Universidad del País Vasco Euskal Herriko Unibertsitatea
VITORIA-GASTEIZKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE VITORIA-GASTEIZ

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TRABAJO FIN DE GRADO

COMPARATIVA DE IMPLEMENTACIÓN DE SISTEMAS EMPOTRADOS ENTRE DISTINTAS PLATAFORMAS

Alumno/Alumna: ORMAZABAL, ANGUIANO, XABIER

Director/Directora (1): GIL-GARCIA, LEIVA, JOSE MIGUEL

Curso: 2021-2022

Fecha: Martes, 14, junio, 2022

Resumen

El presente trabajo se centra en la realización de una comparativa entre diferentes plataformas para el desarrollo de aplicaciones empotradas. Para ello se he basado en dos plataformas ampliamente conocidas como son por un lado “Raspberry pi” junto con el lenguaje de programación “Python” y por otro lado una placa de desarrollo de un microcontrolador de 32 bits del fabricante “ST Microelectronics” y el lenguaje de programación C/C++.

La comparativa se llevará a cabo mediante la implementación de una misma aplicación en ambas plataformas. Las variables más importantes a tener en cuenta en esta comparativa serán las siguientes: el tiempo empleado en la implementación, el coste económico y las capacidades y limitaciones de la plataforma a la hora de desarrollar ciertas aplicaciones.

La aplicación escogida ha sido un tablero compuesto por unas luces de reacción para entrenamiento deportivo y cognitivo. Este tipo de entrenamiento es muy utilizado para la mejora de los tiempos de reacción y reflejos de las personas, por ello es conocido que los pilotos de coches de carreras y porteros de fútbol lo incluyen en sus planes de entrenamiento físico.

Consiste en un tablero en el que se dispone de unos módulos ubicados a diferentes alturas y posiciones en los que una vez comenzado el juego aleatoriamente se activa uno de ellos, encendiendo su luz correspondiente, y esperará a ser desactivado, ya sea pulsando un pulsador o por medio de un sensor de proximidad. En ese momento el sistema activará un nuevo módulo que realizará la misma operación hasta concluir el juego. La finalización del juego se puede determinar ya sea por un número entero de iteraciones ejecutadas o al cabo de un tiempo transcurrido.

Tras el desarrollo de esta aplicación en ambas plataformas se expondrán los datos obtenidos de las variables más importantes previamente mencionadas y se hará una valoración sobre que plataforma es la más adecuada en cada caso dependiendo de los requisitos de desarrollo y producto final.

Objetivos

El objetivo principal de este trabajo es determinar que tipo de plataforma puede ser más adecuada para la realización de proyectos en aplicaciones de sistemas empotrados. Teniendo en cuenta dos opciones de plataformas con diferentes niveles de abstracción, por un lado la plataforma de más bajo nivel que se trata de microcontrolador programado en C/C++, y por otro lado una de más alto nivel basada en Raspberry pi programada en Python.

Para establecer cual es la mejor opción se investigarán los costes, tiempos de realización de proyecto, capacidades que ofrece cada una de ellas y sus limitaciones durante la realización de un mismo proyecto para ambas.

Abstract

The present work focuses on the realization of a comparison between different platforms for the development of embedded applications. For this purpose, it has been chosen two widely known platforms such as on the one hand Raspberry pi together with the Python programming language and on the other hand a development board of a 32-bit microcontroller from the manufacturer ST Microelectronics, the STM32F769NI-DISC0 and the C/C++ programming language.

The comparison will be carried out by implementing the same application on both platforms. The most important variables to take into account in this comparison will be the following: the time spent on the implementation, the economic cost and the capabilities and limitations of the platform when developing the applications.

The chosen application has been a board made up of reaction lights designed for sports and cognitive training. This type of training is widely used to improve people's reaction time and reflexes, it is known that racing car drivers and soccer goalkeepers include it in their physical training plans.

It consists of a board in which we have some modules located at different heights and positions in which, once the game has started, one of them is randomly activated, turning on its corresponding light, and it will wait to be deactivated, either by pressing a button or by a proximity sensor. At that moment the system will activate a new module that will carry out the same operation until the end of the game. The completion of the game can be determined either by an integer number of executed iterations or after an elapsed time.

After the development of this application on both platforms, it will be presented the data obtained from the most important variables previously mentioned and it will be maken an assessment of which platform is the most appropriate in each case depending on the development requirements and the final product.

Objectives

The main objective of this work is to determine what type of platform may be more suitable for carrying out projects in embedded system applications. Taking into account two platform options with different levels of abstraction, on the one hand the lowest level platform that is a microcontroller programmed in C/C++, and on the other hand a higher level one based on Raspberry pi programmed in Python.

To establish which is the best option, the costs, project execution times, capabilities offered by each of them and their limitations during the execution of the same project for both will be investigated.

Agradecimientos

Primeramente, me gustaría agradecer a todas las personas , que han hecho posible el desarrollo del presente proyecto, así como por la atención y ayuda que me han prestado durante este periodo.

En segundo lugar, agradecer a mi entorno más cercano por la calidez, apoyo y motivación constante que he recibido por su parte durante toda mi formación académica y que tanto valor humano aportan a mi vida.

Debería, por supuesto, hacer especial mención a aquellos profesores y compañeros de carrera que, a lo largo de estos años, han fomentado el desarrollo de mi curiosidad, así como la formación que me han entregado y que finalmente me ha permitido crecer tanto profesional como personalmente.

Por último, agradezco profundamente a mi familia y en especial a mis padres, quienes han hecho posible que pueda formarme profesionalmente de la mejor manera, siendo un apoyo fundamental en esos momentos difíciles que todo alumno debe afrontar a lo largo de su carrera académica y vida personal.

Índice general

Índice de ilustraciones

Índice de tablas

Introducción

Un sistema empotrado (también conocido como “empotrado”, “incrustado” o “integrado”) es un sistema de computación diseñado para realizar funciones específicas, y cuyos componentes se encuentran integrados en una placa de circuito impreso. El procesamiento central del sistema se lleva a cabo gracias a un microcontrolador, es decir, un microprocesador que incluye además interfaces de entrada/salida, así como una memoria de tamaño reducido en el mismo chip.

Estos sistemas pueden ser programados directamente en el lenguaje ensamblador del microcontrolador o microprocesador o utilizando otros lenguajes como C o C++ mediante compiladores específicos.

Son diseñados generalmente para su utilización en tareas que impliquen una computación en tiempo real, pero también destacan otros casos como son Arduino y Raspberry Pi, cuyo fin está más orientado al diseño y desarrollo de aplicaciones y prototipos con sistemas empotrados desde entornos gráficos.

Podemos entender un microcontrolador como un computador dedicado. Cuando decimos que son computadores dedicados, nos referimos a la capacidad limitada que suelen tener. Son pequeños, con velocidad relativamente baja y con un diseño sencillo y ligero. En nuestro computador de casa tenemos el procesador por un lado, la RAM por otro, etc. En cambio, un microcontrolador es un único chip en el que se junta un procesador, una memoria RAM, una memoria ROM y otra serie de componentes (llamados periféricos) que serán útiles al programador como conversores ADC y DAC o entrada/salida en diferentes formatos.

Es por ello, que no están pensados para mantener una infraestructura de software muy extensa. La gran mayoría de veces los microcontroladores se programan directamente(bare-metal), prescindiendo de un sistema operativo integrado, aunque para aplicaciones más complejas se implementan soluciones con sistemas operativos. Esto es beneficioso porque hace que nuestro código sea modular, ayudando así a la facilidad de agregación de nuevas características y mantenimiento entre diferentes desarrolladores.

Esta capacidad limitada casi obliga a que haya una amplia gama de microcontroladores formados a partir de elementos variados (diferentes tamaños de RAM, diferentes procesadores, diferentes módulos de entrada/salida) según el uso que se le vaya a dar. Que exista esta diversidad nos permite utilizar el microcontrolador que mejor se adapte a las necesidades de nuestro proyecto, y es uno de los motivos por el que son tan populares en los sistemas empotrados.

A la hora de desarrollar la solución específica para la aplicación se utilizan los kits de desarrollo que proporcionan los respectivos fabricantes para diseñar los prototipos iniciales del proyecto.

Un kit de desarrollo es un elemento hardware que facilita las pruebas y la programación de otro componente hardware, ya sea un microcontrolador, un microprocesador, una FPGA, u otros. Normalmente son placas con el componente en cuestión que se quiere utilizar junto con varios elementos extra que facilitan la programación y el prototipado.

Su principal utilidad es servir como escenario de prácticas y aprendizaje a los ingenieros que más tarde tendrán que trabajar con el microcontrolador o microprocesador de una forma parecida. Ejemplos de kit de desarrollo son Arduino (basados en microcontrolador), Raspberry Pi (basado en microprocesador), o las placas de desarrollo de FPGAs.

Un FPGA o Field Programable Gate Array es un conjunto de circuitos integrados, que está pensado para entregarse al cliente sin configurarlo para que luego cada uno lo personalice y programe según la tarea que necesite que haga esa pieza, una vez esta ya se ha fabricado. Es como si fabricaran un procesador y luego lo configuraran en función de lo que se necesite, en vez de enfocarlo a una tarea o un uso específico durante el proceso de producción.

Los FPGAs suelen contener diferentes bloques de procesamiento en su interior, y diferentes conectores para poder configurar los diferentes bloques de lógica, pudiendo hacer operaciones complejas o pudiendo funcionar como simples puertas lógicas. También dependiendo del módulo en su interior podremos encontrar memoria, especialmente en los módulos de lógica más complejos y avanzados.

Estos chips empezaron como dispositivos de memoria de solo lectura (ROM) programables y procesadores de lógica, los cuales se podían programar en fábrica, pero tenían una gran limitación, y es que esos módulos de lógica estaban ya fijados a las puertas lógicas y no se podían luego personalizar físicamente, solo pudiendo personalizarse mediante la programación. Años más tarde llegaría la compañía Altera, la cual crearía el primer FPGA que permitía su reconfiguración, borrando la configuración anterior del dispositivo y permitiendo su reutilización.

Por su parte, una Raspberry Pi se considera un SBC, es decir, un Single-Board Computer. El concepto es muy parecido al de una placa de desarrollo, pero digamos que de características más extendidas.

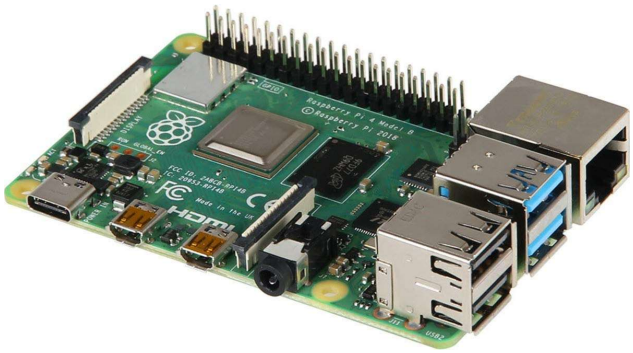

Ilustración 1 Raspberry pi 4B

Si un Arduino nos ayuda a programar microcontroladores que tendrán sólo un programa cargado (sin sistema operativo), poca memoria, poca velocidad, etc, una Raspberry Pi es un ordenador completo en una sola placa. Puede ejecutar una amplia gama de sistemas operativos convencionales de forma nativa como pueden ser distintas distribuciones de Linux, tienen más memoria RAM y almacenamiento. Suele utilizarse para realizar aplicaciones que necesitan más capacidad de cómputo o simplemente precisan tener un sistema operativo que le garantice determinadas librerías o paquetes.

