [Gnu3,www] que define las llamadas al sistema y las funcionalidades básicas del lenguaje C como *printf, malloc, open...* Esta biblioteca se diseñó de forma que sea portable y dé un rendimiento alto como biblioteca del lenguaje C para sistemas que utilicen Linux como núcleo. Sigue los estándares más relevantes incluyendo las normas ISO C11 y POSIX.1-2008. Para utilizar las funciones y los tipos relacionados con sockets se debe incluir la biblioteca *sys/socket.h* en la que aparecen definidos los tipos de socket existentes: SOCK_STREAM para los sockets TCP, SOCK_DGRAM para sockets UDP y SOCK_RAW para sockets de red de nivel bajo no usados habitualmente para desarrollar aplicaciones.

La diferencia entre estos tipos se encuentra en cómo está orientada la conexión en cada uno de ellos. Mientras que el protocolo TCP está orientado a la conexión, proporciona un flujo de datos ordenado y posee mecanismos para detección de fallos en la comunicación, el protocolo UDP no está orientado a la conexión, no requiere de un canal permanente abierto para realizar la comunicación entre los pares y no proporciona fiabilidad en el orden de los mensajes, existe la posibilidad de que aparezca un mensaje duplicado y no permite saber si el mensaje ha sido entregado o no, lo que hace que el protocolo UDP sea más rápido y consuma menos recursos por lo que es recomendable en aplicaciones en las que no se necesite una gran fiabilidad en los datos. En la siguiente figura puede verse lo comentado anteriormente.

En el caso de este proyecto se probaron ambos protocolos optándose finalmente por utilizar TCP ya que no se encontró mejoría en la latencia ni en el rendimiento del sistema utilizando UDP y simplificaba el programa cliente el uso de TCP en el lado del servidor por cuestiones de la biblioteca de adquisición de imágenes de Java de la que se hablará posteriormente.

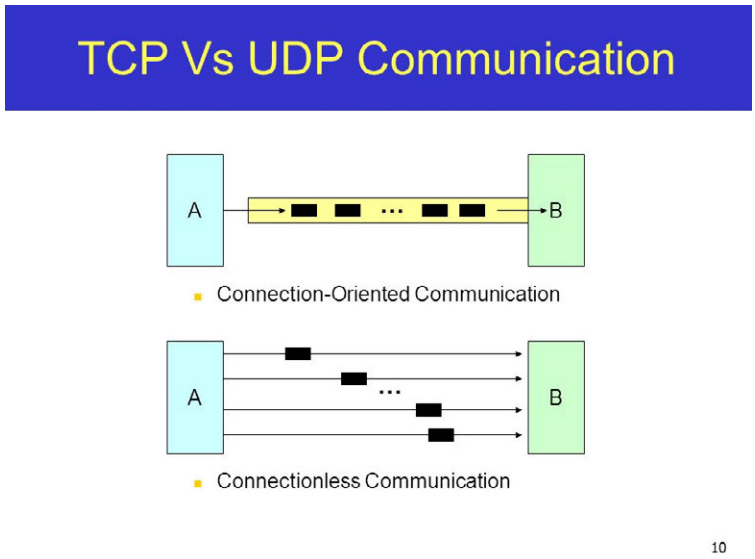


Fig. 48: Diferencias entre comunicación TCP y UDP [Sli,www].

Para definir un socket en GNU/Linux, lo primero que se debe hacer es definir una variable del tipo *struct sockaddr_in* en la que se define la dirección y el puerto en el que escuchará el servidor. Esta estructura contiene tres campos que deben inicializarse antes de enlazar el socket y esperar conexiones:

- *sin_family*: Este campo identifica la familia o formato a la que pertenece el socket. Debe contener el valor AF_INET que indica que el formato pertenece al espacio de nombres de Internet.
- *sin_addr*: Aquí se indica la dirección de Internet del equipo en el que se encuentra el servidor. En este caso el valor será INADDR_ANY que indica que la dirección será cualquiera. Este valor es el usado cuando se desea aceptar conexiones a través de Internet.
- *sin_port*: Este campo indica el numero de puerto en el que escuchará el servidor.

```
server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = INADDR_ANY;
server_addr.sin_port = htons(PORT);
```

Fig. 49: Inicialización de la dirección del servidor.

La función que proporciona la biblioteca para la creación de sockets es la primitiva *int* ***socket*** *(int namespace, int style, int protocol)*. El primer parámetro, *namespace*, indica el espacio de nombres, en este caso AF_INET (o PF_INET) para indicar que se trata del espacio de nombres Internet. En segundo lugar *style* indica el tipo de socket que se va a usar, es decir

el valor debe ser o bien SOCK_STREAM para utilizar una comunicación TCP, o SOCK_DGRAM si va a usarse UDP. Por último el parámetro *protocol* indica el protocolo de bajo nivel que se usará en la comunicación y debe ser el mismo para los dos puntos, el valor dado suele ser 0 para usar el protocolo por defecto. El valor devuelto por la función es el descriptor de fichero del socket, esto es, un número entero que identifica al socket creado y que sirve para trabajar con él.

```
int s = socket(AF_INET, SOCK_STREAM, 0);
```

Fig. 50: Creación del socket.

Para que sea posible la comunicación TCP es necesario que, una vez creado el socket, se le asigne una dirección (del tipo de datos visto anteriormente) en la que el socket espere conexiones entrantes. La función utilizada para llevar a cabo esto es la función *int* ***bind*** *(int socket, struct sockaddr *addr, socklen_t length)*. El primer parámetro es el descriptor de fichero del socket que ha devuelto la función *socket* después de crearlo. El segundo, *addr*, es un puntero a una variable del tipo estructurado *struct sockaddr*. Este tipo de datos actúa como tipo neutro entre las diferentes estructuras usadas para representar direcciones, ya que el formato de éstas depende del espacio de nombres en el que se trabaje, es decir, las funciones que trabajen con parámetros del tipo *struct sockaddr* aceptarán cualquier tipo de direcciones definidas en *sys/socket.h*. Por último el parámetro *length* indica el tamaño de la estructura que almacena la dirección. El tipo *socklen_t* representa un entero sin signo de al menos 32 bits de tamaño. Este tipo de datos es necesario en sistemas en los que el estándar POSIX utiliza el tipo *size_t* para definir tamaños.

```
bind(s, (struct sockaddr *)&server_addr, sizeof(server_addr));
```

Fig. 51: Asignación de dirección al socket.

Tras asignar la dirección al socket lo siguiente que debe hacerse es utilizar la función *int* ***listen*** *(int socket, unsigned int n)* para que el socket escuche conexiones entrantes. Esta función devuelve 0 si todo ha ido como debería y -1 si ha habido algún fallo. El primer parámetro, *socket*, es el descriptor de fichero que hace referencia al socket que va a escuchar las conexiones y el segundo, *n*, es un entero sin signo que indica el número de conexiones pendientes que puede haber.

```
listen(s, 2);
```

Fig. 52: Espera de conexiones.

El siguiente paso es aceptar la primera de las conexiones entrantes que haya en la cola. La función encargada de esto es *int* ***accept*** *(int socket, struct sockaddr *addr, socklen_t *length_ptr)* que tras aceptar la conexión, crea un nuevo socket que queda conectado y devuelve el descriptor de fichero de ese nuevo socket. Esto es debido a que el socket servidor no podría aceptar mas conexiones si quedase bloqueado en la conexión creada. El primer parámetro que acepta la función es el descriptor del socket servidor que está escuchando conexiones. El segundo, *addr*, es un puntero a una variable del tipo estructurado usado para representar direcciones y que recogerá la dirección del cliente que se conecte. Por último, *length_ptr* será un puntero a una variable de tipo *socklen_t* en el que se almacenará el tamaño del área de memoria que ocupa la dirección *addr*.

```
s_client = accept(s, (struct sockaddr *) &client_addr, &clilen);
```

Fig. 53: Aceptación de conexión.

Tras estos pasos, si se ha iniciado la conexión desde el cliente, ya sería posible la comunicación entre cliente y servidor. Para enviar datos al cliente se utiliza la función *ssize_t* ***write*** *(int fd, const void *buf, size_t count)* que pertenece a la biblioteca *unistd.h*. Esta función se usa para escribir datos en ficheros pero, ya que en Linux los sockets se manejan mediante descriptores de fichero, también sirve para enviar un buffer de datos mediante una conexión TCP. La función devuelve el número de bytes que se han escrito, 0 si no se ha escrito nada y -1 si ha ocurrido un error. El primer parámetro es el descriptor de fichero que hace referencia al socket cliente resultante de aceptar la conexión. El segundo es un puntero al la variable que contiene los datos a transmitir, que puede ser de cualquier tipo ya que, en lenguaje C, un puntero de tipo *void* hace referencia a un puntero que no tiene un tipo determinado, es decir, un puntero "genérico". Como último parámetro se debe especificar el tamaño en bytes de la variable que contiene los datos.

```
write(s_client, buffer, i);
```

Fig. 54: Envío de datos sobre TCP.

Para cerrar la conexión se utiliza la función *int* ***close*** *(int filedes)* que cierra el fichero cuyo descriptor es *filedes*. Esta función, al igual que la función *write*, no pertenece al ámbito de los sockets pero, al ser éstos tratados por el sistema operativo de forma semejante a un fichero, la función se aplica para cerrar conexiones.

```
close(s_client);
```

Fig. 55: Cierre de conexión.

Ejecución multihilo

Para intentar mejorar el rendimiento y la cantidad de imágenes por segundo que transmitía el sistema se optó por utilizar hilos (o *threads* en inglés) en la parte del envío de datos al ordenador. Un *thread* es la secuencia de instrucciones más pequeña que puede manejar el planificador del sistema operativo de forma independiente. Un proceso puede contener múltiples hilos que se ejecutan de forma concurrente y comparten recursos como la memoria. En GNU/Linux las funciones y constantes involucradas en la creación y manejo de hilos están definidas en la biblioteca *pthread.h*.

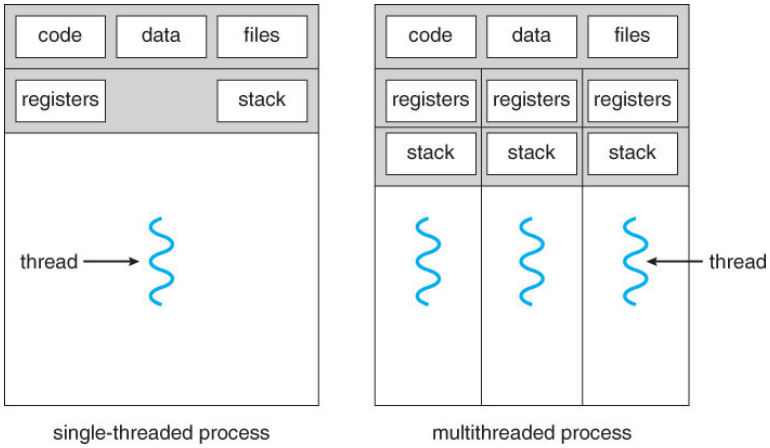


Fig. 56: Proceso monohilo vs. proceso multihilo [Uic,www].

La función que tiene como propósito crear hilos es *int* ***pthread_create*** *(pthread_t * thread, pthread_attr_t *attr, void * (*start_routine)(void *), void * arg)* y que devuelve 0 si el hilo se ha creado correctamente o un valor distinto si ha habido algún error. El primer parámetro necesario para llamar a la función es un puntero que recogerá, en la localización a la que apunta, el identificador del hilo creado. El siguiente, *attr*, especifica atributos que se le aplicarán al hilo creado y que puede ser *NULL* si no se desea aplicar ninguno. El tercer parámetro es un puntero genérico que apuntará a la función que ejecutará el nuevo hilo. Por último, *arg* recogerá el parámetro que se pasará a la función que va a realizar el hilo creado.

```
pthread_create(&thread, NULL, enviar_datos, &i);
```

Fig. 57: Creación del hilo.

La función *enviar_datos* creada se encarga de enviar al receptor los datos capturados por la cámara almacenados en un buffer de bytes. El parámetro que se pasa a la función, *i*, es un número entero que indica el número de bytes de información almacenados en el buffer.

Finalmente, al realizar pruebas con la versión monohilo y la versión con *threads,* se observó que el rendimiento era menor y había caídas de FPS en el vídeo recibido, por lo que se terminó utilizando una versión sin *threads*.

Pruebas de la versión fallida

En las versiones que se intentaron realizar con PICs de Microchip se empezó realizando una prueba básica en cada implementación en la que el chip fuese capaz de realizar una tarea sencilla como encender un led con un pin de E/S estándar. Con esta prueba lo que se intentaba era por un lado, probar si se podía programar el chip para comprobar que no fuese defectuoso o estuviese dañado, y por otro, la configuración básica necesaria para hacer funcionar un microcontrolador de Microchip. Para hacer funcionar un PIC hay que tener en cuenta señales, valores de registros y circuitería adicional necesaria como condensadores, resistencias y osciladores para que funcione correctamente.