Para comprender mejor qué son los sistemas empotrados, a continuación se listan sus características:

- Tienen un procesador central. Los módulos centrales de procesamiento pueden estar formados por microcontroladores, FPGAS, microprocesadores o una mezcla de los anteriores, principalmente.
- Son sistemas confiables.
- Requieren poco o nulo mantenimiento. En su gran mayoría no exigen mantenimiento dedicado.
- Son sistemas seguros. Tienen protección del software incrustado, utilizan protocolos encriptados, se diseñan para ser dispositivos Anti – Hacking. Aunque el gran auge de estos sistemas y su uso masivo y muchas veces descuidado está abriendo una brecha en su seguridad.
- Son eficientes en cuanto al consumo de energía.
- Son de propósito específico.
- Tienen interfaz de usuario simple o carecen de ella. Algunos de ellos cuentan con interfaz gráfica de usuario lo cual facilita su programación, configuración y/o control. Cabe resaltar que la interfaz de usuario no es de uso general (como lo pueden ser las interfaces de usuario de una computadora personal), sino, son de uso específico (como el teclado de un cajero automático, por ejemplo).

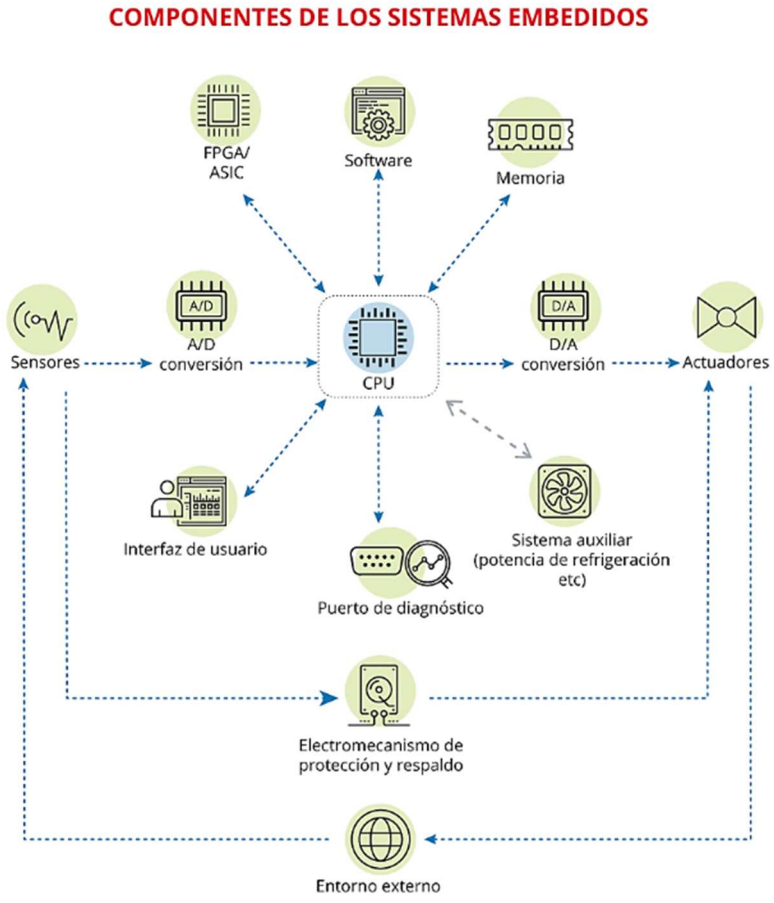

Ilustración 2 Componentes de los sistemas empotrados

En muchas aplicaciones es necesario mantener unos tiempos determinados para interactuar con el entorno de nuestro sistema empotrado, es por ello que se utiliza el término Sistema en Tiempo Real. Los Real-Time Systems son sistemas capaces de mantener medidas de tiempo estrictas y exactas, útiles para todos aquellos entornos en los que el tiempo de reacción sea crítico y la validez de los resultados obtenidos dependa también de si se han cumplido los plazos de tiempo esperados.

Por ejemplo, los sistemas de control que tienen casi todos los coches modernos se sustentan en sistemas de tiempo real. El airbag tiene unos marcos de tiempo muy concretos para activarse o deja de ser útil. Igual que la calibración del ABS, los frenos automáticos al detectar una posible colisión, o la corrección del rumbo al detectar una desviación. A todos estos factores hay que reaccionar rápido porque el tiempo de respuesta es crucial para la efectividad del sistema. Por ello se diseñan sistemas en tiempo real, que aseguren respetar y cumplir estas condiciones temporales con exactitud.

Cuando hablamos de sistemas empotrados, el software que controla el dispositivo hardware se conoce como Firmware. La programación puede realizarse bien directamente en el lenguaje ensamblador del microcontrolador o microprocesador, o bien en otros lenguajes como C, C++, JAVA, Python, etc. utilizando compiladores específicos.

Debido al aumento en la utilización de estos sistemas en una amplia variedad de sectores, existe esa gran cantidad de lenguajes de programación y plataformas disponibles para su desarrollo.

Como principales lenguajes de programación para microcontroladores nos podemos encontrar con dos tipos de lenguajes según como se comunican con el procesador: lenguajes compilados y lenguajes interpretados.

La diferencia entre lenguajes compilados e interpretados está contenida en la forma en que interactúan con el procesador. Es una noción común que los procesadores se expresan y entienden el código binario. En un lenguaje compilado el código escrito por el programador es convertido en ensamblador por el compilador y posteriormente ensamblado en lenguaje binario para que la procesador pueda ejecutarlo.

Sin embargo, el ensamblador es un código que varía según la familia de procesadores, por lo que es muy costoso en términos de tiempo y energía escribir un programa para cada código ensamblador. Por tanto, la solución pasa por escribir un programa de alto nivel que pueda ser universal. Sin embargo, el problema sigue siendo que para realizar cambios tendrás que acceder al código fuente, cambiar el necesario y volver a convertirlo.

El archivo así obtenido, sin embargo, será un archivo binario que en cada lanzamiento no necesitará ser convertido, luego se separan las fases de conversión y lanzamiento, y en ausencia de cambios la conversión se llevará a cabo solo una vez sin importar cuántas veces se ejecutará el programa.

En el caso de los lenguajes interpretados, el proceso de interpretación consiste en la traducción de un idioma a otro idioma como nos indica su definición, en este se da la traducción del idioma elegido al idioma binario para que el procesador pueda leerlo y ejecutarlo.

La diferencia con los lenguajes compilados radica en que el programa de un lenguaje compilado está representado por el archivo ya escrito en binario que, una vez lanzado, se lee directamente y se ejecuta. En los lenguajes interpretados, el programa está representado por el código fuente que primero se traduce y luego se compila cada vez, con la consiguiente disminución del rendimiento. Sin embargo, esto permite una mayor simplicidad en términos de sintaxis, pero sobre todo permite que el programa se ejecute en cualquier máquina con cualquier sistema operativo(es necesario que la plataforma sea compatible con el interprete).

Para poder programar los circuitos lógicos dentro de las FPGAs se utilizan lenguajes de descripción de hardware como lo son: VHDL, Verilog, ABEL HDL, etc. Pero en este documento no profundizaré en ellos.

En nuestro caso se han seleccionado los lenguajes C/C++ como lenguaje compilado y Python como lenguaje interpretado compatible con Raspberry pi.

C++ es un lenguaje de programación diseñado en 1979 por Bjarne Stroustrup. La intención de su creación fue extender al lenguaje de programación C mecanismos que permiten la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, C++ es un lenguaje híbrido.

Posteriormente se añadieron facilidades de programación genérica, que se sumaron a los paradigmas de programación estructurada y programación orientada a objetos. Por esto se suele decir que el C++ es un lenguaje de programación multiparadigma.

Python es un lenguaje de programación de alto nivel, orientado a objetos, con una semántica dinámica integrada, principalmente para el desarrollo web y de aplicaciones informáticas.

Es muy atractivo en el campo del Desarrollo Rápido de Aplicaciones (RAD) porque ofrece tipificación dinámica y opciones de encuadernación dinámicas.

Python es relativamente simple, por lo que es fácil de aprender, ya que requiere una sintaxis única que se centra en la legibilidad. Los desarrolladores pueden leer y traducir el código Python mucho más fácilmente que otros lenguajes.

Por tanto, esto reduce el costo de mantenimiento y de desarrollo del programa porque permite que los equipos trabajen en colaboración sin barreras significativas de lenguaje y experimentación.

Además, soporta el uso de módulos y paquetes, lo que significa que los programas pueden ser diseñados en un estilo modular y el código puede ser reutilizado en varios proyectos. Una vez se ha desarrollado un módulo o paquete, se puede escalar para su uso en otros proyectos, y es fácil de importar o exportar.

Por otro lado, uno de los beneficios más importantes de Python es que tanto la librería estándar como el intérprete están disponibles gratuitamente, tanto en forma binaria como en forma de fuente.

Tampoco hay exclusividad, ya que Python y todas las herramientas necesarias están disponibles en todas las plataformas principales. Por lo tanto, es una opción multiplataforma, bastante utilizada por los desarrolladores que no quieren preocuparse por pagar altos costos de desarrollo.

Es una tarea importante para el desarrollador la de seleccionar la plataforma y lenguaje que mejor se adapten a las características del proyecto, ya que el tiempo de desarrollo, el dinero invertido y las limitaciones del proyecto son consecuencias de esta decisión.

Debido a que se trata de la decisión más importante en este tipo de proyectos se ha querido plantear esta comparativa entre dos plataformas distintas. La primera ampliamente conocida en el mundo del software libre y que cuenta con una gran comunidad, la raspberry pi junto con Python, y la segunda es la plataforma sobre la que se basa la asignatura de sistemas empotrados impartida por la Universidad del País Vasco Sistemas Empotrados, la familia de microcontroladores STM32 concretamente la placa de desarrollo STM32F769I-DISC0 utilizando C/C++ como lenguaje.

Ilustración 3 STM32F769NI-DISC0

Para poder realizar una comparativa justa entre ambas partes se ha decidido implementar una aplicación con la que practicar los reflejos y el tiempo de reacción, puesto que era un proyecto que tenía pensado en ejecutar personalmente.

La aplicación consta de una estructura sobre la que colocar distintos módulos colocados estratégicamente en posiciones a las cuales el jugador deberá moverse. Cuando comienza el juego uno de los módulos se activará aleatoriamente y lo mostrará encendiendo su luz, entonces el jugador deberá desactivarlo acercando su mano al módulo donde el sensor de proximidad detectará que ese módulo es desactivado. De nuevo otro módulo se activará y el jugador tendrá que desactivarlo, este proceso se repite durante 20 iteraciones en este caso.

La idea es hacer un prototipo funcional que cumpla con las especificaciones previamente mencionadas y que disponga de una interfaz gráfica para que el usuario indique cuando comenzar el juego. Se ha intentado mantener los costes del material lo más reducidos posibles, ya que es un proyecto personal y sin ánimo de lucro. También se ha querido diseñar una estructura modular y que no sea muy pesada, para así poder transportarla de la forma más sencilla posible y ayudando a mantener un volumen reducido durante su conservación cuando no es utilizada.

El proyecto consta de dos partes bien diferenciadas. Por un lado, nos encontramos con una parte común que será igual para ambas plataformas. En la que se diseñará tanto la estructura como los módulos, se realizará la elección del material y se llevará a cabo la producción de la parte mecánica. Por otro lado, está la parte que nos interesa más en este trabajo y en la que profundizaremos más. Se trata de la parte que es propietaria de cada plataforma, en la que es necesario realizar las conexiones con los módulos, para lo cual deberá hacerse las adaptaciones necesarias, seleccionar y diseñar el material necesario y programar la parte correspondiente en el lenguaje de programación definido por el estudio.

Una vez concluido el desarrollo en sendas plataformas se realizará un estudio comparando las variables más importantes que se han mencionado al comienzo de este documento: el tiempo empleado en la implementación, el coste económico, la dificultad de desarrollo y las limitaciones de la plataforma a la hora de desarrollar la aplicación.

Motivación

Tras cursar la asignatura impartida por la Universidad del País Vasco en la carrera de Ingeniería electrónica industrial y automática llamada sistemas empotrados, me volví un apasionado de las infinitas posibilidades que ofrece esta tecnología.

Estos dispositivos permiten dotar de lógica a cualquier instrumento que podamos imaginar de nuestra vida cotidiana, aportando nuevas características en él o mejorando el control sobre el proceso para el que fue diseñado.

Teniendo en cuenta que además cada vez tiene más mercado y que su uso continuará aumentando, considero que es una buena opción especializarse en su desarrollo debido a que la demanda de personal cualificado en el sector no dejará de crecer.

El problema primordial que se afronta a la hora de desarrollar la aplicación es la elección de la plataforma en la cual el trabajo será llevado a cabo. Es por ello que como trabajo final de grado he pensado en realizar esta comparativa.

Material y presupuesto

El material que se ha utilizado para el desarrollo de este proyecto ha sido el que se ha considerado que puede dar unas prestaciones razonables manteniendo el coste dentro de unos márgenes asequibles.

A la hora de seleccionar el material para la estructura primeramente se pensó en el aluminio, puesto que es un metal con características muy apetecibles y que ofrece un peso de conjunto reducido y la estabilidad es correcta, pero su elevado precio obligó a pensar en una alternativa. La cual fue encontrada en los tubos de PVC destinados al uso hidráulico, se trata de un material muy liviano y económico que debido a su gran tamaño de mercado en el sector de la construcción para el uso en fontanería es fácilmente adquirible. La geometría con la que lo podemos encontrar en las ferreterías, tubos de diferentes diámetros, codos y tes para su ensamblado son óptimos para la realización de este proyecto.

Además como la estructura está compuesta por tubos, el cableado será fácilmente introducido por su interior, dando así un aspecto más ordenado.

Los módulos han sido impresos en una impresora 3D con material PLA. Constan de dos piezas: un cuerpo y una tapa. Esta tecnología para producir los módulos nos permite generar una geometría con unas tolerancias muy respetables, manteniendo el peso muy reducido y unos costes muy económicos.

Cada módulo tiene en su interior un hueco en el que se ha podido introducir la circuitería asociada al sensor de proximidad y LEDs. Su geometría es un prisma hexagonal para poder incluir de una forma correcta tramos de tiras de LED en su tapa, que es el estándar de venta de LEDs. En el interior se han colocado los sensores de proximidad pegados con unas escuadras de carpintería para mantener la orientación necesaria. La tapa dispone de un orificio que permite colocar el emisor y el receptor del sensor para interactuar con el exterior.

Los sensores escogidos han sido los módulos de sensor de tubo infrarrojo yyuezhi, un sensor muy económico que es ampliamente utilizado en aplicaciones de desarrollo de coste reducido. Sus prestaciones son correctas para aplicaciones que requieren detección de objetos manteniendo el coste del proyecto lo más limitado posible.

En cuanto a la tecnología utilizada para mantener la tapa y el cuerpo de los módulos ensamblados y también mantener los módulos anclados a la estructura se ha escogido el velcro. Es un material que permite adherir una cinta a cualquier superficie y esta se puede acoplar a otra superficie que tenga el complementario del velcro con otra cinta, la principal característica que ofrece este tipo de conexión es la facilidad de desmontaje.

Cada módulo necesita conectarse a cuatro pines del controlador que se utilice en cada momento, dos para alimentación, uno para la señal del sensor y otro para encender el LED. Es por eso por lo que se ha escogido un cable 4G0.25, un cable de 4 hilos en el que cada hilo tiene una sección de 0.25 mm^2.

Las conexiones que sean necesarias entre los controladores, los cables y los módulos se han realizado con los conectores JST-XH de cuatro pines, una opción muy económica de 4 pines generalmente utilizada para conexiones con PCBs.

Todo componente electrónico necesita ser alimentado para realizar la función para la cual está diseñado, es por eso que el proyecto necesita una alimentación. La idea es poder alimentar ambas plataformas con la misma fuente dado que son alimentadas a 5V. El componente escogido para esta labor ha sido un cargador de teléfono móvil que es capaz de suministrar 15W, más que suficiente en este proyecto.

La conexión a la Raspberry pi 400 se establecerá mediante un cable con conector tipo USB-C mientras que en el caso de STM32 se establecerá mediante un cable con conector microUSB.

Para el desarrollo en la plataforma de Raspberry pi se ha utilizado la placa Raspberry pi 400, los componentes son los mismos que los de una Raspberry Pi 4, pero en un formato de placa mejorado, con un enorme disipador pasivo que ocupa casi todo el teclado. Es un componente del que dispongo previo a la ejecución de este proyecto, pero se podría utilizar una Raspberry pi 4 reduciendo así el coste y el tamaño manteniendo todas las características necesarias de la aplicación.

Para la interfaz gráfica se ha escogido una pantalla táctil de 10,1 pulgadas compatible con esta plataforma. Una pantalla que permite interactuar al usuario con una calidad de imagen y tamaño muy confortable.

Como este tipo de placas no dispone de memoria no volátil integrada, el sistema operativo se ejecuta a través de almacenamiento externo. Lo más habitual es utilizar una tarjeta de memoria microSD como en este caso.

En el caso de la plataforma de microcontrolador se ha optado por la placa de desarrollo STM32F769NI-DISC0 cedida para el desarrollo de este estudio por parte del departamento de electrónica de la Universidad del País Vasco.

Esta placa es una completa plataforma para la demostración y desarrollo para el microcontrolador STM32F769NI, basado en el núcleo Arm® Cortex®-M7.

Permite generar una gran diversidad de aplicaciones obteniendo beneficios de características de audio, soporte multi sensor, gráficos, seguridad, vídeo y conectividad de alta velocidad.

Durante el desarrollo de la aplicación para esta segunda plataforma ha sido necesario el diseño y producción de una PCB compatible con el conector de ARDUINO® para la conexión con los conectores JST-XH. La fabricación se ha llevado a cabo por la empresa JLCPCB.

A continuación se mostrará en una tabla con el presupuesto de los materiales empleados para la realización de este proyecto:

Material	Unidades	Precio(unidad) €	Precio €
Tubo pvc-u w 40x1,9	3	4,75	14,25
Codo PVC C-40	2	1	2
Te PVCT-40	10	1,8	18
Yyuezhi Módulo Sensor Infrarrojo	8	1,05	8,4
Tira LED	1	9	9
Escuadra 90º	12	0,2	2,4
Bobina PLA impresión 3D (kg)	1	20	20
Manguera 4G0.25 (m)	20	1	20
Conector JST-XH 4 pin	40	0,1	4
Raspberry pi 400	1	80	80
HMTECH Monitor 1024 x 600 HDMI	1	100	100
Tarjeta de memoria microSDXC	1	12	12
STM32F769NI-DISC0	1	80	80
PCB fabricada	1	5	5
Fuente de alimentación	1	5	10
Cable USB-C	1	2	2
Cable microUSB	1	2	2
TOTAL			**389,05**

Tabla 1 Presupuesto del material

Las horas invertidas para la implementación de la aplicación en ambas plataformas ha sido un total de treinta y cuatro horas, a razón de treinta euros por hora de trabajo, el coste de la mano de obra del proyecto es de 1020 €. Sumando esta cantidad al coste del material el coste total del proyecto es de 1408,05 €.

Herramientas

En este capítulo serán presentadas las herramientas que he empleado para el diseño y la realización de cada una de las partes del estudio.

Todas las herramientas software utilizadas en este proyecto son herramientas de software libre el cual permite su uso de forma gratuita, he seleccionado este tipo de aplicaciones para demostrar que se puede ejecutar un gran conjunto de proyectos sin tener que invertir en licencias de software propietario.

Tal y como se ha presentado en el capítulo anterior los módulos han sido producidos por impresión 3D, lo que conlleva un diseño CAD previo que en este caso ha sido generado por el software de diseño 3D FreeCAD.

El archivo STL generado por FreeCAD es importado a Ultimaker Cura, un software de preparación de impresión 3D, lo que hace esta aplicación es mediante la geometría importada y los parámetros introducidos por el usuario de la impresora 3D genera el archivo gcode. Este archivo es leído por el firmware de la impresora para completar la impresión de la pieza esperada.

La impresora 3D utilizada en este proyecto es la ANET A8, una impresora 3D de bajo coste capaz de imprimir piezas en un tamaño de 220x220x240mm.

Ilustración 4 Anet A8

La estructura principal ha sido fabricada a partir de los tubos de PVC mencionados en el capitulo de material, los cuales han sido cortados con una sierra de calar a las dimensiones necesarias para poder completar la forma de la estructura deseada junto con los codos y las tes.

Las conexiones eléctricas del conjunto se han logrado mediante la utilización de la manguera de 4 hilos y los conectores JST-XH. Para ello los hilos se han soldado utilizando un soldador de estaño a los terminales correspondientes de cada componente.

A la hora de realizar las conexiones y pelar las mangueras se han utilizado estas herramientas físicas: tijeras de electricista, crimpadora, pistola térmica y pistola termofusible.

En el caso de la plataforma de Raspberry es necesario cargar la imagen del sistema operativo que el usuario decida en la tarjeta de memoria que se utilizará. Raspberry pi imager es un programa que nos facilita esta tarea y que también es software libre. Además, es necesario instalar el interprete de Python para poder ejecutar código de ese lenguaje, que descargaremos e instalaremos desde su página web oficial: https://www.python.org/downloads/ o bien desde la terminal de Linux.

El desarrollo de la aplicación en esta plataforma se ha llevado a cabo con el entorno de desarrollo integrado VSCode. Además de este IDE también nos hemos apoyado en QT Designer, un programa basado en gráficos que nos proporciona una interfaz muy visual para diseñar nuestras interfaces gráficas.

Para el desarrollo en la plataforma compuesta por el microcontrolador y el lenguaje C/C++ se ha tenido que diseñar una PCB que permite conectar los módulos a la placa de desarrollo. El software seleccionado para este propósito ha sido la suite KiCad EDA, una suite de diseño automatizado de electrónica open source. Una vez diseñada la placa se diseño mediante la empresa JLCPCB.