La configuración del dispositivo se realiza modificando los valores de las palabras de configuración. Las palabras de configuración forman parte de la memoria de programa del microcontrolador y cada bit hace referencia a una opción de configuración disponible del dispositivo.

REGISTER 4-1: CONFIG1: CONFIGURATION WORD 1

R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	U-1
FCMEN(1)	IESO(1)	$\overline{\text{CLKOUTEN}}$	BOREN<1:0>(2)		—
bit 13					bit 8

R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1
$\overline{\text{CP}}$(3)	MCLRE	$\overline{\text{PWRTE}}$	WDTE<1:0>		FOSC<2:0>		
bit 7							bit 0

Legend:		
R = Readable bit	P = Programmable bit	U = Unimplemented bit, read as '1'
'0' = Bit is cleared	'1' = Bit is set	-n = Value when blank or after Bulk Erase

Fig. 58: Palabra de configuración del PIC 16F1508 [Mic3,www].

Los bis de configuración toman sus valores de directivas del compilador especificadas por el desarrollador de la aplicación. Las características que manejan estos bits son:

- Sincronización del sistema
- Gestión de energía
- Seguridad del dispositivo
- Características operativas

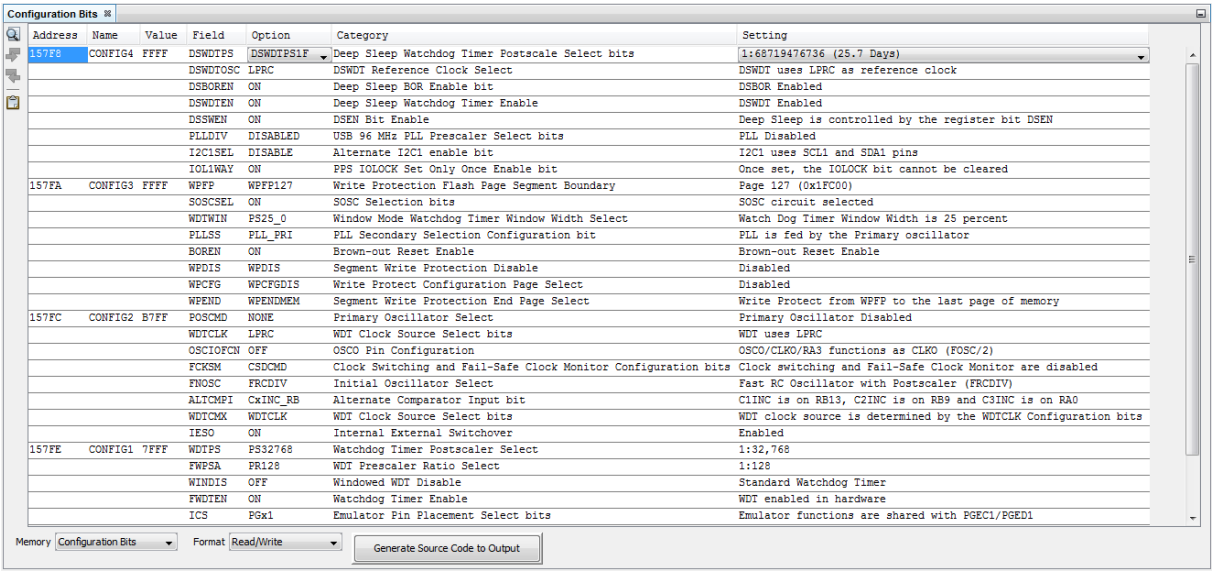


Fig. 59: Ventana de bits de configuración en MPLAB X

Para configurar estos bits, la forma más sencilla de hacerlo es mediante los complementos de configuración de MPLAB X como MPLAB Code Configurator [Mic14,www] para familias de 8 y 16 bits y MPLAB Harmony Configurator [Mic15,www] para chips de 32 bits. Estas herramientas permiten cambiar los bits de configuración de forma gráfica sin necesidad de conocer las directivas específicas del compilador necesarias para hacerlo de forma tradicional.

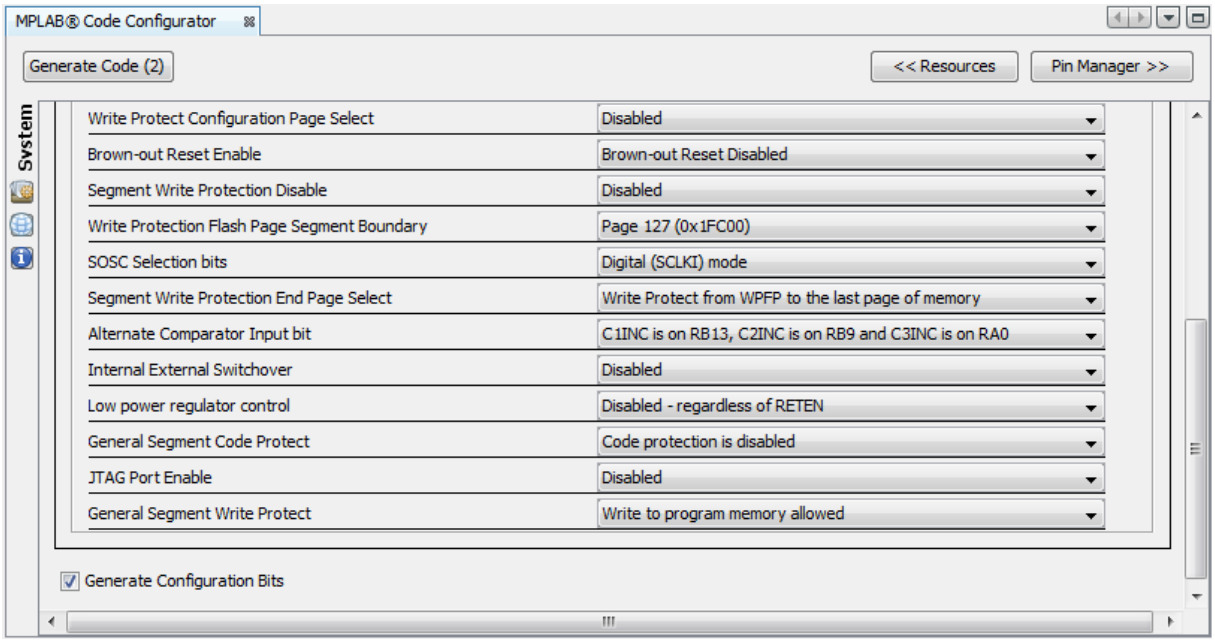


Fig. 60: Manejo de bits de configuración a través de MPLAB Code Configurator

Esta misma herramienta también sirve para configurar los distintos módulos que se deseen usar en la aplicación. Los módulos disponibles aparecen listados en una ventana y se añaden de forma sencilla al proyecto. Una vez añadidos, se tienen que configurar las características propias del componente y seleccionar los pines que realizarán funciones referentes a esos módulos (si se toma el ejemplo de una UART [Mbe,www], la velocidad de transmisión, los bits de paridad y de datos, y los pines TX y RX para transmisión y recepción de datos respectivamente).

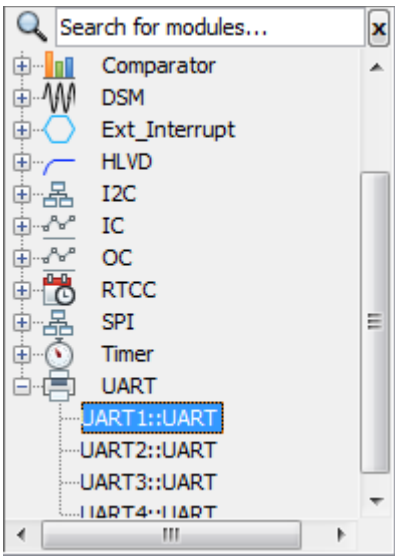


Fig. 61: Panel de selección de módulos.

Una vez se han seleccionado y configurado los módulos a usar y los bits de configuración, se debe pulsar el botón "Generate Code" en el que aparecerán entre paréntesis el número de ficheros del proyecto de MPLAB X que se verán afectados. Tras pulsar el botón por primera vez el programa pregunta si se desean crear automáticamente los ficheros necesarios, opción que resulta útil ya que separa la parte de los controladores generados por MPLAB Code Configurator de la parte en la que irán los fuentes referidos a el programa principal de la aplicación, y también los ficheros de cabeceras de la implementación.

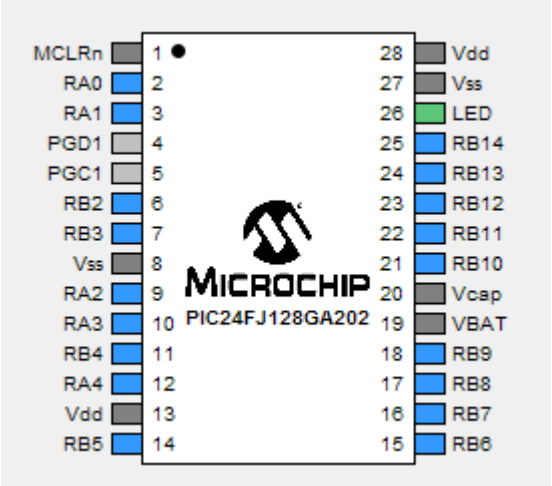


Fig. 62: Administrador de pines.

Como precaución se deben revisar los ficheros generados para comprobar que el código es el correcto, ya que el programa puede cometer errores y colocar de forma incorrecta alguna instrucción que provoque que el programa no funcione de forma debida. A este respecto cabe destacar que en la realización de una de las pruebas, en la que se hacía parpadear un led tras recibir una interrupción, se encontró un error en el código generado por la extensión MPLAB Code Configurator. Este error llevó a que el programa nunca realizara la función que se programó al no cambiarse el bit que indica que se ha tratado la interrupción y no poder volver a atender otra.

También debe tenerse en cuenta que, tras modificar las características de alguno de los módulos o añadir módulos nuevos, los cambios realizados de forma manual sobre los ficheros generados automáticamente son mostrados como código a suprimir por parte de la herramienta, por lo que se deberá prestar atención a la hora de seleccionar los cambios que quieren aplicarse sobre el código existente.

Tras conseguir realizar las pruebas básicas iniciales, el siguiente paso será configurar y conseguir comunicación a través del módulo UART. Una UART (*Universal Asynchronous Receiver/Transmitter* por sus siglas en inglés) es un tipo de dispositivo hardware que se utiliza para realizar comunicaciones en serie asíncronas y en la que el formato de datos y la velocidad de transmisión son configurables. La comunicación que realiza este tipo de dispositivos puede ser *símplex* (en una sola dirección, sin la posibilidad de que el dispositivo receptor envíe información al emisor), *semidúplex* (los dispositivos actúan como emisor o receptor por turnos) o *dúplex* (ambos dispositivos envían y reciben al mismo tiempo).

La comunicación vía UART era esencial en esa versión del proyecto ya que el módulo WiFi usado (RN171XV presentado en un capítulo anterior) recibe los datos a enviar vía comunicación serie compatible con una UART TTL. La conexión esquemática que puede verse en la figura 63 muestra cómo deben conectarse las señales entre el microcontrolador y el módulo WiFi. Las señales RTS y CTS son señales usadas para el control de flujo a nivel hardware, esto es, son señales que indican el comienzo y el final de la transmisión y sirven

para dotar de cierta sincronía y fiabilidad a la comunicación ya que estas señales son usadas por transmisor y receptor para indicar cuando están preparados para emitir/recibir información evitando así que se pierdan datos en la comunicación.

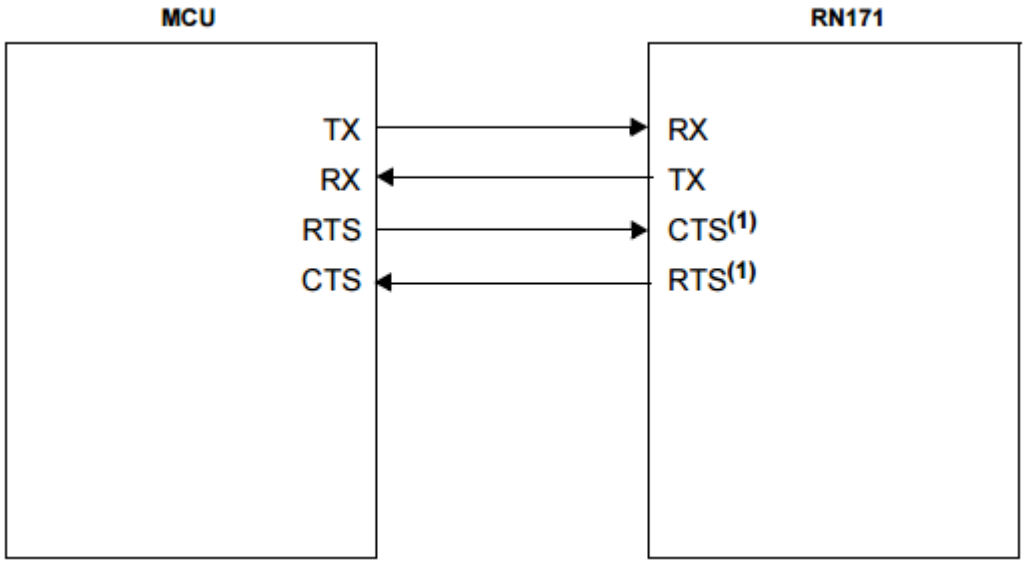


Fig. 63: Esquema de conexión entre el módulo RN171XV y el microcontrolador [Mic4,www].