El desarrollo de firmware correspondiente a esta plataforma se ha realizado empleando los programas que facilita el fabricante del microcontrolador ST Microelectronics de forma gratuita. TouchGFX, un programa especializado con el que el desarrollador puede diseñar y personalizar las interfaces gráficas, y STMCubeIDE, un entorno de desarrollo integrado que nos proporciona muchas características interesantes a la hora de escribir nuestro código y que cuenta con un compilador para cada microcontrolador de la familia STM32.

Desarrollo

El desarrollo del proyecto consta de dos partes bien diferenciadas. Por un lado, nos encontramos con una parte común que será igual para ambas plataformas. En la que es necesario diseñar la estructura y los módulos, realizar la elección del material, establecer el diseño de las conexiones eléctricas y confeccionar lo previamente descrito para llevarlo a la realidad. Y por otro lado, la parte correspondiente a la implementación de cada una de las plataformas.

Común

El diseño de la estructura ha sido muy rudimentario, puesto que se trata de un prototipo se han probado diferentes configuraciones de la disposición de los módulos, hasta dar con el diseño final que se adapta perfectamente a las dimensiones de juego para una persona adulta estándar.

Las dimensiones finales son las que aparecen en la siguiente imagen en mm:

Ilustración 5 Dimensiones de la estructura

Para construirla tal y como se figura en la imagen, se han cortado los tubos de PVC a las medidas indicadas con una sierra de calar y se han unido las piezas generadas de este modo con los codos y tes que aparecen en ella.

Dando como resultado una estructura modular, puesto que los codos y las tes son desmontables. Pudiendo de esta manera transportar el juego de una manera muy rápida y sencilla, comprometiendo ligeramente la estabilidad.

La solución a los problemas de estabilidad se han obtenido a la hora de seleccionar el tipo de actuador que se ha montado en los módulos, ya que originalmente se planteó que estos fueran pulsadores. En el caso de haber mantenido este tipo de actuadores, la estructura recibiría impactos del jugador para desactivar el modulo activo en ese momento y haría que esta se tambalease pudiendo incluso caerse.

El actuador más adecuado a emplear teniendo en cuenta lo previamente mencionado ha sido un sensor de proximidad, que permite al jugador desactivar el módulo sin tener contacto físico con los módulos y la propia estructura.

A continuación se presentará el diseño de cada uno de los 8 módulos necesarios para interactuar con el jugador. Como ya se ha mencionado el tipo de sensor ha sido escogido dadas las limitaciones provenientes de la estructura, es por ello que se debe diseñar una geometría que sea capaz de albergar en su interior uno de eso sensores y que permitan mediante un orificio central en la tapa superior leer el entorno.

También es necesario que la tapa pueda acoger de una manera correcta el indicador LED de módulo activado, teniendo en cuenta todas estas restricciones y necesidades se han diseñado en CAD las siguientes piezas que conforman la parte mecánica del módulo:

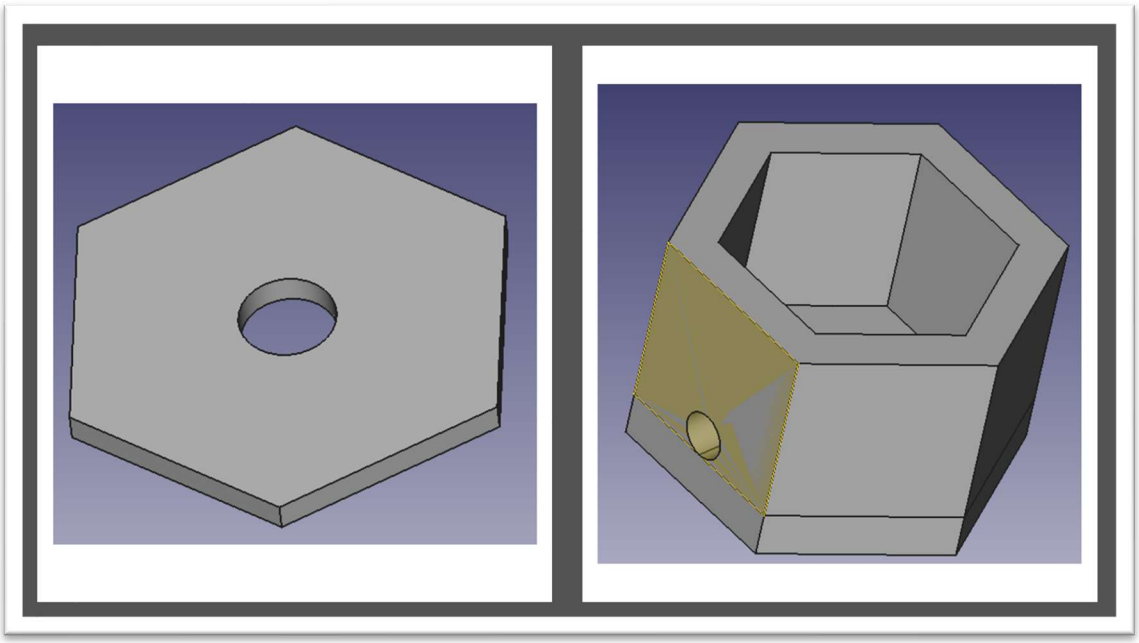

Ilustración 6 Diseño CAD del módulo

Sobre la tapa, la geometría que aparece en la parte izquierda de la ilustración, se colocarán las tiras de LED pegadas individualmente y conectadas entre sí con estaño.

Con el diseño de estas geometrías finalizado, se exportan en formato STL para poder abrir las mismas en Cura. En este software, se importan cada una de las anteriores piezas una por una y se parametriza la impresión 3D como más convenga tras lo cual se genera el archivo gcode capaz de ser interpretado por la impresora:

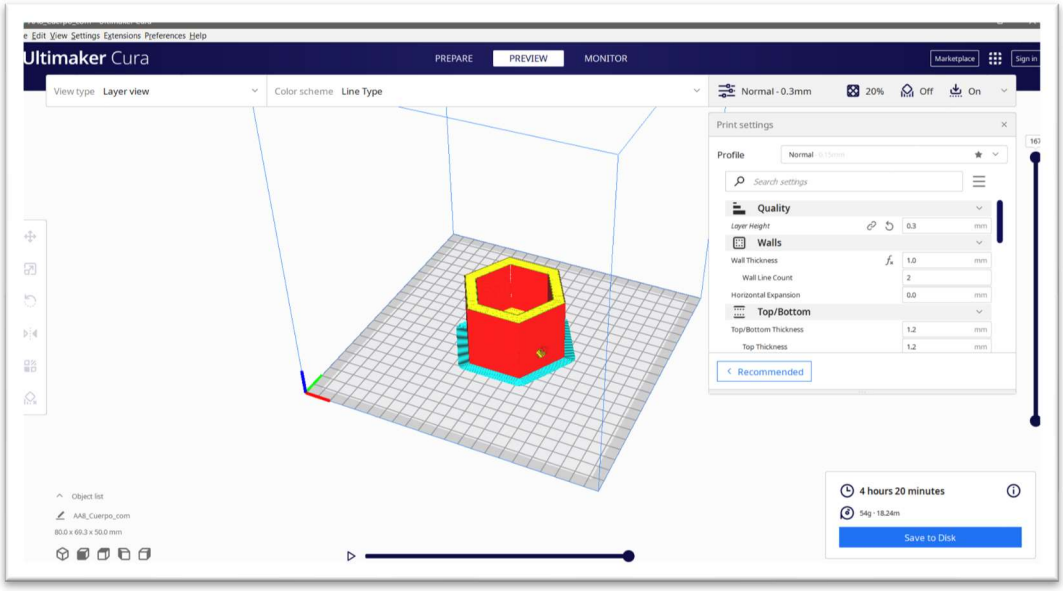

Ilustración 7 Apariencia de Ultimaker Cura

Se Copia ese archivo generado por Ultimaker Cura en una tarjeta microSD, la cual será introducida en la ranura de lectura que contiene la impresora 3D y se ejecuta la impresión de la pieza.

En ese momento la impresora realizará su trabajo y tras el tiempo requerido para su impresión se habrán obtenido las piezas diseñadas y que cumplen las especificaciones necesarias. Se repite este proceso con ambas geometrías ocho veces, puesto que en la aplicación se ha decidido utilizar esa cantidad de módulos.

Con los módulos impresos tras un tiempo de espera extendido y tras solucionar algún que otro problema causado por la impresora, se colocan los sensores de proximidad con una escuadra de carpintería pegado con pegamento termofusible en posición que sea capaz de medir los objetos cercanos a la tapa. Estos son los sensores escogidos debido a su bajo coste:

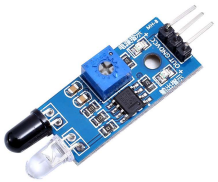

Ilustración 8 Sensor de proximidad por infrarrojos

Se trata de un sensor compuesto por un diodo emisor de luz infrarroja que cuando se coloca un cuerpo en su proximidad que hace rebotar ese haz, que es recogido por un fototransistor el cual cuando recibe ese haz de luz se satura. Cuando el transistor se satura cambia el potencial de la patilla no inversora del amplificador operacional a GND, de esta manera el amplificador se satura a negativo cuando detecta algún objeto en su proximidad obteniendo GND a la salida.

En el caso que no detecte un cuerpo y el haz no se haya dirigido hacia el fototransistor la patilla no inversora del amplificador se encontrará al potencial de alimentación mediante una resistencia de pull-up. En este caso se saturará a positivo obteniendo a la salida la tensión de alimentación.

El rango en el que mide el sensor se puede manipular mediante un potenciómetro que al variar su resistencia, variaremos la tensión que se aplica a la puerta inversora del amplificador operacional. Este funcionamiento se puede deducir a partir de la siguiente ilustración en la que se representa el circuito del sensor:

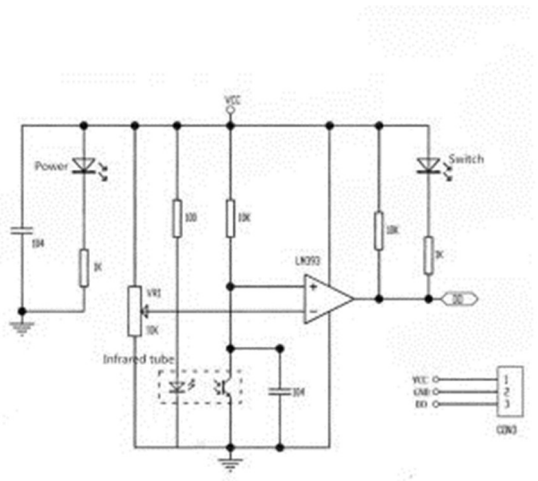

Ilustración 9 Diseño electrónico del sensor de proximidad

En la tapa superior del módulo se colocan las tiras con el pegamento que contienen las tiras LED formando un hexágono que se adapta perfectamente con la geometría de la tapa y se conectan en paralelo cada uno de los pares resistencia diodo LED que hemos adquirido en las tiras.