En la figura 64 puede verse los pines que corresponden a las señales citadas anteriormente en el módulo RN171XV. Los pines correspondientes a la UART son TX, RX, RTS y CTS y sus respectivas funciones son las siguientes:

- RX: Este pin es la entrada de datos que deberá estar conectado al pin de transmisión del microcontrolador.
- TX: Por él saldrá el flujo de datos que recibirá el PIC. Es el pin transmisor.
- RTS: *Request to send* por sus siglas en inglés, es el pin por el que se enviará la señal preguntando si el canal está libre para realizar un envío de datos.
- CTS: *Clear to send* es el pin por el que se indicará que el canal está libre para la comunicación y actúa como respuesta a la señal anterior.

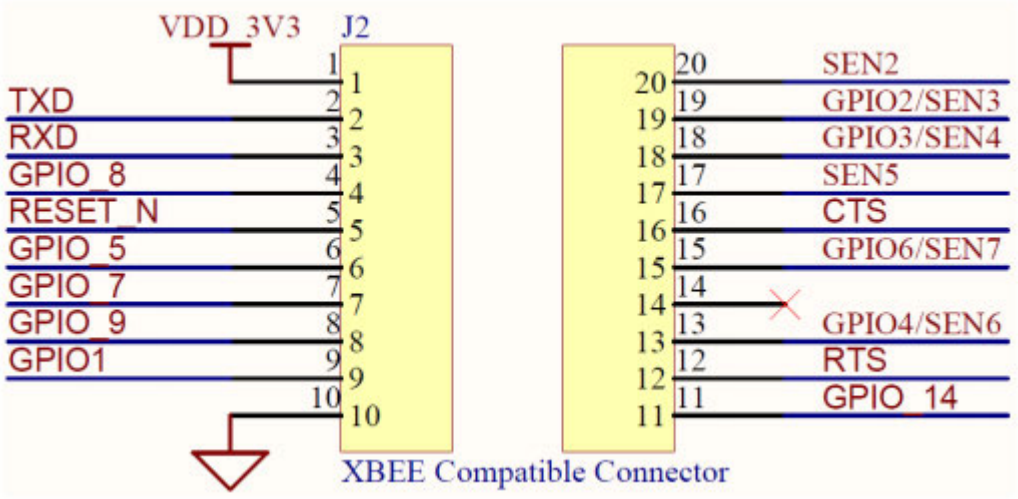


Fig. 64: Pines del módulo RN171XV [Mic5,www].

En la configuración por defecto del módulo WiFi no está habilitado el control de flujo por hardware, así como otras características de la conexión, por lo que será necesario cambiarlas para poder utilizarlo en el sistema. Para poder conectarse al módulo y enviarle los comandos de configuración de forma inalámbrica sin la necesidad de hardware adicional que actúe como interfaz entre el PC y el módulo RN171XV, es necesario que el módulo trabaje en el modo AP (como punto de acceso creando su propia red inalámbrica). Para habilitar este modo a nivel hardware se debe conectar, aparte de los pines de alimentación Vcc y GND, el pin GPIO 9 a la alimentación (3,3V) y reiniciar el módulo (basta con retirar la alimentación y volverlo a conectar). Tras esto la red que crea el módulo debe aparecer entre las conexiones de red inalámbricas cercanas y solo habría que conectarse como si de una red doméstica cualquiera se tratase. Una vez se ha conectado a la red, el siguiente paso será realizar una conexión vía Telnet [Wik,www] para poder enviar comandos al módulo.

Telnet es un protocolo de comunicación utilizado en Internet o en redes de área local que proporciona un canal de comunicación bidireccional interactivo para intercambio de texto utilizando para ello una conexión con un terminal virtual.

El cliente utilizado para realizar la conexión fue PuTTY, ya citado anteriormente, por lo que se obviará la explicación del mismo. El puerto en el que está escuchando el servidor Telnet es por defecto el puerto 2000 y la dirección IP es 1.2.3.4, aunque estos valores dependen de la versión del firmware que el módulo esté usando [Mic6,www]. Si la conexión se ha realizado de forma satisfactoria se mostrará un mensaje a modo de saludo en el terminal.

Para empezar a enviar comandos de configuración, el módulo debe entrar en "Modo Comando" para lo que se debe introducir la secuencia de caracteres "$$$", tras esto el módulo debe mostrar en el terminal "CMD" indicando que se ha entrado en modo comando (figura 65).

Fig. 65: Modo comando del módulo RN171XV.

Una vez dentro de este modo se pueden configurar muchos aspectos de la conexión, por ejemplo, las características de la UART. Para ver las opciones que están activas y sus valores se utiliza el comando *get uart* que devuelve el estado de los distintos atributos de configuración de la UART.

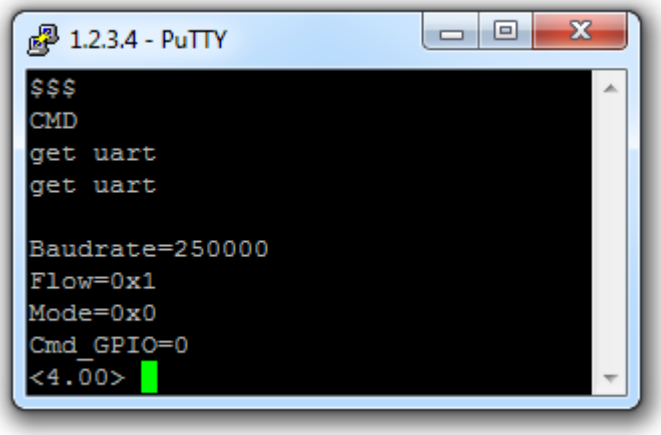


Fig. 66: Configuración de la UART.

En este caso, al estar ya configurado el módulo para ser usado dentro del sistema y poder realizar pruebas, los valores difieren a los que trae por defecto. La velocidad de transmisión por defecto que aparecería si no se hubiesen realizado cambios sería de 9600 baudios y, si se desea cambiarse, el comando que se debe usar es *set uart baud X* siendo X el valor en baudios de la nueva velocidad que podrá ser 2400, 4800, 9600, 19200, 38400, 57600, 115200 o 230400. Si se desea que el valor sea distinto de los citados anteriormente, el comando que se deberá usar es *set uart raw X* que se usa para establecer valores que no

sean estándar como en el ejemplo de la figura 66. El valor mínimo que puede darse es 2400 baudios. En la figura 67 se muestra una tabla con los valores que pueden usarse según el manual del fabricante, aunque se han probado otros valores distintos con éxito.

Raw Baud Rate	Comment
458,333	This is 460,800
500,000	Raw baud rate
550,000	Raw baud rate
611,111	Raw baud rate
687,599	Raw baud rate
785,714	Raw baud rate
916,667	This is 921,600
1,100,000	Raw baud rate

Fig. 67: Valores no estándar para la velocidad de transmisión [Mic6,www].

El siguiente valor indica que está habilitado el control de flujo por hardware. Para habilitarlo se debe cambiar el valor del registro que lo controla con el comando *set uart flow X* siendo X un número hexadecimal que siga el formato que aparece en la figura 68 para configurar el control de flujo hardware y la paridad.

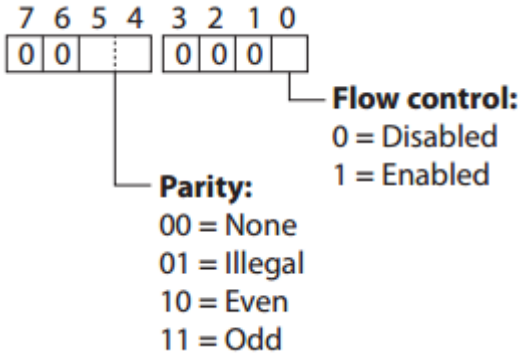


Fig. 68: Bits del registro de control [Mic6,www].

Teniendo en cuenta lo expuesto en la figura anterior, el valor que se usó fue 0x01 para no utilizar paridad y habilitar el control de flujo.

A continuación aparece el registro que controla los modos en los que funciona el módulo. Para cambiar el modo se utiliza el comando *set uart mode X* con el parámetro X como número hexadecimal que actúa como máscara con el formato que aparece en la figura 69. En la configuración usada la máscara del modo es 0x00 que es el valor por defecto. Esta máscara hace que el módulo realice las siguientes funciones:

- Habilita el eco de los comandos enviados por la UART.
- Deshabilita la creación de una conexión TCP que envíe los datos leídos de la UART.
- Deshabilita la opción de que el módulo se duerma tras la llegada de la señal de interrupción.
- Limpia el buffer cuando se cierra la conexión TCP.
- Muestra la versión del firmware.

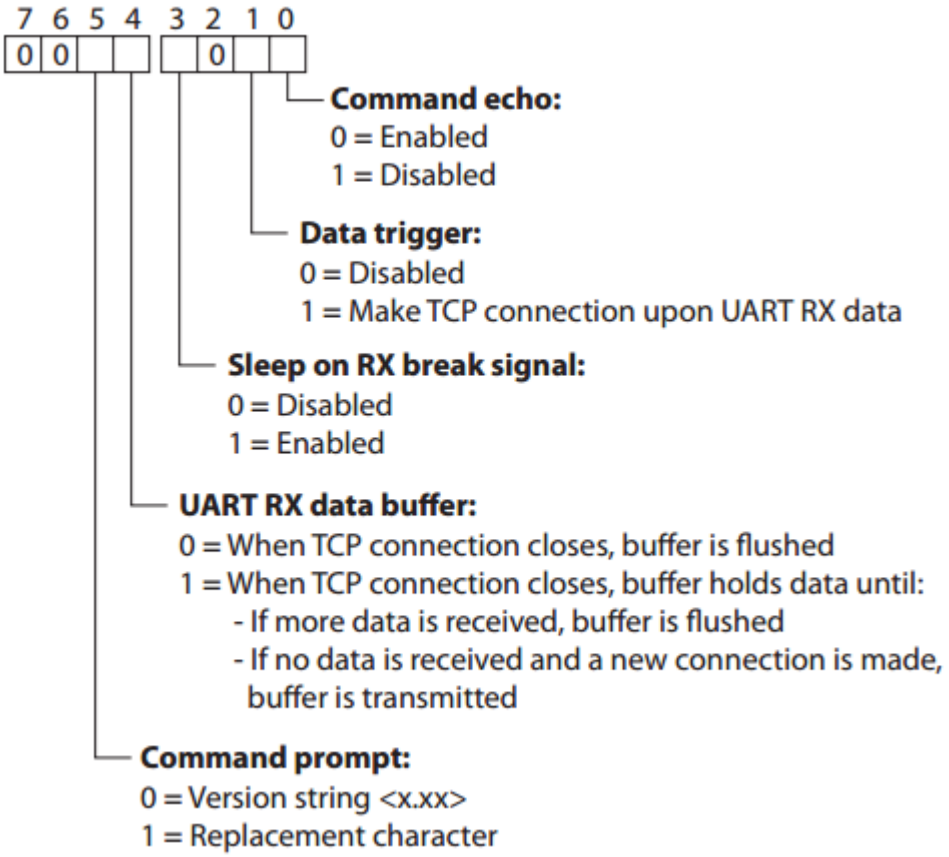


Fig. 69: Formato de la máscara del registro de modo [Mic6,www].

El último valor que aparece en la configuración de la UART sirve para indicar si está habilitado el pin TX de la UART. Para cambiar el valor se utiliza el comando *set uart tx X* con X como 0 para deshabilitarlo o 1 para habilitarlo. En este caso el valor que aparece es el valor por defecto.

En la parte del PIC la configuración de la UART se lleva a cabo por medio de las herramientas citadas anteriormente, MPLAB Code Configurator para PICs de 8 y 16 bits y MPLAB Harmony Configurator para PIC32 de 32 bits. Utilizando el primero de ellos se configuró una prueba básica de eco a través de la UART para el modelo PIC24FJ128GA202,

que consistía en recibir y repetir el carácter enviado a través de la conexión desde el PC. Para ello lo primero que debe hacerse es añadir el módulo correspondiente a la UART de la forma explicada cuando se presentó la herramienta. En la configuración de la UART del microcontrolador que puede verse en la figura 70 se observa que se optó por utilizar interrupciones para la recepción, el envío de datos y el manejo de los errores que pudiesen surgir. El uso de interrupciones puede complicar el código del driver del módulo pero mejora el rendimiento del programa ya que no se gastan recursos en hacer *polling* a los registros de recepción de datos de la UART. La velocidad de transmisión usada es la velocidad por defecto, 9600 baudios, ya que al ser una prueba sencilla no se necesitaban grandes velocidades y el error al calcular la velocidad, del que se hablará más adelante, es pequeño. Como puede verse, la opción de control de flujo está marcada en hardware para poder configurar los pines CTS y RTS y sean tenidos en cuenta por el driver.

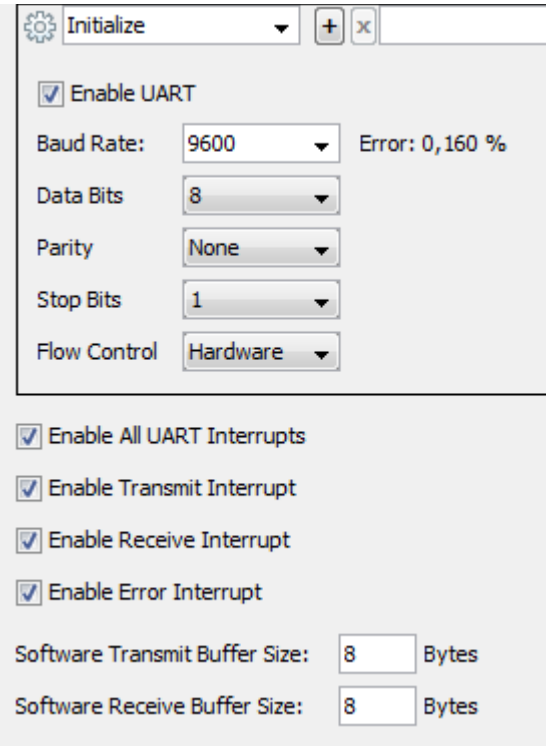


Fig. 70: Configuración de la UART.

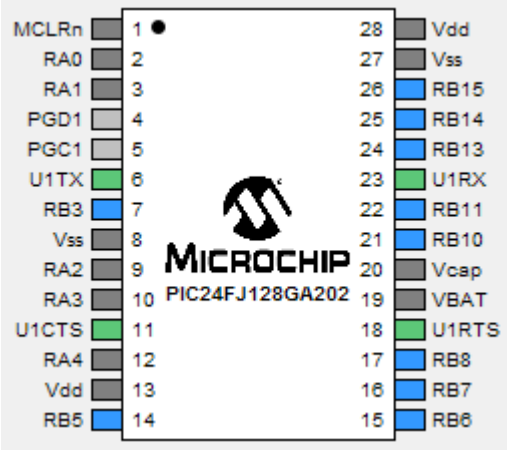


Fig. 71: Pines elegidos para cada función.

La velocidad de transmisión (baud rate) se calcula en función de la frecuencia del reloj del sistema y el valor de un registro en el que se almacena un valor constante que se utiliza para conseguir la velocidad deseada. Este valor será lo que modifique la herramienta MPLAB Code Configurator de forma automática para conseguir la velocidad que se le indique. Este cálculo depende también del modo de velocidad en el que esté trabajando el microprocesador ya que este modelo en concreto tenía dos modos de funcionamiento: normal y alta velocidad.

$$Baud\ Rate = \frac{FCY}{16 \bullet (UxBRG + 1)} \qquad Baud\ Rate = \frac{FCY}{4 \bullet (UxBRG + 1)}$$

$$UxBRG = \frac{FCY}{16 \bullet Baud\ Rate} - 1 \qquad UxBRG = \frac{FCY}{4 \bullet Baud\ Rate} - 1$$

Fig. 72: Cálculo de velocidad para modo normal (izquierda) y alta velocidad (derecha) [Mic7,www].

En la figura anterior, *Fcy* denota la frecuencia de un ciclo de instrucción, que es la frecuencia del oscilador del sistema dividida entre 2. Las ecuaciones de la parte inferior se utilizan para calcular el valor que tendrá el registro citado anteriormente que controla la velocidad por medio de un *timer* de 16 bits. El valor que contendrá el registro UxBRG será un valor entero por lo que el cálculo de la velocidad producirá un error si el resultado no es exacto. A continuación se explica con un ejemplo el problema:

En el ejemplo, la velocidad que se desea utilizar serán 9600 y la frecuencia de ciclo de instrucción (FCY) serán 4MHz.

Velocidad deseada = FCY/(16 (UxBRG + 1))

Se despeja UxBRG:

UxBRG = ((FCY/Velocidad deseada)/16) – 1

UxBRG = ((4000000/9600)/16) – 1

UxBRG = 25

Si se calcula con la fórmula de la velocidad en modo normal:

Velocidad calculada = 4000000/(16 (25 + 1))

= 9615

Error = (Velocidad calculada – velocidad deseada)/Velocidad deseada

= (9615 – 9600)/9600

= 0,16%

Cuanto más grande sea el porcentaje de error más probabilidades habrá de que ocurran errores en el envío o la recepción de la información. En las pruebas realizadas con velocidades superiores en las que aparecían porcentajes más grandes de error se daban los llamados *framing errors*, errores que suceden cuando la velocidad no es exactamente la deseada y ocurren desfases entre los extremos de la comunicación por lo que los bits se reciben en un orden incorrecto o se pierden. Para saber si se han producido estos errores se debe entrar en el modo comando del módulo WiFi y ejecutar el comando *show stats* que

muestra estadísticas de diversos aspectos del módulo, entre ellos el parámetro *RXerror* que hace referencia al número de bytes recibidos con errores (figura)

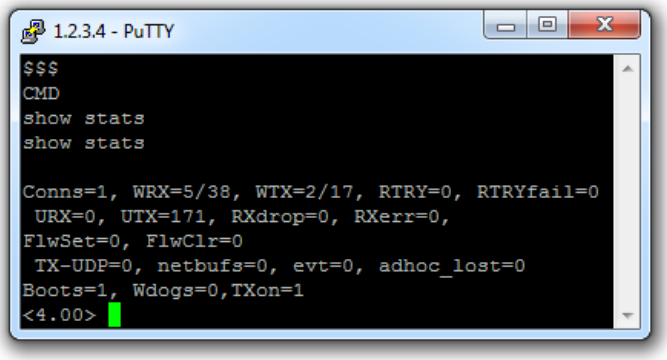


Fig. 73: Salida del comando que muestra estadísticas.

El siguiente paso una vez conseguida la comunicación vía UART fue añadir soporte para la comunicación mediante el bus SCCB, que es necesario para poder configurar las cámaras de OmniVision. Las herramientas de configuración de Microchip ofrecen la posibilidad de usar un driver I^2C que, como se dijo anteriormente, en teoría es compatible con las especificaciones del bus SCCB sin embargo, durante las pruebas fue imposible conseguir un resultado satisfactorio con ese driver por lo que se decidió implementar una versión "*bit bang*" [Dna,www], es decir, una versión en la que la funcionalidad del driver es controlada e implementada mediante software haciendo uso de las entradas y salidas de propósito general del microcontrolador.

El protocolo SCCB es un protocolo de comunicación serie que utiliza 3 señales para controlar las cámaras de OmniVision que actuarán como esclavo en la comunicación. Para encapsulados con pines reducidos el protocolo trabajará en un modo modificado de comunicación serie con 2 señales, que será el usado en la implementación realizada.

Las transmisiones de datos a través de este protocolo se dividen en fases. Cada fase contiene 9 bits, 8 de datos enviados de forma secuencial y un noveno bit que será un bit sin valor usado si existe más de un esclavo en el bus y que indica el fin de la transmisión. El número máximo de fases que se pueden incluir en una transmisión es tres y el bit más significativo siempre será el primero de cada fase.

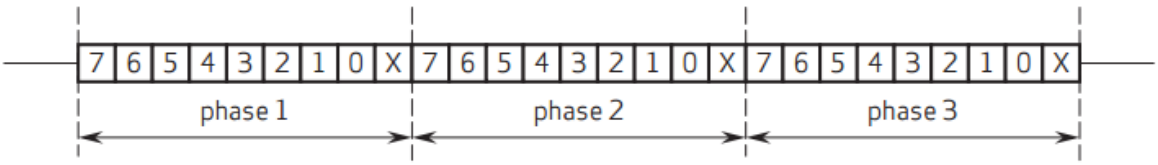


Fig. 74: Fases de una comunicación SCCB [Ovt,www].

- **Fase 1: Dirección del esclavo**

En la fase 1 el dispositivo que actúa como maestro en la comunicación envía la dirección del esclavo con el que quiere comunicarse. La dirección consta de 7 bits ordenados del bit 7 al bit 1 con los que puede identificarse hasta 128 esclavos. El octavo bit, el bit 0, es el selector de lectura/escritura con el que se especificará la dirección del ciclo de transmisión actual. Un 0 representa un ciclo de escritura y un 1 un ciclo de lectura.

- **Fase 2: Sub-dirección/Lectura de datos**

Tanto el maestro como el esclavo pueden realizar la fase 2 de la comunicación. La fase 2 iniciada por el maestro identifica la dirección del registro del esclavo al que se intenta acceder. Si la segunda fase ha sido iniciada por el esclavo indica los datos que el maestro va a recibir en la operación de lectura. El noveno bit es un bit NA cuando la fase 2 es realizada por el esclavo. El dispositivo maestro debe mantener un 1 lógico en la línea de datos (SIO_D) durante la transmisión del bit NA.

- **Fase 3: Escritura de datos**

La fase 3 solamente puede ser iniciada por el maestro. Esta fase contiene los datos que el maestro quiere escribir en el esclavo.

Hay tres tipos de transmisiones que pueden realizarse mediante el protocolo SCCB:

- **Ciclo de transmisión de escritura de 3 fases:**

El ciclo de transmisión de escritura de 3 fases es un ciclo de escritura completo en el que el maestro escribe un byte de datos en un esclavo específico. La dirección identifica al esclavo al que el maestro intenta acceder. La sub-dirección identifica el registro específico de ese esclavo en el que se escribirán los datos. Los datos de escritura consisten en 8 bits de datos que el maestro pretende sobrescribir en el contenido de esta dirección de destino específica. El noveno bit de cada fase será un bit sin valor.

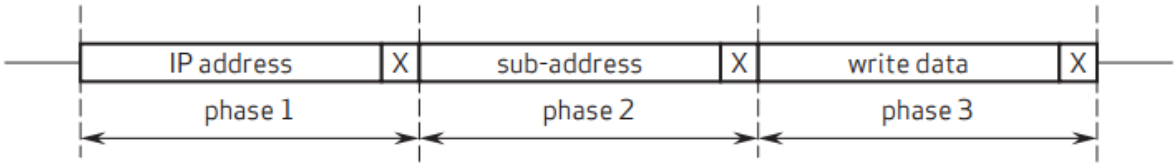


Fig. 75: Ciclo de transmisión de escritura de 3 fases [Ovt,www].

- **Ciclo de transmisión de escritura de 2 fases**

El ciclo de transmisión de escritura de dos fases va seguido de un ciclo de transmisión de lectura de 2 fases. El propósito de este ciclo de transmisión es identificar la dirección del registro del esclavo del que se pretende leer datos en el ciclo de lectura en dos fases que se realizará a continuación. El noveno bit de cada fase deberá ser un bit sin valor.

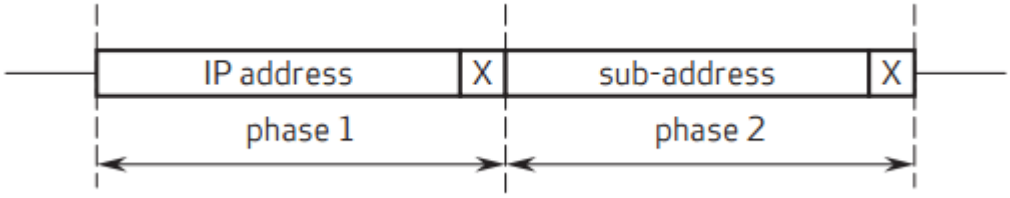


Fig. 76: Ciclo de transmisión de escritura de 2 fases [Ovt,www].

- **Ciclo de transmisión de lectura de 2 fases**

Debe haberse enviado un ciclo de transmisión de escritura de 2 o 3 fases antes de este ciclo ya que el ciclo de transmisión de lectura en 2 fases no tiene la capacidad de identificar la sub-dirección del esclavo en la que se intenta leer. Este ciclo contiene 8 bits de datos de lectura leídos del esclavo y un noveno bit que será un bit sin importancia o un bit NA. Si se trata de un bit NA, el dispositivo maestro será el encargado de enviar un 1 lógico en ese bit.

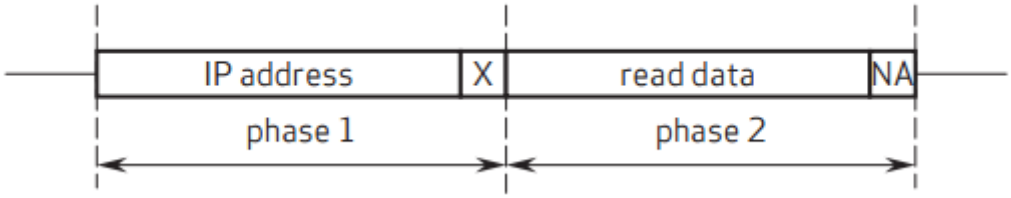


Fig. 77: Ciclo de transmisión de lectura de 2 fases [Ovt,www].