Una vez completados los pasos anteriores cortaremos 20 cm de manguera y se realiza la siguiente conexión con el sensor y los diodos LED:

HILOS	PIN
white	GND_com
brown	V+
yellow	S_sensor
green	Led +

Tabla 2 Pinout del módulo

Esta conexión se realiza por el lado del conector mediante soldadura con estaño, y por parte contraria se pela cada uno de los hilos y se conecta después del crimpado correspondiente a los pines del conector JST-XH de 4 pines macho como se observa a continuación:

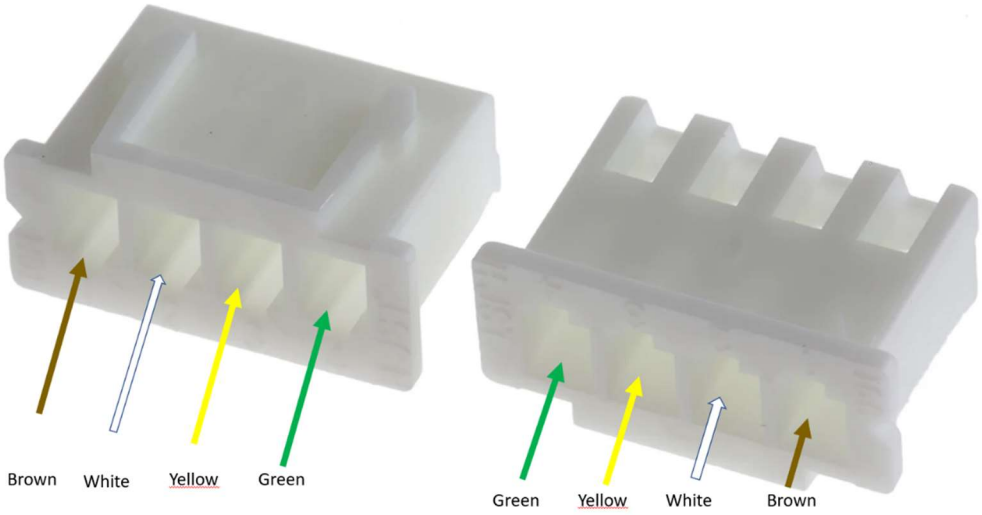

Ilustración 10 Pinout módulo

Completando este proceso el resultado es un módulo funcional que contiene todo lo necesario para interactuar con el jugador y que es conectable mediante un conector JST-XH de 4 pines hembra a cualquiera de las dos plataformas.

Es necesario que estos módulos puedan ser sostenidos por la estructura en las posiciones determinadas a las cuales se quiere hacer que el jugador tenga que desplazarse para desactivar el módulo activo. Esto se ha resuelto colocando una tira de velcro en las tuberías de la estructura en las posiciones que más adelante contemplaremos y la tira complementaria de velcro en la base de cada uno de ellos. De esta manera se pueden colocar los módulos en la estructura cuando se requiera para jugar y retirarlos de forma sencilla para su transporte.

Los módulos fabricados son de la siguiente forma:

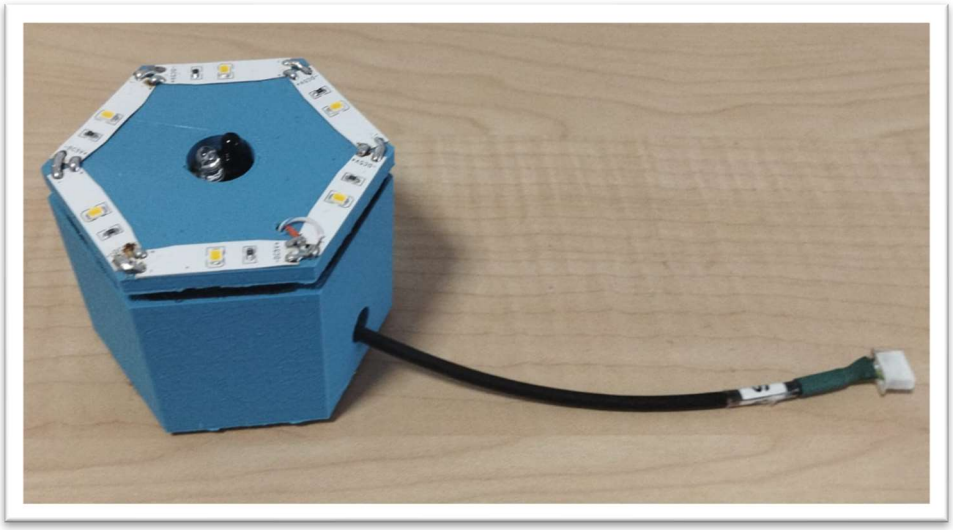

Ilustración 11 Módulo completo

La ubicación de esos módulos en la estructura se ha dispuesto de la siguiente manera en cuatro niveles diferentes: dos módulos en un primer nivel con una separación de 550mm con respecto al centro de la estructura, dos módulos en un nivel superior con una separación de 250mm con respecto al centro, dos módulos más en un nivel superior al segundo con la misma separación y un ultimo nivel que alberga otro dos módulos con una separación de 550mm con respecto al centro al igual que los módulos del primer nivel.

La estructura queda de la siguiente manera con los módulos montados, y estos presentan la siguiente numeración:

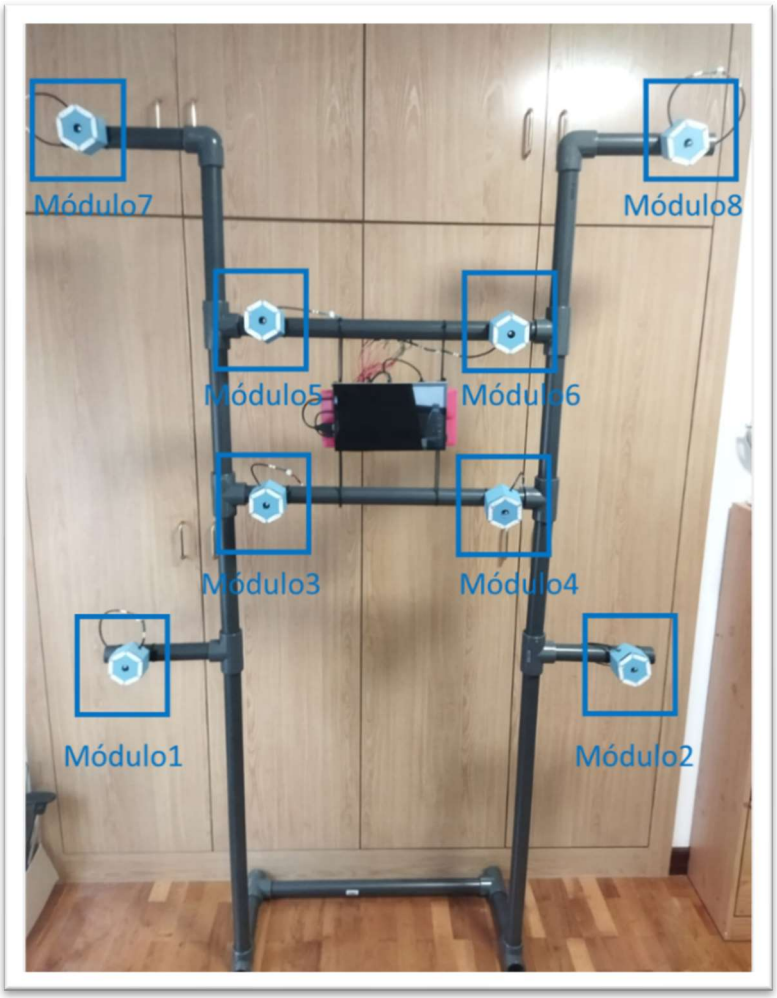

Ilustración 12 Disposición de los módulos numerados

Este es el punto de partida del desarrollo de la aplicación que no es común para sendas plataforma, por lo que en este momento se realiza un desarrollo paralelo en cada una de las plataformas.

Primeramente se completa el desarrollo en la plataforma de Raspberry pi y se implementa la aplicación por completo para posteriormente hacer lo propio con el microcontrolador. A continuación se explicará como ha sido el desarrollo para cada una de ellas.

Raspberry pi

La primera fase del desarrollo en cualquiera de las plataformas es realizar un esquema eléctrico, para poder posteriormente realizar las conexiones de los GPIO(General Purpose Input Output) de la placa a utilizar con los módulos. La Raspberry pi 400 cuenta con 40 pines de propósito general con el siguiente Pinout:

Ilustración 13 Raspberry pi 400 GPIO Pinout

Como se puede observar en la imagen existen algunos pines de alimentación desde donde podremos alimentar nuestros módulos, la alimentación se logra puenteando los GND y V+, es decir, los hilos blanco y marrón respectivamente con sus homónimos.

Esta alimentación se realizará desde los pines de 5V y Ground que se encuentran en los pines: 02,04,06,09,14,20,25,30,34 y 39.

Para conectar los sensores de proximidad y LEDs se basará en la siguiente tabla y se conectarán los hilos de las mangueras correspondientes a los pines del conector de GPIO de la Raspberry soldándolos con estaño. Esta es la tabla de conexión de la Raspberry con los módulos:

Sensor	GPIO	PIN
1	GPIO02	3
2	GPIO04	7
3	GPIO17	11
4	GPIO27	13
5	GPIO23	16
6	GPIO10	19
7	GPIO11	23
8	GPIO05	29

Led	GPIO	PIN
1	GPIIO03	5
2	GPIO14	8
3	GPIO15	10
4	GPIO22	15
5	GPIO24	18
6	GPIO09	21
7	GPIO08	24
8	GPIO06	31

Tabla 3 Conexionado módulos con GPIO Raspberry

Este es el esquema eléctrico correspondiente a la conexión de Raspberry con los módulos:

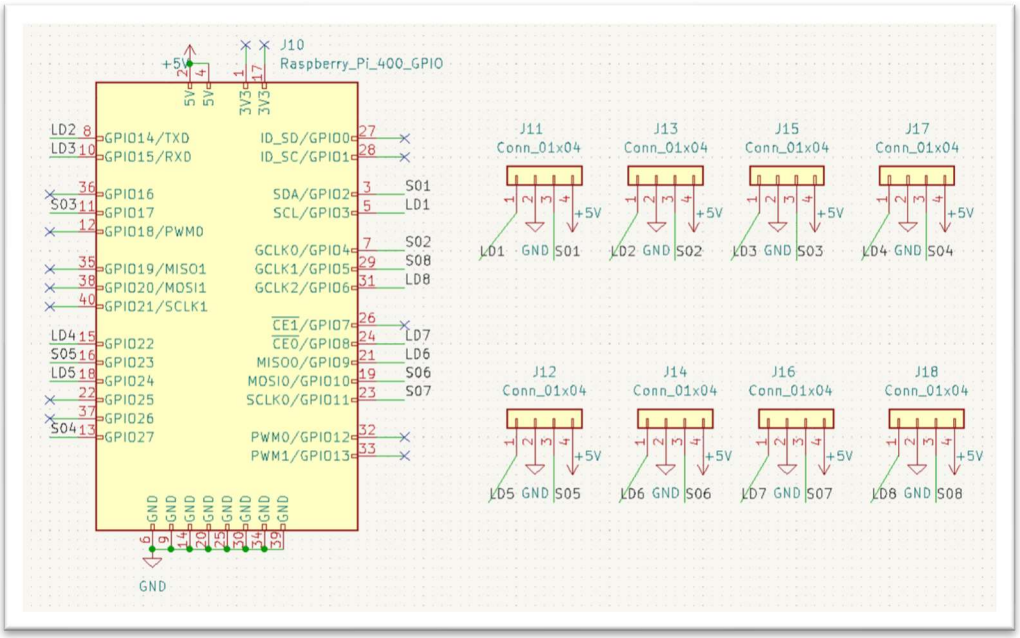

Ilustración 14 Esquema eléctrico conexionado Raspberry

La Raspberry pi 400 necesita un sistema operativo para funcionar por eso debemos crear una tarjeta microSD con la imagen del sistema operativo que hayamos escogido, en este caso Ubuntu, una distribución de Linux muy amigable para el usuario.