Estas fases especifican cómo deben realizarse los intercambios de información entre los dispositivos maestro y esclavo pero estos intercambios necesitan estar sincronizados mediante la señal SIO_C, es decir, la señal de reloj que sincroniza la comunicación, también llamada CKL en el protocolo I^2C. Esta señal es controlada por el dispositivo maestro, que debe mantener un 1 en la línea cuando es bus está inactivo. Un 1 lógico en la señal SIO_C durante una transmisión de datos indica un bit transmitido.

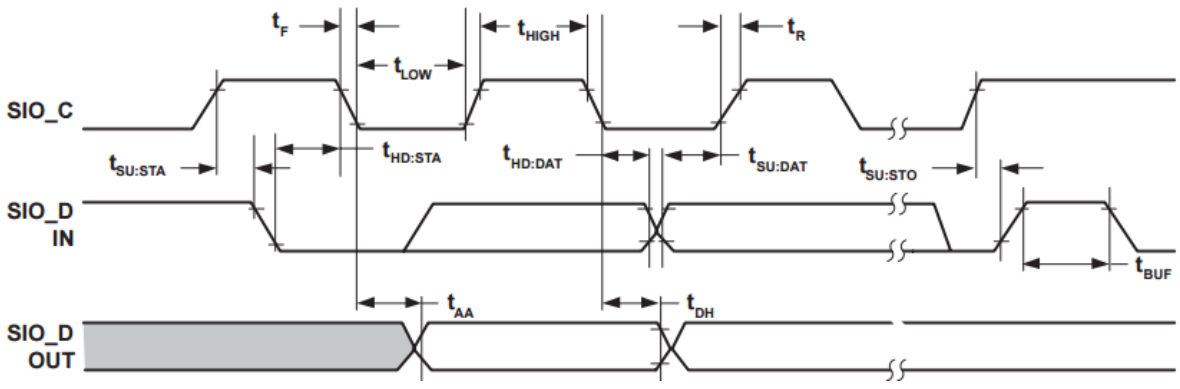


Fig. 78: Cronograma de comunicación SCCB [Vot,www].

En el cronograma de la figura 78 se observan los diferentes tiempos que se deben tener en cuenta a la hora de generar la señal SIO_C. Aunque parezca por la imagen que se trata de una señal de reloj simétrica, según el documento de especificaciones de la cámara el tiempo que la señal está a nivel alto es la mitad del tiempo a nivel bajo. A continuación se detallan los valores de los tiempos importantes que se tuvieron en cuenta al implementar la librería:

Símbolo	Parámetro	Valor
t_{LOW}	Periodo bajo del reloj	1,3 µs
t_{HIGH}	Periodo alto del reloj	0,6 µs
t_{BUF}	Tiempo de bus libre antes de nuevo comienzo	1,3 µs
$t_{SU:STA}$	Tiempo de preparación de comienzo	0,6 µs
$t_{HD:STA}$	Tiempo de espera para comienzo	0,6 µs
$t_{SU:DAT}$	Tiempo de preparación para recepción de datos	0,1 µs
$t_{SU:STO}$	Tiempo de preparación de final	0,6 µs

Tabla 5: Tiempos usados en la implementación de la señal SIO_C.

En el fichero de la implementación de la biblioteca, *bitbang_i2c.c*, que se encuentra disponible en el CD del proyecto se definen las constantes que representarán estos tiempos en las funciones que implementan el comportamiento de la señal SIO_C (figura 79).

```
#define T_HIGH 0.6L
#define T_LOW 1.3L
#define T_BUS_FREE 1.3L
#define T_START_SETUP 0.6L
#define T_START_HOLD 0.6L
#define T_DATA_IN_SETUP 0.1L
#define T_STOP_SETUP 0.6L
```

Fig. 79: Tiempos de la comunicación SCCB.

Al comienzo y al final de las operaciones de lectura y escritura se deben enviar secuencias preestablecidas de inicio y de final de transmisión. Estas secuencias deben ser enviadas por el dispositivo que actúa como maestro en la comunicación y sirven para indicar al esclavo que comienza o termina un intercambio y para evitar la propagación de estados desconocidos en el bus.

La secuencia de comienzo consiste en mantener ambas señales a nivel alto durante un tiempo determinado, cambiar la señal de datos a nivel bajo, esperar otro intervalo y cambiar a nivel bajo la señal de reloj como puede verse en la figura 80. Los tiempos que aparecen marcados se corresponden con los tiempos $t_{SU:STA}$ y $t_{HD:STA}$ de la tabla 5.

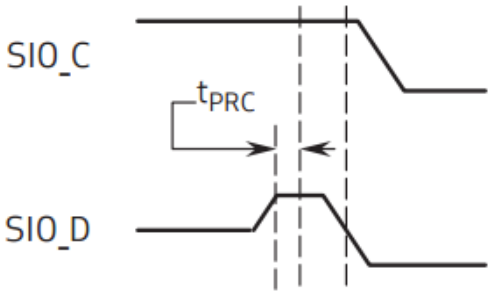


Fig. 80: Secuencia de comienzo de transmisión [Ovt,www].

La secuencia de final de transmisión se caracteriza por el paso de la señal de datos de nivel bajo a nivel alto, es decir, de forma inversa a la secuencia de inicio, de la forma mostrada en la figura 81. En esta secuencia los tiempos de la tabla 5 a tener en cuenta son $t_{SU:STO}$ y t_{BUF} .

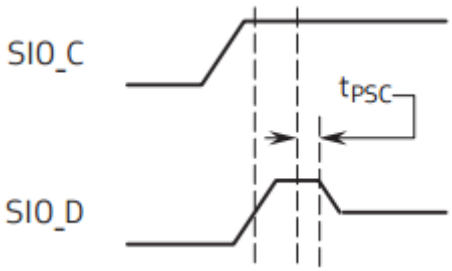


Fig. 81: Secuencia de final de transmisión [Ovt,www].

Para poder desarrollar la versión bitbang de la biblioteca es necesario hacer uso de las funciones relacionadas con las entradas y salidas estándar. Para ello se debe agregar el módulo correspondiente en la herramienta de configuración de Microchip. Para este caso solo será necesario configurar dos pines ya que la cámara utiliza la versión reducida del protocolo SCCB. Las funciones básicas del módulo están definidas como macros, las principales son (siendo *PIN* el nombre dado a la E/S con la que se desea realizar la función):

Macro	Función
PIN_SetDigitalOutput()	Convierte el pin en una salida digital
PIN_SetDigitalInput()	Convierte el pin en una entrada digital
PIN_SetHigh()	El valor de la salida será un nivel alto
PIN_SetLow	El valor de la salida será un nivel bajo
PIN_GetValue()	Devuelve el valor de la entrada

Tabla 6: Funciones para operar con los pines de E/S.

Con las funciones comentadas en la tabla anterior se implementaron las funciones básicas para manejar los pines de entrada/salida y poder dar los valores necesarios en las señales o leer la entrada de datos SIO_D en una operación de lectura.

```
bool Read_SDA (void)
{
    SDA_SetDigitalInput();
    return (SDA_GetValue());
}

void Clear_SDA (void)
{
    SDA_SetDigitalOutput();
    SDA_SetLow();
}

void Set_SDA (void)
{
    SDA_SetDigitalOutput();
    SDA_SetHigh();
}
```

```
bool Read_SCL (void)
{
    SCL_SetDigitalInput();
    return (SCL_GetValue());
}

void Clear_SCL (void)
{
    SCL_SetDigitalOutput();
    SCL_SetLow();
}

void Set_SCL (void)
{
    SCL_SetDigitalOutput();
    SCL_SetHigh();
}
```

Fig. 82: Funciones usadas para implementar la funcionalidad básica del bus SCCB.

También es necesaria una forma de hacer que el programa espere esos tiempos mencionados para que la señal SIO_C se comporte como se detalla en la hoja de especificaciones de la cámara. Para este cometido se usó la función *__delay_us(long microseconds)* que es una macro definida en la biblioteca *libpic30.h* que pertenece a las bibliotecas del compilador XC16 de Microchip. Para usar esta macro se debe definir una constante llamada *FCY* que contendrá el valor de la frecuencia del oscilador dividido entre dos, o sea, la frecuencia con la que ejecuta una instrucción el microcontrolador. El parámetro que acepta la función es un número al que la macro antes de pasarlo a la instrucción del compilador realiza un cast a un valor de tipo *long* por lo que a priori debería aceptar cualquier tipo de valores numéricos.

Utilizando esta función y las desarrolladas para dotar de funcionalidad básica al bus, se implementaron las funciones que generan las secuencias de inicio y fin de la comunicación comentadas anteriormente necesarias para poder iniciar y terminar transacciones de información. Ambas funciones reproducen los cronogramas mostrados en las figuras 80 y 81 realizando el paso de un nivel a otro con los tiempos explicados previamente.

```
void I2C_Start (void)
{
    Set_SDA();
     __delay_us(T_START_SETUP);
    Set_SCL();
    __delay_us(T_START_SETUP);
    Clear_SDA();
    __delay_us(T_START_HOLD);
    Clear_SCL();
    __delay_us(T_DATA_IN_SETUP);
}
```

```
void I2C_Stop (void)
{
    Clear_SDA();
    __delay_us(T_STOP_SETUP);
    Set_SCL();
    __delay_us(T_STOP_SETUP);
    Set_SDA();
    __delay_us(T_BUS_FREE);
}
```

Fig. 83: Implementación de las secuencias de inicio y fin de transacción.

El siguiente paso fue desarrollar funciones para la lectura y escritura de bits (figura 84) ya que estas operaciones se realizan haciendo uso de las dos señales del bus, SIO_C y SIO_D, y deben seguir las secuencias especificadas en el cronograma de la figura 78 haciendo uso de varios de los tiempos mencionados anteriormente. El propósito de estas funciones es utilizar la funcionalidad básica ya desarrollada para realizar operaciones de lectura o escritura de forma síncrona utilizando para ello la señal SIO_C. Estas funciones ayudan a reducir y simplificar el código de las funciones posteriores que realizarán lecturas y escrituras de bytes completos. La idea es desarrollar las funciones de la biblioteca de forma que las funciones más complejas hagan uso de funciones simples que ayuden a reutilizar código y a que sea más fácil entender el funcionamiento de la biblioteca.

```
void I2C_Write_Bit (const bool b)
{
    if (b)
    {
        Set_SDA();
    }
    else
    {
        Clear_SDA();
    }
    __delay_us(T_DATA_IN_SETUP);
    Set_SCL();
    __delay_us(T_HIGH);
    Clear_SCL();
    __delay_us(T_LOW);
}
```

```
bool I2C_Read_Bit (void)
{
    bool b;
    Set_SCL();
    Read_SDA();
    __delay_us(T_DATA_IN_SETUP);
    b = Read_SDA();
    __delay_us(T_HIGH);
    Clear_SCL();
    __delay_us(T_LOW);
    return (b);
}
```

Fig. 84: Funciones de lectura y escritura de bits.

En la función de escritura se observa que el parámetro que recibe es de tipo *bool*. Este tipo está definido en la biblioteca *stdbool.h*, que sigue el estándar C99 y debe ser incluida en el fichero de cabeceras de la biblioteca en desarrollo.

Fue necesario desarrollar también una versión de la función de escritura de bits para controlar el envío del bit NA al final de cada transacción de lectura en 2 fases (figura 85). Las diferencias son que esta función no tiene parámetros por lo que escribe valores fijos en el bus y realiza una espera adicional al final de las operaciones. Esta función es necesaria para mantener estable el bus tras una transacción de lectura en 2 fases.

```
void I2C_Send_NA (void)
{
    Set_SDA();
    __delay_us(T_DATA_IN_SETUP);
    Set_SCL();
    __delay_us(T_HIGH);
    Clear_SCL();
    __delay_us(T_LOW);
    Clear_SDA();
    __delay_us(T_DATA_IN_SETUP);
}
```

Fig. 85: Función de envío del bit NA.

Utilizando estas funciones de lectura/escritura de bits se desarrollaron a continuación las funciones que operan a nivel de bytes, que será la unidad de información con la que

trabajará el bus. Estas funciones realizan envíos o recepciones de bits en bucle utilizando para operar con ellos y formar bytes operadores de manejo de bits del lenguaje C.

```
void I2C_Write_Byte (const uint8_t byte)
{
    uint8_t byte_aux = byte;
    int i;
    for (i = 0; i < 8; i++)
    {
        I2C_Write_Bit((byte_aux & 0x80) != 0);
        byte_aux = byte_aux << 1;
    }
}

uint8_t I2C_Read_Byte (void)
{
    uint8_t resul = 0;
    int i;
    for (i = 0; i < 8; i++)
    {
        resul = (resul << 1) | I2C_Read_Bit();
    }
    return (resul);
}
```

Fig. 86: Funciones de envío y recepción de bytes.