Esta tarea se lleva a cabo con el software Raspberry pi imager, esta herramienta nos permite seleccionar la imagen del sistema operativo que deseemos y lo escribirá en la microSD seleccionado. Generando así de una manera muy sencilla un disco de arranque que será utilizado por la CPU de la Raspberry pi para ofrecernos todos los servicios. Así es la interfaz de Raspberry pi imager:

Ilustración 15 Interfaz Raspberry pi imager

Tras este paso, se insertará la tarjeta en la ranura de la Raspberry y se alimentará. Para poder ejecutar la aplicación que será escrita en Python, puesto que este lenguaje cuenta con una variedad de librerías diseñadas para controlar de una forma muy sencilla los periféricos de esta

placa, es necesario instalar el interprete. Para instalar el interprete en el sistema operativo, se ejecuta el siguiente comando en la terminal de nuestro equipo: *sudo apt install python3.10*

En esta aplicación además del código escrito en Python, también se ha integrado la parte de la interfaz gráfica con Qt, Qt es un framework multiplataforma orientado a objetos ampliamente usado para desarrollar programas que utilicen interfaz gráfica de usuario, así como también diferentes tipos de herramientas para la línea de comandos y consolas para servidores que no necesitan una interfaz gráfica de usuario.

Es un framework que se integra perfectamente con las aplicaciones escritas en Python y además cuenta con un programada llamado Qt Designer que permite diseñar el layout de nuestra interfaz grafica de una forma práctica y visual.

Esta es la apariencia de Qt Designer:

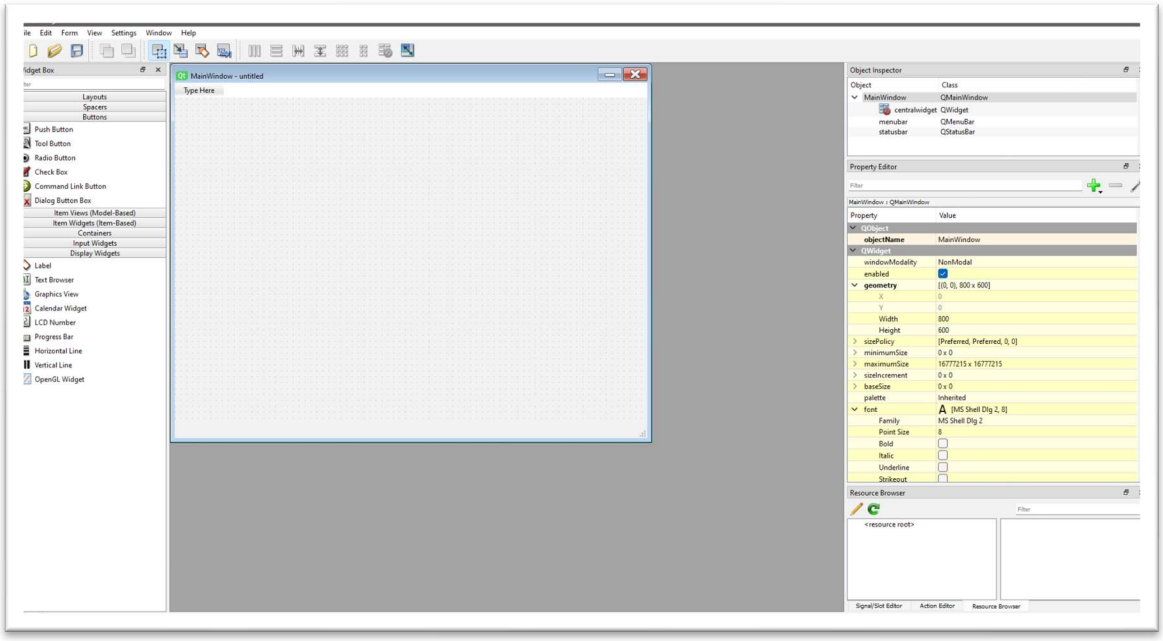

Ilustración 16 Apariencia de Qt Designer

Esta herramienta permite dar la apariencia requerida y añadir widgets a la pantalla de aplicación. Los widgets son los elementos básicos que nos ofrece Qt y hay una amplia variedad de ellos como pueden ser: botones, etiquetas, calendarios, etc. Existe una gran cantidad de ellos que proporcionan al desarrollador nuevas características para la aplicación.

En este proyecto la apariencia de la pantalla de la aplicación será la siguiente:

Ilustración 17 Apariencia interfaz de usuario Raspberry pi

Para poder utilizar la interfaz creada con Qt Designer en la aplicación de Python basta con importar el módulo recursos, que contiene las imágenes insertadas en la interfaz, y el binding PyQt, que nos permite cargar el archivo con extensión ui generado por Qt Designer.

Este tipo de arquitectura es muy utilizado para realizar aplicaciones de escritorio que se ejecutan en nuestro ordenador personal, permitiendo que el usuario disponga de una interfaz llamativa visualmente y fácil de diseñar. Como existe una gran comunidad de desarrolladores que utilizan esta plataforma, en internet podemos encontrar mucha documentación que puede ser de utilidad, además de foros como por ejemplo stack overflow donde los desarrolladores pueden colaborar para solventar las dudas que se generan durante el desarrollo de la aplicación.

El desarrollo de la parte de código y Qt Designer fue realizado directamente desde la propia Raspberry pi, ya que es posible ejecutar tanto Qt Designer como VS Code desde la propia placa. Esto facilita el desarrollo y permite al desarrollador no tener que apoyarse en un equipo distinto para realizar el trabajo, también permite depurar el programa de una forma más sencilla realizando los cambios que vea conveniente en cualquier parte.

La última tarea para concluir el desarrollo de la aplicación en esta plataforma es la de escribir el código y depurarlo hasta conseguir cumplir las especificaciones establecidas. Al final del documento se anexará el código fuente y como se puede observar consta de un único archivo de unas 100 líneas. Cuando el lenguaje de programación de la aplicación es Python, es una característica común que el código quede tan reducido teniendo en cuenta que la mayoría del código se encuentra en los módulos importados.

Al tratarse de una prueba de concepto en la cual lo interesante es poder generar un prototipo funcional sin la importancia de mantener una buena apariencia y tratando de minimizar los tiempos de desarrollo al máximo, es muy común el hecho de realizar las conexiones entre los componentes de la forma en la que se muestra en la siguiente imagen:

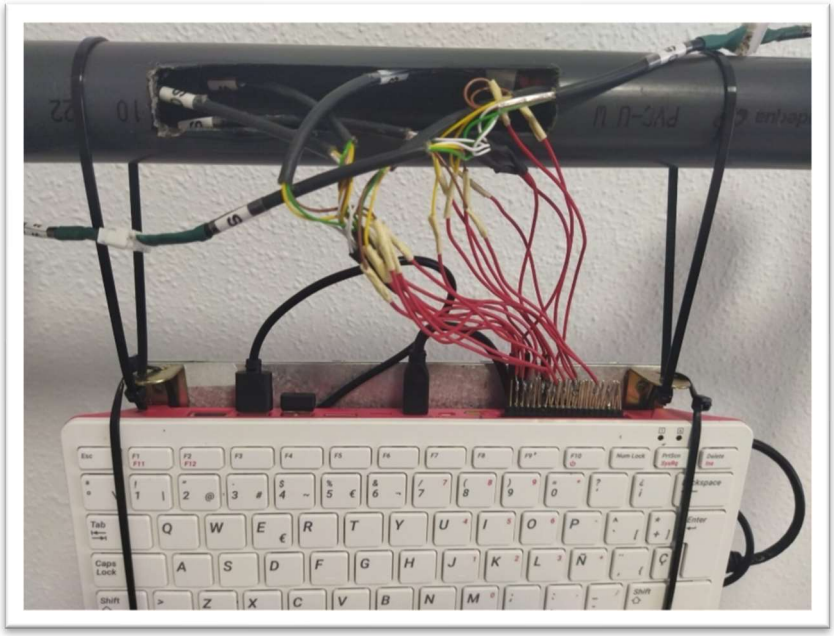

Ilustración 18 Conexionado físico GPIO Raspberry

Para una implementación real, estos detalles no son permisibles y sería necesario realizar un conexionado con conectores correctamente aislados evitando cualquier mal contacto y dando una apariencia visual más cuidada.

STM32

Tras concluir el desarrollo de la aplicación en la plataforma de Raspberry es más sencillo realizar un nuevo desarrollo en la plataforma de microcontrolador, debido a que se ha visualizado la aplicación en su conjunto y contamos con la experiencia de los problemas que han ido surgiendo durante el desarrollo anterior.

Pese a ello, se trata de una plataforma más compleja de controlar y que necesita profundizar más en su arquitectura para poder desarrollar las aplicaciones. Esto requiere tener a mano mucha documentación del fabricante para entender como funciona cada uno de sus periféricos y tener un conocimiento avanzado de programación en C/C++.

Al igual que para la anterior plataforma, se ha comenzado teniendo en cuenta como realizar las conexiones con los módulos y la alimentación tanto de la placa como de los módulos. Esta plataforma cuenta con unas limitaciones de potencia más severas que impiden alimentar los módulos desde la propia placa, debido a esta circunstancia se ha decidido diseñar una PCB capaz de conectarse a la placa por medio del conector compatible con Arduino del que dispone. Esta será capaz de alimentar tanto la placa como los módulos mediante microUSB y permitirá comunicar los sensores y LEDs con el microcontrolador a través de los pines de propósito general que se encuentran en el conector.