Por último se desarrollaron las funciones que implementan los ciclos de transmisión explicados anteriormente. Estas funciones hacen uso de las mostradas en la figura 86 para enviar o recibir la información necesaria en cada caso, por ejemplo, las direcciones del esclavo y del registro que se desea acceder, el valor que se desea dar a un registro, leer el valor de un registro, etc. Al principio y al final de su ejecución deben llamar a las funciones que implementan las secuencias de inicio y final de transmisión para indicar al esclavo que se va a realizar un intercambio. Tras cada envío o recepción de un byte debe llamarse a la función de lectura de bits para recoger el bit sin valor que envía el esclavo para indicar el final de los datos.

```
void SCCB_Write_Data (const uint8_t address,
        const uint8_t reg, const uint8_t data)
{
    uint8_t address_aux = address;
    I2C_Start();
    address_aux = address_aux << 1;
    I2C_Write_Byte(address_aux);
    I2C_Read_Bit();
    I2C_Write_Byte(reg);
    I2C_Read_Bit();
    I2C_Write_Byte(data);
    I2C_Read_Bit();
    I2C_Stop();
}
```

Fig. 87: Función que realiza un ciclo de transmisión de escritura en 3 fases.

La función mostrada en la figura 87 realiza las distintas operaciones necesarias para ejecutar un ciclo de transmisión de escritura en 3 fases.

```
void SCCB_Write_Dir (const uint8_t address, const uint8_t reg)
{
    I2C_Start();
    uint8_t address_aux = address;
    address_aux = address_aux << 1;
    I2C_Write_Byte(address_aux);
    I2C_Read_Bit();
    I2C_Write_Byte(reg);
    I2C_Read_Bit();
    I2C_Stop();
}
```

Fig. 88: Función que implementa un ciclo de transmisión de escritura en 2 fases.

La función de la figura anterior implementa el ciclo de transmisión de escritura en 2 fases en el que se manda la dirección de un registro del esclavo.

Por último, la función de la figura 89 realiza las operaciones para ejecutar un ciclo de transmisión de lectura en 2 fases. Esta función devolverá un byte con la información enviada por el esclavo a través del bus.

```
uint8_t SCCB_Read (const uint8_t address, const uint8_t reg)
{
    uint8_t address_aux = address;
    SCCB_Write_Dir(address, reg);
    I2C_Start();
    address_aux = (address_aux << 1) | 0x01;
    I2C_Write_Byte(address_aux);
    I2C_Read_Bit();
    uint8_t resul = I2C_Read_Byte();
    I2C_Send_NA();
    I2C_Stop();
    return (resul);
}
```

Fig. 89: Función que realiza un ciclo de transmisión de lectura en 2 fases.

Con esta versión "*bit bang*" de la biblioteca I^2C fue posible lograr la comunicación necesaria para configurar las distintas opciones de la cámara de OmniVision. Una ventaja de realizar de esta forma la biblioteca fue que para portarla a la plataforma MPLAB Harmony solamente fue necesario definir las macros de manejo de pines y crear la función *__delay_us* que no está implementada en el compilador XC32.

Las macros definidas se basan en los nombres usados en las macros del compilador XC16 incluidas en los drivers de manejo de los pines de entrada/salida. Se renombraron de la forma mostrada en la figura 90 las funciones pertenecientes a la biblioteca de puertos incluida en las bibliotecas de periféricos de MPLAB Harmony (PLIB) para reducir el tamaño de las instrucciones, reutilizar el código existente de la versión para PIC24 y hacer más legible la biblioteca.

```
#define SDA_SetDigitalOutput() PLIB_PORTS_PinDirectionOutputSet
        (PORTS_ID_0, PORT_CHANNEL_B, PORTS_BIT_POS_7)

#define SCL_SetDigitalOutput() PLIB_PORTS_PinDirectionOutputSet
        (PORTS_ID_0, PORT_CHANNEL_B, PORTS_BIT_POS_6)

#define SDA_SetDigitalInput() PLIB_PORTS_PinDirectionInputSet
        (PORTS_ID_0, PORT_CHANNEL_B, PORTS_BIT_POS_7)

#define SCL_SetDigitalInput() PLIB_PORTS_PinDirectionInputSet
        (PORTS_ID_0, PORT_CHANNEL_B, PORTS_BIT_POS_6)

#define SDA_GetValue() PLIB_PORTS_PinGet
        (PORTS_ID_0, PORT_CHANNEL_B, PORTS_BIT_POS_7)
#define SCL_GetValue() PLIB_PORTS_PinGet
        (PORTS_ID_0, PORT_CHANNEL_B, PORTS_BIT_POS_6)

#define SDA_SetLow() PLIB_PORTS_PinClear
        (PORTS_ID_0, PORT_CHANNEL_B, PORTS_BIT_POS_7)

#define SCL_SetLow() PLIB_PORTS_PinClear
        (PORTS_ID_0, PORT_CHANNEL_B, PORTS_BIT_POS_6)

#define SDA_SetHigh() PLIB_PORTS_PinSet
        (PORTS_ID_0, PORT_CHANNEL_B, PORTS_BIT_POS_7)

#define SCL_SetHigh() PLIB_PORTS_PinSet
        (PORTS_ID_0, PORT_CHANNEL_B, PORTS_BIT_POS_6)
```

Fig. 90: Definición de las macros de control de pines.

La función usada para realizar esperas durante un tiempo determinado no se encontraba presente entre las directivas disponibles por lo que fue necesario implementarla para poder usar la biblioteca en la plataforma PIC32.

Para desarrollar la función se pensó en primer lugar en utilizar un *timer* añadiéndolo como periférico al proyecto pero se vio que no era necesario ya que en implementaciones de esta función realizadas para la familia de microcontroladores usada se realizaba el control del tiempo utilizando el *timer* del núcleo del sistema. Para acceder a este *timer* se utiliza la instrucción "mfc0" del set de instrucciones de ensamblador de la familia PIC32 que sirve para mover valores de los registros del coprocesador 0.

```
static float ReadCoreTimer()
{
    volatile float timer;

    // get the count reg
    asm volatile("mfc0   %0, $9" : "=r"(timer));

    return(timer);
}
```

Fig. 91: Función que devuelve el valor del timer del sistema.

Con esta función ya es posible controlar el tiempo transcurrido entre dos acciones pero es necesario utilizar la frecuencia del reloj del sistema para realizar el cálculo. Para realizar esperas basadas en valores de tiempo en microsegundos se debe calcular el tiempo de un ciclo como se indica en la figura 92.

```
#define FCK 100000000
#define CCLK_US (FCK/2000000)
```

Fig. 92: Cálculo del tiempo por ciclo.

Con todo lo anterior ya se puede desarrollar la función *__delay_us(float miliseconds)* para realizar esperas por un tiempo determinado. Para ello se usa la función *ReadCoreTimer* mostrada en la figura 91 al inicio del código de la función y se almacena el valor devuelto para luego compararlo con el tiempo actual y comprobar si ha transcurrido el intervalo deseado.

```
void __delay_us (float tiempo)
{
    float delay_count, start_time;
    start_time = ReadCoreTimer();
    delay_count = tiempo * CCLK_US;
    while ((ReadCoreTimer() - start_time) < delay_count)
    {
        _nop();
    }
}
```

Fig. 93: Función para realizar esperas en microsegundos.

El resto del código de la biblioteca portada a la plataforma PIC32 es idéntico al desarrollado para el microcontrolador de 16 bits explicado anteriormente salvo que se

añadieron instrucciones *_nop()* de no-operación después de los cambios en los pines para asegurarse de que se ejecuta la instrucción de cambio de dirección del pin.

Para desarrollar las pruebas en el modelo de 32 bits se usó el microcontrolador PIC32MZ2048ECH064 y la plataforma MPLAB Harmony. Para hacer más sencilla la configuración de los distintos módulos usados se utilizó la herramienta MPLAB Harmony Configurator que ayuda a gestionar de forma gráfica los distintos componentes del microcontrolador a usar y las distintas configuraciones aplicables a esos módulos.

Una parte de la herramienta que se diferencia bastante con respecto al MPLAB Code Configurator usado con microcontroladores de 8 y 16 bits, es el Diagrama del Reloj (Clock Diagram) con el que se pueden configurar todos los aspectos relacionados con el reloj del sistema y los relojes que afectan a los distintos módulos y buses que puede implementar el microcontrolador.

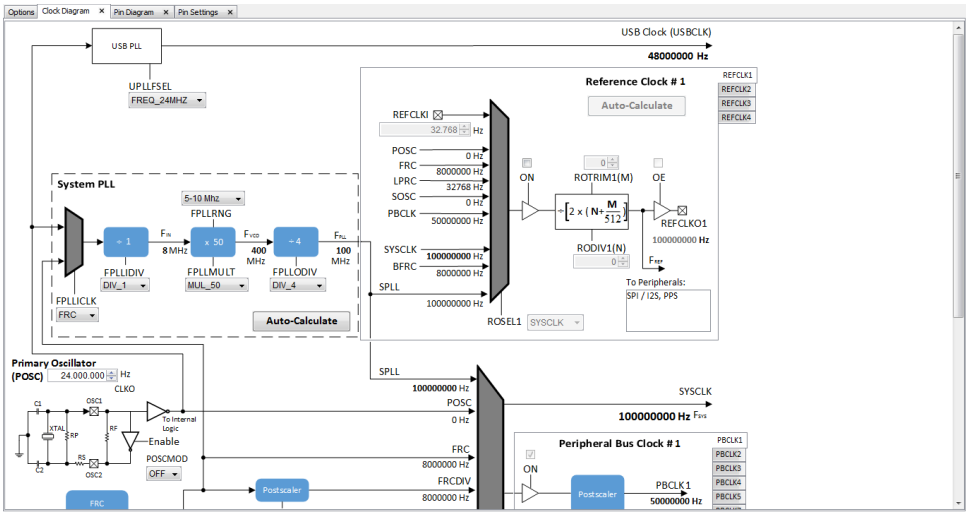


Fig. 94: Diagrama del Reloj del PIC32MZ2048ECH064.

En la figura 94 se observan parte de los aspectos relacionados con el sistema de reloj del microcontrolador. Este modelo de PIC cuenta con un PLL o circuito de de bucle de enganche de fase (*Phase Lock-Loop*) por sus siglas en inglés. Este circuito es un sistema realimentado cuyo objetivo principal consiste en la generación de una señal de salida con amplitud fija y frecuencia coincidente con la de entrada, dentro de un margen determinado [Upc,www].

Los valores de multiplicación y división de frecuencia usados en el PLL se configuran desde el Diagrama del Reloj de forma gráfica y los valores elegidos los asignará la herramienta a los campos del registro de control del PLL del sistema (*SPLLCON*) de la figura 95. Los valores elegidos deben generar una frecuencia de salida de 200 MHz como máximo en este modelo de PIC.

Register 42-3: SPLLCON: System PLL Control Register

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	U-0	U-0	U-0	U-0	U-0	R/W-y	R/W-y	R/W-y
	—	—	—	—	—	PLLODIV<2:0>		
23:16	U-0	R/W-y	R/W-y	R/W-y	R/W-y	R/W-y	R/W-y	R/W-y
	—	PLLMULT<6:0>						
15:8	U-0	U-0	U-0	U-0	U-0	R/W-y	R/W-y	R/W-y
	—					PLLIDIV<2:0>		
7:0	R/W-y	U-0	U-0	U-0	U-0	R/W-y	R/W-y	R/W-y
	PLLICLK	—	—	—	—	PLLRANGE<2:0>		

Fig. 95: Registro de control del PLL del sistema [Mic8,www].

Durante las pruebas se observó que el microcontrolador no soportaba trabajar a frecuencias cercanas a la frecuencia máxima por lo que se usó como máximo 100 MHz. Este valor representa la frecuencia del sistema, llamada *SYSCLK*, que será dividido para generar las diferentes señales de reloj necesarias para los periféricos que se usen, en este caso el que será de mayor importancia es la señal *PBCLK2* que es usada en el bus I^2C, en la UART, en el bus SPI y en el Puerto Maestro Paralelo (PMP). Para controlar la frecuencia de esta señal se debe asignar un valor al divisor *PB2DIV* que es un registro del oscilador que contiene un valor por el que se dividirá la frecuencia que salga de las operaciones realizadas por el PLL.

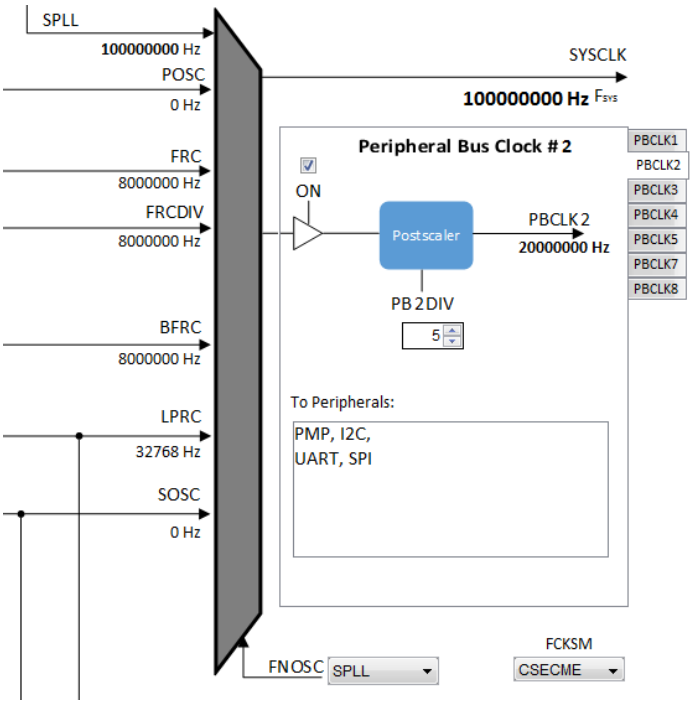


Fig. 96: Reloj para buses y periféricos 2 (PBCLK2).

En la figura 96 se observa que el divisor fue configurado para dividir la señal entre 5 por lo que la frecuencia de PBCLK2 es 20 MHz. Es importante tener en cuenta este valor ya que será necesario calcular la velocidad de transmisión de la UART. En el documento de referencia de la familia PIC32 específico de la UART aparecen las fórmulas para el cálculo de la velocidad así como diversas tablas con el error calculado en función de la velocidad de transmisión deseada y la frecuencia de PBCLK2. Las operaciones a realizar son similares a las explicadas anteriormente para el PIC24F y no es necesario realizar cálculos si se toman valores de las tablas que aparecen. En la figura 97 se pueden ver las funciones de cálculo usadas para generar las tablas mencionadas, en ellas F_{PB} hace referencia a la frecuencia de PBCLK2.

Para BRGH = 0

$$Baud\ Rate = \frac{F_{PB}}{16 \cdot (UxBRG + 1)}$$

$$UxBRG = \frac{F_{PB}}{16 \cdot Baud\ Rate} - 1$$

Para BRGH = 1

$$Baud\ Rate = \frac{F_{PB}}{4 \cdot (UxBRG + 1)}$$

$$UxBRG = \frac{F_{PB}}{4 \cdot Baud\ Rate} - 1$$

Fig. 97: Funciones para calcular la velocidad de transmisión [Mic9,www].

Como ejemplo, en la figura 98 aparece la tabla consultada para una de las pruebas. Se eligieron los valores, marcados en la imagen, de la frecuencia del bus y de la velocidad de transmisión en función del error ya que se estaba trabajando con velocidades de transmisión altas y cuanto menor fuese el error en el cálculo menos probable sería que ocurriesen errores de transmisión.

Peripheral Bus Clock: 20 MHz		
Actual Baud Rate	**% Error**	**BRG Value (decimal)**
110.0	0.0	11363.0
300.0	0.0	4166.0
1199.6	0.0	1041.0
2399.2	0.0	520.0
9615.4	0.2	129.0
19230.8	0.2	64.0
37878.8	-1.4	32.0
56818.2	1.5	21.0
113636.4	-1.2	10.0
250000.0	0.0	4.0
19	0.0	65535
1250000	0.0	0

Fig. 98: Tabla de valores de velocidad, error y generador de velocidad [Mic9,www].

Capítulo 6: Planificación y costes

En este apartado se detallará la planificación y organización del trabajo llevado a cabo y se desglosarán los costes de los componentes así como de su realización.

Planificación

La totalidad del proyecto se dividió en hitos que se debían cumplirse periódicamente. Algunos de estos hitos se fueron cumpliendo simultáneamente con otros a pesar de aparecer aquí ordenados como posteriores. Los hitos son:

Hito 1: Análisis y estudio de componentes y herramientas

- Composición de sistemas FPV
- Proyectos basados en Microchip PIC y Raspberry Pi
- Conexión entre el PC y el controlador (WiFi, Bluetooth, cable...)
- Cálculo de velocidad necesaria de conexión
- Desarrollo con MPLAB X

Hito 2: Elección de componentes

- Controlador (PIC, Raspberry Pi)
- Cámara
- Módulo WiFi

- Oscilador
- Alimentación
- Componentes varios (cables, condensadores, resistencias, LEDs, ...)

Hito 3: Compra de componentes

Hito 4: Montaje del prototipo

Hito 5: Configuración de herramientas y sistemas

Hito 6: Pruebas de componentes y control desde PIC/Raspberry Pi

- Comunicación Módulo WiFi – Controlador
- Comunicación Cámara – Controlador
- Configuración de la cámara

Hito 7: Implementación de software controlador

Hito 8: Comunicación PC – Controlador

- Pruebas con modo de red (Conexión local/red Ad Hoc)
- Pruebas con sockets (TCP/UDP)
- Implementación de software de captura y reproducción de vídeo

Hito 9: Pruebas de captura

- Captura de imagen y guardado local
- Captura de imagen y envío por WiFi
- Captura y envío de vídeo

Hito 10: Pruebas finales

Hito 11: Documentación

Los hitos mencionados se fueron cumpliendo periódicamente, no así los requisitos temporales unidos a éstos. Esto se ha debido a los diferentes problemas surgidos tanto en el desarrollo del software como en la configuración y comunicación con el hardware usado.

Hito	Tiempo estipulado	Tiempo real
Hito 1	30	60
Hito 2	30	50
Hito 3	10	20
Hito 4	10	10
Hito 5	20	30
Hito 6	30	50
Hito 7	50	80
Hito 8	15	20
Hito 9	10	8
Hito 10	20	30
Hito 11	50	85
Total	**275**	**443**

Tabla 7: Tiempos estimados y tiempos reales

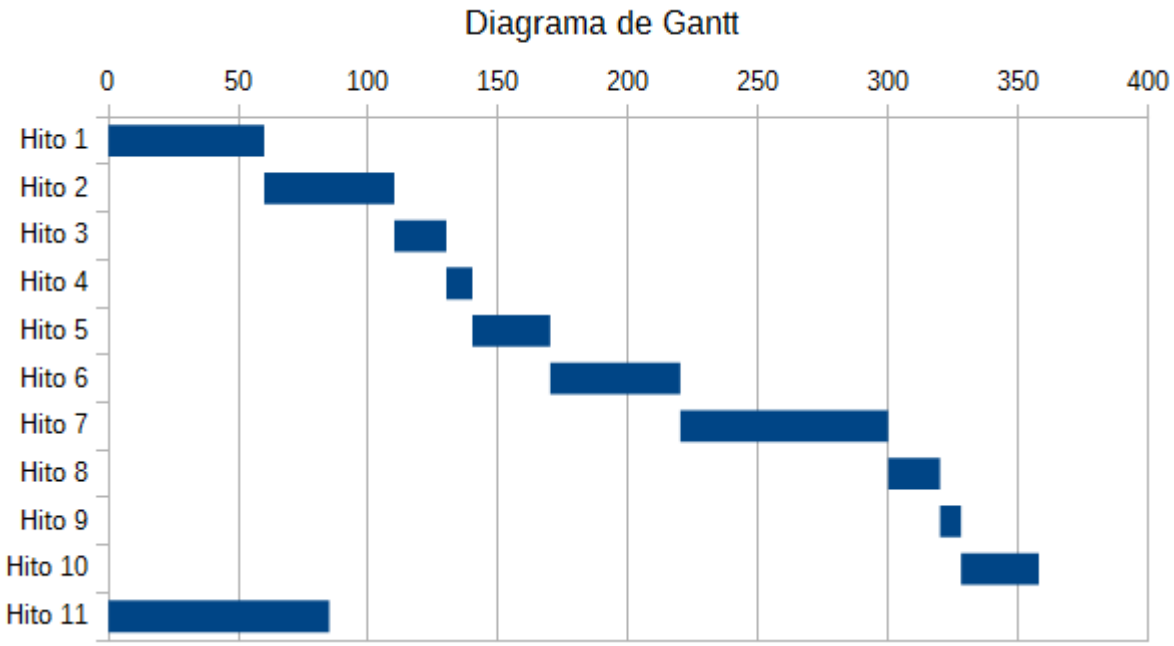


Fig. 99: Diagrama de Gantt.

Costes

Los costes del proyecto pueden dividirse en dos tipos: el coste de los materiales y componentes usados y las horas de trabajo dedicadas.

La siguiente tabla muestra los costes de los componentes:

Componente	Cantidad	Precio (€)
Módulo wifi RN171XV	1	45,44
PIC16F1508	1	1,29
PIC24FJ128GA202	1	3,44
PIC32MZ2048ECH064	1	10,17
Regulador de corriente	1	1,44
Cámara OV7670	1	6,09
Raspberry Pi	1	21
Arducam Mini 2MP	1	24,93
Cables	3 lotes	0,85/lote
Oscilador	1	1,69
Wifi dongle	1	11,90
Zócalo adaptador TQFP64	1	11,62
Pilas	15	2,90
Varios (condensadores, LEDs...)	ND	0,2/unid
Total	**144,46**	

Tabla 8: Costes de los componentes.

Para los costes derivados de las horas invertidas en la realización del proyecto, se tomó como sueldo medio de un Ingeniero Informático en España unos 1200 euros brutos al mes. El sueldo por hora derivado de la suposición anterior son unos 6,25€ por lo que según las horas indicadas en la tabla 7 el coste por horas invertidas sería de 2768,75 que sumados al coste de los componentes da un coste total de 2913,21.

Capítulo 7: Aspectos éticos, sociales y ambientales

En este capítulo se expondrán los diferentes aspectos que deben tenerse en cuenta para evaluar los distintos factores del proyecto que afecten tanto a la sociedad como al medio ambiente.

El proyecto desarrollado se sitúa en el sector tecnológico, concretamente en el ámbito de los sistemas empotrados, el desarrollo de controladores y el I+D+i, por lo que los impactos a tener en cuenta estarán relacionados con estas áreas. Se ha de tomar en consideración a la hora de identificar estos aspectos el ciclo de vida del proyecto, ya que dependiendo de la fase en la que se encuentre afectará de distinta forma a la sociedad o al entorno, por ejemplo, durante la fase de diseño no se verá afectado el medio ambiente de forma significativa pero una vez en producción se han de tener en cuenta los materiales usados en su desarrollo y los posibles residuos generados ya que eso sí puede tener consecuencias medioambientales.

En la siguiente tabla aparecen recogidas las ideas sobre los posibles aspectos afectados y problemáticas éticas a considerar durante las distintas fases de desarrollo del proyecto.

	Aspectos éticos	Aspectos sociales	Aspectos ambientales
Diseño, desarrollo software, producción y simulación de pruebas	No copiar software sin debida autorización.	Protección de datos (LOPD).	Eficiencia energética, teniendo en cuenta el consumo de energía. Uso de equipos aprovechables en vez de nuevos.
Implementación física del sistema	Reconocer mérito de contribuciones de terceros.	Prevención de riesgos laborales si los hubiese.	Evitar contaminación en la fase de implantación en base a las leyes medioambientales.
Uso y mantenimiento del sistema	Derechos de privacidad.	Obsolescencia programada.	Uso del bajo consumo si los equipos disponen de ello.
Reutilización, reciclaje o desecho	Derecho de propiedad intelectual.	Integración y aceptación social ante posibles innovaciones.	Reutilización de componentes utilizados en fase de prototipado.

Tabla 9: Aspectos éticos, sociales y medioambientales.

En cuanto a los aspectos éticos del proyecto, se han tomado en consideración las recomendaciones del director del proyecto a la hora de atribuir la autoría de información consultada o bibliotecas de software usadas. También referido a esto ha sido útil la normativa de la UPM para la realización de proyectos, ya que varios puntos van enfocados a ello.

El impacto social de un proyecto está estrechamente relacionado con la envergadura del mismo, por lo que, en este caso concreto, no resulta muy significativo a pesar de haber sido tenido en cuenta, ya que la magnitud del proyecto no deja de ser pequeña en comparación con los desarrollados por las grandes compañías tecnológicas. En lo referido a la problemática social, problemas como la obsolescencia programada son difíciles de prevenir ya que lo sufren los componentes usados y es complicado aumentar la vida útil del sistema.

Sobre los aspectos medioambientales cabe destacar que se ha intentado que todos los componentes y equipos usados en el desarrollo cumplan con la directiva RoHS (*Restriction of Hazardous Substances* por sus siglas en inglés). Esta directiva restringe el uso de seis materiales peligrosos en la fabricación de diversos tipos de equipos electrónicos y eléctricos. Está muy relacionada con la directiva de Residuos de Equipos Eléctricos y Electrónicos (*WEEE* por sus siglas en inglés).

Se suele mencionar a esta directiva como la directiva “libre de plomo” pero restringe el uso de otras 5 sustancias más:

- Plomo
- Mercurio
- Cadmio
- Cromo VI
- PBB
- PBDE

PBB y PBDE son sustancias retardantes de la combustión en plásticos.