El pinout del conector de Arduino que se encuentra en el datasheet de la placa es el siguiente:

6.2 ARDUINO® Uno V3 connectors (CN11, CN14, CN13, and CN9)

Table 6. ARDUINO® connectors (CN11, CN14, CN13, and CN9)

Left connectors					-	Right connectors				
CN No.	Pin No.	Pin name	STM32 pin	Function	-	Function	STM32 pin	Pin name	Pin No.	CN No.
-					-	I2C1_SCL	PB8	D15	10	CN9 digital
						I2C1_SDA	PB9	D14	9	
						AVDD	-	AREF	8	
						Ground	-	GND	7	
CN11 power	1	NC	-	-		SPI2_SCK	PA12	D13	6	
	2	IOREF	-	3.3 V Ref		SPI2_MISO	PB14	D12	5	
	3	RESET	NRST	RESET		TIM12_CH2, SPI2_MOSI	PB15	D11	4	
	4	+3V3	-	3.3 V input/output		TIM1_CH4, SPI2_NSS	PA11	D10	3	
	5	+5 V	-	5 V output		TIM12_CH1	PH6	D9	2	
	6	GND	-	Ground		-	PJ4	D8	1	
	7	GND	-	Ground		-				
	8	VIN	-	Power input	-	-	PJ3	D7	8	CN13 digital
-						TIM11_CH1	PF7	D6	7	
CN14 analog	1	A0	PA6	ADC1_IN6		TIM3_CH3	PC8	D5	6	
	2	A1	PA4	ADC1_IN4		-	PJ0	D4	5	
	3	A2	PC2	ADC1_IN12		TIM10_CH1	PF6	D3	4	
	4	A3	PF10	ADC3_IN8		-	PJ1	D2	3	
	5	A4	PF8 or PB(1)	ADC3_IN6 (PF8) or I2C1_SDA (PB9)		USART6_TX	PC6	D1	2	
	6	A5	PF9 or PB8(1)	ADC3_IN7 (PF9) or I2C1_SCL (PB8)		USART6_RX	PC7	D0	1	

Ilustración 19 Pinout del Conector Arduino STM32

Teniendo en cuenta este pinout se ha diseñado la siguiente PCB con KiCad, con la que alimenta la placa de desarrollo a través del pin VIN y GND. Esta PCB permite a su vez la conexión con los módulos y su alimentación con los conectores JST-XH.

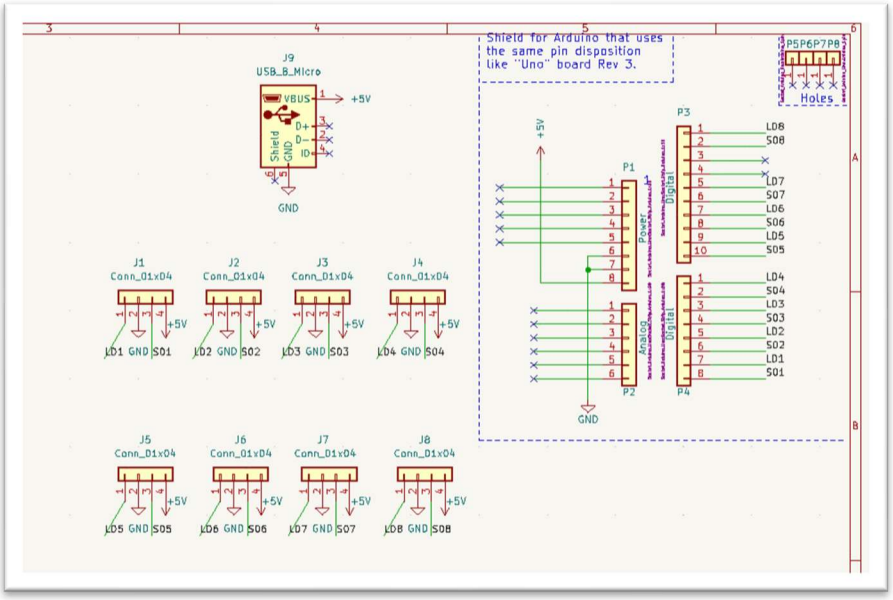

Ilustración 20 Diseño eléctrico STM32

KiCad permite la opción de usar como referencia un template con las dimensiones del conector de Arduino, lo que facilita en gran medida el diseño. Partiendo de ese template se ha añadido el símbolo y los footprint de los ocho conectores JST-XH de cuatro pines hembra y un conector microUSB, una vez dispongo de los componentes en el esquemático se procede conectando los pines como corresponde y se puede observar en esta tabla:

Sensor	GPIO	PIN
1	D0	PA6
2	D2	PC2
3	D4	PJ0
4	D6	PF7
5	D8	PJ4
6	D10	PA11
7	D12	PB14
8	D14	PB9

Led	GPIO	PIN
1	D1	PA4
2	D3	PF6
3	D5	PC8
4	D7	PJ3
5	D9	PH6
6	D11	PB15
7	D13	PA12
8	D15	PB8

Tabla 4 Conexionado módulos con STM32

Con las conexiones establecidas y verificando que ningún pin este mal conectado o sin conectar, se diseña el layout de la PCB. En esta fase se define la ubicación de los componentes en la PCB.

Con los componentes dispuestos en su lugar, se trazan las pistas por donde mejor convenga para realizar la conexión física entre los pines que fueron conectados en el esquemático. A veces es necesario utilizar vías para pasar de una capa a otra la pista, esta situación aparece cuando no hay espacio para trazar una pista por el hecho de que ya hay otras pistas ocupando ese lugar.

Cuando todos los pines se encuentren conectados como se ha estipulado en el esquemático, será posible generar los archivos necesarios por la empresa de fabricación de PCBs para poder fabricar la placa. Estos archivos, llamados Gerber, son generados por el propio KiCad y tras ser subidos a la página web del fabricante donde se realizará el pedido se realizará la petición de fabricación. Tras unos días de espera se recibe la placa con los pads preparados para colocar los componentes necesarios.

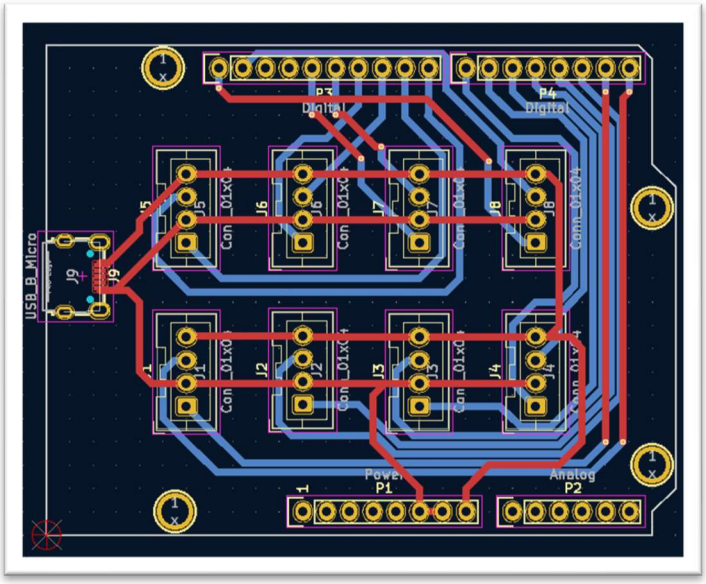

Ilustración 21 Disposición de los componentes en PCB

Los componentes son colocados manualmente sobre los pads donde deben ser soldados con estaño. En este caso la mayoría de componentes utilizados son THD, de esta manera basta con un soldador común y una bobina de estaño para ensamblar la placa. El resultado es el siguiente:

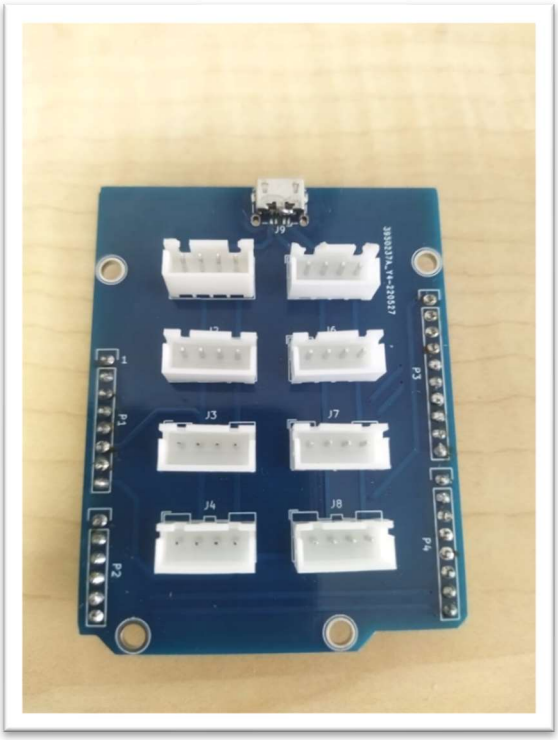

Ilustración 22 PCB diseñada para la conexión de módulos con STM32

Este es el último proceso en cuanto a desarrollo de hardware respecta, a continuación se presentará como se ha implementado el firmware de la aplicación con las herramientas que ofrece el fabricante del microcontrolador.

Inicialmente se diseña la parte gráfica al igual que en la anterior plataforma, para lo cual es utilizado TouchGFX, un framework para desarrollo de gráficos optimizado para los microcontroladores STM32.

Esta herramienta es muy similar a Qt Designer pero en vez de ser multiplataforma, únicamente tiene como objetivo dar soporte al desarrollador para generar aplicaciones con lenguaje C++ en microcontroladores.

A la hora de crear un nuevo proyecto se puede partir de una plantilla que cubre todas las características de la placa de desarrollo utilizada en el proyecto, ayudando de esta manera con el trabajo de configuración de los periféricos al desarrollador.

Al igual que con Qt Designer, se diseñará la interfaz añadiendo los widgets que necesarios y dando una apariencia visualmente agradable. Este es el resultado:

Ilustración 23 Apariencia interfaz de usuario STM32

Al finalizar el diseño de la interfaz, TouchGFX genera el código como un proyecto C/C++, este proyecto puede ser modificado para añadir las características que el desarrollador cree oportunas. Pudiendo de esta manera utilizar distintos periféricos y controlarlos a su gusto.

Para poder modificar ese proyecto se puede utilizar la herramienta STM32CubeIDE, una plataforma de desarrollo C/C++ con configuración de periféricos, generación de código, compilación de código y características de depurado para el ecosistema STM32. Esta herramienta es capaz de abrir el proyecto generado por TouchGFX y permite al desarrollador realizar los cambios que mejor se adapten a su aplicación.

En este IDE, se inicializarán los periféricos necesarios para el proyecto, se escribe el código para que el microcontrolador se comporte de la manera que se ha estipulado en las especificaciones del proyecto, se compilará el proyecto para generar los archivos binarios válidos para que sea ejecutado y se depurará el código en tiempo de uso para cambiar el código cuando no se comporta como debería.

Para mayor facilidad a la hora de escribir el código se utilizará el HAL (Hardware Abstraction Layer) que proporciona el fabricante, se trata de una capa de abstracción de hardware que permite al desarrollador controlar el hardware del microcontrolador de una manera más sencilla utilizando las funciones que esta biblioteca proporciona. Para el desarrollo de aplicaciones con HAL es necesario disponer de la documentación que indica cual es la utilización correcta de tales funciones.

Este es el árbol de archivos que contiene el proyecto:

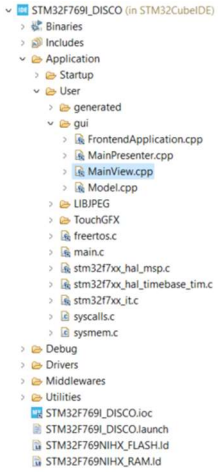

Ilustración 24 Árbol de archivos proyecto C/C++

El desarrollador debe incluir el código escrito por él en la carpeta de User/gui en el archivo correspondiente a la pantalla en la que se encuentra el widget que activa esa ejecución de código. El código empleado en este proyecto se puede visualizar en el apartado de anexos del documento, ese código ha sido incluido en el archivo “MainView.cpp”.