Conclusiones

El proyecto presentado en el presente documento constituye un sistema completo de visión en primera persona capaz de capturar imágenes en directo y enviarlas a un dispositivo que actúe como receptor. Este objetivo se ha alcanzado siguiendo los pasos expuestos a lo largo de este documento. Como receptor se ha usado un portátil por comodidad a la hora de desarrollar el software de recepción.

Se analizaron las distintas posibilidades a tener en cuenta a la hora de desarrollar el proyecto y se consultaron las especificaciones y componentes de las implementaciones de este tipo de sistemas orientados al ocio, como los usados en los drones de competición, y se adecuaron para reducir el coste final del sistema. Por este motivo se usaron como plataformas sobre las que desarrollar el controlador Microchip PIC y Raspberry Pi, siendo esta última con la que se obtuvieron resultados satisfactorios.

Para la comunicación entre el controlador y el PC que actúa como receptor se optó por utilizar tecnología WiFi ya que a pesar de ser una opción poco contemplada a la hora de desarrollar este tipo de sistemas, ofrece gran versatilidad a la hora de desarrollar el software receptor y permite un amplio número de dispositivos que pueden utilizarse como receptores de la información (por ejemplo un smartphone, una smart TV, etc).

Como lenguaje de desarrollo del software de recepción se eligió Java por estar familiarizado con él y por tratarse de un lenguaje versátil, multiplataforma y libre en su mayor parte. Para la parte del controlador se utilizó el lenguaje C tanto en las versiones realizadas con PICs como en la final con la Raspberry Pi como controlador. Se optó por este lenguaje por ofrecer capacidades de bajo nivel con la sintaxis de un lenguaje de alto nivel.

Conclusiones

El desarrollo del proyecto se alargó más de lo esperado debido al diseño inicial en el que se utilizaba como controlador un PIC de Microchip con el que fue imposible obtener resultados satisfactorios. Esto se debió a la imposibilidad de configurar ninguna de ambas cámaras por lo que el comportamiento de la misma no era el deseado. A título personal, el alumno observó que trabajando con hardware, si no es posible obtener un resultado satisfactorio en un tiempo razonable es mejor optar por buscar una alternativa antes que obstinarse en seguir con lo mismo.

El proyecto resultante constituye una alternativa novedosa en lo referente a este tipo de sistemas y más teniendo en cuenta su modularidad y la capacidad de poder adaptarlo a otros componentes distintos.

Líneas futuras

Con respecto a los posibles desarrollos futuros basados en este proyecto existen varias líneas que pueden seguirse para continuar con el trabajo ya hecho.

En primer lugar sería interesante la posibilidad de utilizar otro tipo de receptor que también actuase como display de la información captada por la cámara, como por ejemplo un smartphone. Esto reduciría el volumen del sistema completo ya que evita tener que usar un PC como receptor y a su vez posibilita el uso de dispositivos de realidad virtual del tipo *cardboard* o Samsung Gear VR los cuales utilizan un smartphone como display.

Otra posibilidad sería la de mejorar el software de control y envío de información de la Raspberry para intentar mejorar la cantidad de frames por segundo enviados para que la señal de vídeo sea lo más fluida posible. Con una tasa de frames mayor se incrementaría la viabilidad del prototipo como alternativa a los sistemas de visión más extendidos.

Una tercera opción sería utilizar la señal de vídeo capturada para añadir información adicional sobre ella usando técnicas de Realidad Aumentada, con lo que por ejemplo podría disponerse en pantalla de un HUD (Heads-Up Display) que ofrezca más datos que ayuden al manejo del vehículo tales como su velocidad, altura (si se trata de un vehículo que vuele), porcentaje de batería o combustible, etc. También se podría utilizar Realidad Aumentada para generar componentes propios de un videojuego, como rivales para un juego de carreras o enemigos y obstáculos para un videojuego tipo arcade.

Anexos

Anexo A: Referencias

A

[Ada,www] http://www.adaic.org/

[Ali,www] https://ae01.alicdn.com/kf/HTB1efalMVXXXXXIXVXXq6xXFXXX6/TQFP64-LQFP64-QFP64-socket-adapter-IC-chip-test-seat-STM32-and-FPQ-64-0-5-06.jpg

[All,www] http://www.alliedelec.com/images/products/Small/70417120.jpg

[Ang,www] http://angryip.org/

[App,www] https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html

[Ard,www] http://www.arducam.com/wp-content/uploads/2015/03/ArduCAM_mini-1.jpg

[Ard2,www] http://www.arducam.com/downloads/shields/ArduCAM_Mini_2MP_Camera_Shield_Hardware_Application_Note.pdf

[Ard3,www] http://www.arducam.com/tag/arducam-mini/

[Ard4,www] https://www.arduino.cc/

B

[Bea,www] https://beagleboard.org/black

[Big,www] http://cdn6.bigcommerce.com/s-m8o52p/products/79/images/508/PZ0420M-L24-1__02400.1473413483.1280.1280.jpg?c=2

C

[Cam,www] http://dictionary.cambridge.org/dictionary/english/diy

[Cam2,www] http://www.hep.phy.cam.ac.uk/vnc_docs/index.html

[Chi,www] http://chipkit.net/

[Chw,www] http://static.betazeta.com/www.chw.net/up/2013/06/138-660x350.jpg

D

[Def,www] https://www.defense.gov/

[Dli,www] http://www.dlink.com/es/es/home-solutions/connect/adapters/-/media/images/products/dwa/121/dwa-121-left.png

[Dli2,www] http://www.dlink.com/es/es/-/media/consumer_products/dwa/dwa-121/datasheet/dwa_121_datasheet_en_deu.pdf

[Dro,www] http://dronesportsassociation.com/regulations/

[Dig,www] https://www.digi.com/products/xbee-rf-solutions/modules/xbee-802-15-4

[Dna,www] http://www.dnatechindia.com/Tutorial/8051-Tutorial/BIT-BANGING.html

E

[Ecl,www] https://eclipse.org/ide/images/desktop-ide-screenshot.jpg

[Ecl2,www] https://www.eclipse.org/mpc/

[Ecl3,www] https://www.eclipse.org/ide/

[Ele,www] http://www.electronicaestudio.com/i/f/sht001a.jpg

[Els,www] http://elstore.pl/946-thickbox_default/lf33cv-lm7803-3v3.jpg

[Eur,www] http://en.eurny-rc.fr/wp-content/uploads/sites/3/2014/09/FPV-Diagram.jpg

F

[Fpv,www] https://fpvracing.tv/builds/300

[Fut,www] http://www.futurlec.com/Pictures/Package/PIC32MX695F512L.jpg

G

[Gee,www] http://www.geekalerts.com/u/rc-vid-car.jpg

[Git,www] https://github.com/arducam

[Git2,www] https://github.com/ArduCAM/RaspberryPi

[Giz,www] http://gizmodo.com/robo-sallys-bomb-disposal-skills-will-blow-you-away-511048993

[Gnu,www] https://gcc.gnu.org/

[Gnu2,www] https://www.gnu.org/software/make/manual/make.html

[Gnu3,www] https://www.gnu.org/software/libc/

[Gol,www] https://golang.org/

[Gre,www] http://www.chiark.greenend.org.uk/~sgtatham/putty/team.html

H

[Har,www] http://hardwarefun.com/wp-content/uploads/sites/2/2013/09/arduino-raspberry-pi.jpg

I

[I2c,www] http://www.i2c-bus.org/

[Ibi,www] http://www.ibiblio.org/pub/languages/fortran/ch1-1.html

J

[Jav,www] https://www.java.com/es/

K

L

[Lao,www] http://fotos00.laopinioncoruna.es/fotos/noticias/318x200/2010-06-23_IMG_2010-06-16_01.30.55__3289054.jpg

[Lib,www] https://libreelec.tv/

M

[Mbe,www] http://www.mbedded.ninja/electronics/communication-protocols/uart-protocol

[Mct,www] http://www.mct.net/faq/spi.html

[Mic,www] http://www.microchip.com/_ImagedCopy/DM320006.png

[Mic2,www] http://www.microchip.com/_images/mplabx/mplabx_large23.jpg

[Mic3,www] http://ww1.microchip.com/downloads/en/DeviceDoc/40001609D.pdf

[Mic4,www] http://ww1.microchip.com/downloads/en/DeviceDoc/70005171B.pdf

[Mic5,www] http://ww1.microchip.com/downloads/en/DeviceDoc/rn-171-xv-ds-v1.04r.pdf

[Mic6,www] http://ww1.microchip.com/downloads/en/DeviceDoc/50002230B.pdf

[Mic7,www] http://ww1.microchip.com/downloads/en/DeviceDoc/30010038c.pdf

[Mic8,www] http://ww1.microchip.com/downloads/en/DeviceDoc/60001191G.pdf

[Mic9,www] http://ww1.microchip.com/downloads/en/DeviceDoc/61107G.pdf

[Mic10,www] http://www.microchip.com/

[Mic11,www] https://developer.microsoft.com/es-es/windows/iot/Downloads.htm

[Mic12,www] http://www.microchip.com/mplab/mplab-x-ide

[Mic13,www] http://www.microchip.com/mplab/mplab-harmony

[Mic14,www] http://www.microchip.com/mplab/mplab-code-configurator

[Mic15,www] http://www.microchip.com/mplab/mplab-harmony/mhc_mhgc

[Mit,www] http://web.mit.edu/rhel-doc/4/RH-DOCS/rhel-rg-es-4/ch-ssh.html

N

[Net,www] https://netbeans.org/

O

[Opt,www] http://optionsmegastore.com/2548-tm_thickbox_default/lenovo-g5080-156-inch-laptop-intel-core-i3-4005u-17ghz-4gb-ram-500gb-hdd.jpg

[Ora,www] https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html

[Ora2,www] https://www.oracle.com/es/index.html

[Osm,www] https://osmc.tv/

[Ovt,www] http://www.ovt.com/download_document.php?type=document&DID=63

[Ovt2,www] http://www.ovt.com/

P

[Pih,www] https://pihw.wordpress.com/guides/direct-network-connection/

[Pin,www] http://pinout.xyz/pinout/wiringpi

[Pin2,www] http://pinet.org.uk/

[Put,www] http://www.putty.org/

[Pyt,www] https://www.python.org/

Q

R

[Ras,www] http://www.raspberrypi-spy.co.uk/wp-content/uploads/2012/09/Raspberry-Pi-GPIO-Layout-Revision-1.png

[Ras2,www] http://www.raspberrypi-spy.co.uk/wp-content/uploads/2012/06/Raspberry-Pi-GPIO-Layout-Model-B-Plus-rotated-2700x900-1024x341.png

[Ras3,www] https://www.raspberrypi.org/documentation/remote-access/ssh/README.md

[Ras4,www] https://www.raspberrypi.org/documentation/remote-access/vnc/

[Ras5,www] https://www.raspberrypi.org/

[Ras6,www] https://www.raspbian.org/

[Ras7,www] https://www.raspberrypi.org/downloads/noobs/

[Rce,www] https://rcexplorer.se/educational/2009/09/fpv-starting-guide/

[Ris,www] https://www.riscosopen.org/content/downloads/raspberry-pi

[Rso,www] http://es.rs-online.com/largeimages/R7694144-02.jpg

[Rub,www] https://www.ruby-lang.org/es/

S

[Sli,www] http://images.slideplayer.com/8/2307206/slides/slide_10.jpg

[Sou,www] https://sourceforge.net/projects/win32diskimager/

[Spa,www] https://cdn.sparkfun.com//assets/parts/7/5/6/08276-01.jpg

[Sta,www] https://stallman.org/

[Stm,www] http://www.st.com/en/microcontrollers/stm32-32-bit-arm-cortex-mcus.html?querycriteria=productId=SC1169

T

[Tig,www] http://www.tightvnc.com/download.php

[Tin,www] http://www.ti.com/lit/ds/symlink/lm2937-3.3.pdf

U

[Ubu,www] https://ubuntu-mate.org/raspberry-pi/

[Ubu2,www] https://developer.ubuntu.com/core/get-started#snappy-raspi2

[Uic,www] https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/images/Chapter4/4_01_ThreadDiagram.jpg

[Upc,www] http://www.jcee.upc.es/JCEE2001/PDFs2001/pindado.pdf

V

[Ves,www] http://www.vesalia.de/pic/raspberrypirev2.jpg

[Vot,www] http://www.voti.nl/docs/OV7670.pdf

[Vrs,www] http://www.vrs.org.uk/virtual-reality-gear/glasses/

W

[Wik,www] https://en.wikipedia.org/wiki/Telnet

[Wir,www] http://wiringpi.com/

X

[Xil,www] https://www.xilinx.com/content/dam/xilinx/imgs/kits/dk-s6-embd-g.jpg

Y

Z

Anexo B: Bibliografía

[Rit,93] The Development of the C Language
Dennis M. Ritchie, Abril 1993
ISBN: 0897915704

[Str,97] The C++ Programing Language (3ª edición)
Bjarne Stroustrup, Junio 1997
ISBN: 0201889544