Algo que sucede habitualmente cuando se desarrolla un prototipo es que las placas diseñadas en primera instancia no son totalmente válidas, esto sucede porque no se tienen en cuenta todos los factores en el momento del diseño. Durante el desarrollo del proyecto primero se diseño y se fabricó la PCB y posteriormente se programó el firmware, pero a la hora de generar el código con TouchGFX dos de los pines que se esperaban destinar a la interacción con los módulos fueron utilizados para la comunicación con la RAM externa que cuenta la placa. Es por ello que fue necesario modificar manualmente la PCB que se había fabricado, dando como resultado el circuito de la siguiente imagen:

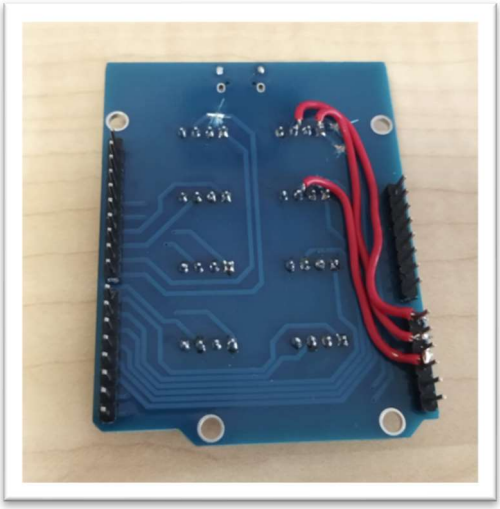

Ilustración 25 Modificación de PCB

Esta nueva conexión se tendría en cuenta para volver a diseñar una nueva PCB con los cambios establecidos una vez se haya comprobado que todo funciona correctamente y se haya depurado el código.

Resultados

En este capítulo se expresará de la forma más objetiva posible los resultados obtenidos correspondientes a las variables más importantes a la hora de desarrollar la aplicación para ambos casos. De esta manera, se quiere transmitir al lector del documento una idea global de las diferencias y similitudes a la hora de trabajar con estas plataformas.

Las variables sobre las que se hará hincapié son de nuevo: el tiempo de desarrollo, coste del desarrollo, y las capacidades y limitaciones de cada plataforma.

Tiempo de desarrollo

El tiempo de desarrollo a tener en cuenta será exclusivamente el invertido en el desarrollo de esta aplicación, no se tendrá en cuenta el tiempo destinado a formación u obtención de información acerca de las plataformas. Pero es importante recalcar que la curva de aprendizaje en el caso de los microcontroladores junto con C/C++ es más lenta y requiere más tiempo de formación. Ya que con la plataforma Raspberry y Python es posible realizar aplicaciones sencillas sin requerir muchos conocimientos de electrónica y programación.

De esta forma nos centraremos en los tiempos destinados al diseño y programación de la aplicación que se ha desarrollado en este caso. Comenzando por la primera plataforma se puede decir que el tiempo dedicado para la realización del proyecto no es muy elevada, puesto que se ha destinado alrededor de dos horas en la configuración del sistema operativo, otras dos horas diseñando la interfaz gráfica en Qt Designer y seis horas más escribiendo y depurando el código de Python. Dando como resultado un total de diez horas para el desarrollo completo.

En el caso de la plataforma STM32 el tiempo invertido ha sido superior en este caso, el hecho de tener que diseñar una PCB para la conexión de los módulos ha hecho que el proyecto requiera más tiempo de ejecución. Además el tiempo dedicado a la programación y depuración del código ha sido notablemente superior al de la otra plataforma. Para el diseño de la PCB se han destinado seis horas, otras dos horas en el diseño de la interfaz gráfica y dieciséis horas más a la programación y depuración del código. Dando como resultado un total de veinticuatro horas.

Como se puede observar el tiempo dedicado al desarrollo en la plataforma de microcontrolador es más de dos veces superior al empleado en la plataforma de Raspberry, se puede añadir que presumiblemente cuanto más complicada sea la aplicación y más características contenga este valor aumentará.

En esta gráfica se puede apreciar de forma visual esta diferencia:

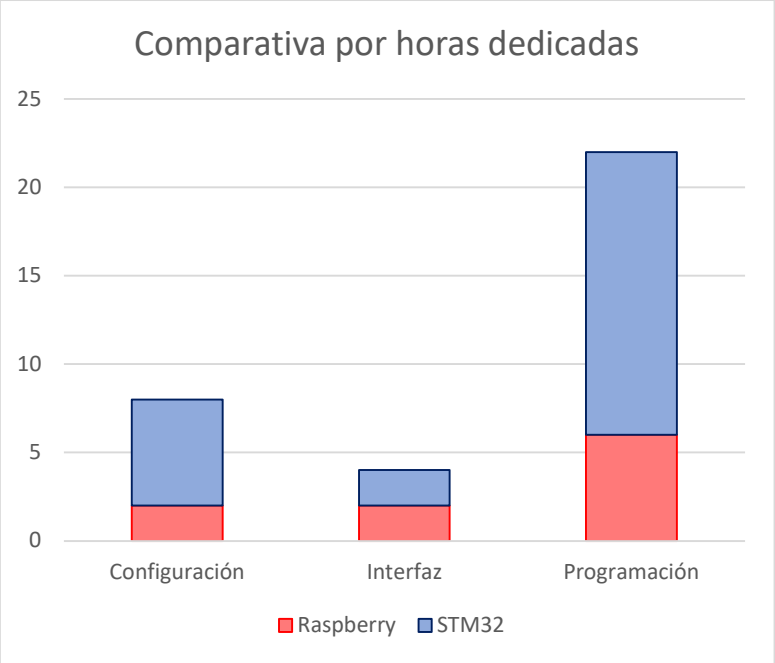

Ilustración 26 Gráfica comparativa de tiempo

Coste del desarrollo

En el apartado anterior Material y presupuesto de este documento se ha presentado un coste global de proyecto que en este punto dividiremos entre cada plataforma para ver cual es el gasto asignado a cada una de ellas.

Como ha sido costumbre durante todo el documento se comenzará por indicar los gastos relacionados con la implementación de la aplicación en Raspberry. Para esta tarea no se tendrán en cuenta los gastos de material común y se mostrarán únicamente los gastos referidos al material necesario por cada plataforma y el tiempo dedicado a su desarrollo para incluir las horas del técnico.

En las siguientes tablas se puede observar de forma desglosada los coste de desarrollo para cada plataforma:

Material	Unidades	Precio(unidad) €	Precio €
Raspberry pi 400	1	80	80
HMTECH Monitor 1024 x 600 HDMI	1	100	100
Tarjeta de memoria microSDXC	1	12	12
Cable USB-C	1	2	2
Mano de obra(horas)	10	30	300
TOTAL			**494**

Tabla 5 Costes desglosados del desarrollo en Raspberry

Material	Unidades	Precio(unidad) €	Precio €
STM32F769NI	1	10	10
PCB fabricada	1	5	5
Cable microUSB	1	2	2
Mano de obra(horas)	24	30	720
TOTAL			**737**

Tabla 6 Costes desglosados del desarrollo en STM32

El hecho de que el coste del desarrollo en STM32 sea tan superior es debido a las horas invertidas por el técnico. Este aspecto es determinante a la hora de decidir que tipo de plataforma es la óptima para implementar la solución y se discutirá en el capítulo próximo del documento.

Capacidades y limitaciones

Debido a que en este proyecto las dos opciones cuentan con las capacidades necesarias para poder ejecutarlo, en este apartado quiero presentar a grandes rasgos cuales son las capacidades y limitaciones de cada una de las plataformas en el momento de la creación de aplicaciones en ellas.

La Raspberry pi teniendo en cuenta que se trata de un SBC que puede ejecutar sistemas operativos de propósito general como Linux, es una alternativa muy interesante cuando lo que se requiere es una capacidad de cómputo considerable y es necesario mover grandes cantidades de memoria, como por ejemplo sucede con los gráficos.

Las capacidades destacables son las siguientes:

- Capacidad de cómputo elevada.
- Facilidad de desarrollo de aplicaciones (tiempo + coste de prototipado).
- Amplia comunidad.
- Integración de capacidad de computo + interacción física(GPIO)
- Posibilidad de ejecución de servicios integrados (BBDD, páginas web, etc.)

Las limitaciones que nos encontramos en este dispositivo son:

- La ausencia de conversores analógico-digitales, lo que impide la lectura de una gran cantidad de sensores sin componentes adicionales.
- El control sobre el hardware, puesto que en todo momento este equipo se encuentra ejecutando un sistema operativo de propósito general los tiempos de ejecución de la aplicación no se pueden considerar real time, algo que puede ser indispensable en muchas aplicaciones de sistemas empotrados.
- Mala integración en dispositivos de venta masiva, a la hora de implementar una solución empotrada es necesario diseñar completamente la electrónica en una PCB donde se incluirá la parte de control, en este caso esta placa se comercializa como un producto final y no como un componente.

En el caso de STM32 nos encontramos con un microcontrolador de 32 bits que se trata de un componente muy económico capaz de integrarse en cualquier dispositivo y que debe ser programado para que cumpla con un propósito especifico.

Por ello las cualidades positivas son las siguientes:

- Precio muy económico.
- Integración en cualquier dispositivo.
- Consumo energético reducido.
- Interacción física(GPIO + ADC/DAC)

Las limitaciones más destacables son las siguientes:

- Capacidad de cómputo limitada.
- Dificultad en el desarrollo (tiempo + coste de desarrollo).

Conclusión

En el presente trabajo se ha llevado acabo la implementación de una misma aplicación en dos plataformas de sistemas empotrados diferentes, tras la cual se ha realizado una comparativa del proceso de desarrollo entre ambas plataformas.

En la siguiente tabla se pueden apreciar las características en las que destaca cada plataforma:

Variable	Raspberry	STM32
Tiempo de desarrollo	✓	✗
Coste del material	✗	✓
Capacidad de cómputo	✓	✗
Capacidad de Real-time	✗	✓
Interacción con HW	✗	✓

Tabla 7 Características destacables de cada plataforma

En la comparativa se ha podido determinar que las aplicaciones se pueden poner en funcionamiento de una forma más rápida con la Raspberry, convirtiéndose así en la mejor opción a la hora de generar un prototipo funcional.

Este prototipo podrá ser presentado para comprender mejor la idea inicial del proyecto y realizar los cambios necesarios que satisfagan al cliente. De esta forma los costes de mano de obra invertidos en el diseño del producto se verán reducidos en una cantidad notable. También será destacable la reducción del tiempo de diseño del producto, ya que el proceso es más veloz y los cambios que se tengan que efectuar en él no requerirán tanto detalle para actuar sobre los periféricos como en el caso del microcontrolador.

La mayor ventaja de esta plataforma es la capacidad de computación que ofrece y que puede ser aprovechada para aplicaciones de Edge computing, un concepto que está adquiriendo mucha importancia en la actualidad y consiste en el procesamiento de los datos en el lugar donde se producen. Esta plataforma presenta las características necesarias para satisfacer este proceso y creo que es la plataforma que mejor se adapta para este cometido, su consumo no es tan elevado como un computador de arquitectura x86 y la capacidad no dista demasiado además de ofrecer los pines de propósito general capaces de controlar por si misma muchos actuadores y sensores.

Por su parte, la plataforma de microcontrolador se adapta mejor a los casos en los que las aplicaciones no sean tan demandantes en términos de capacidad de computación, para la mayoría de aplicaciones de sistemas empotrados que podemos encontrar en la actualidad son más que suficientes. Su mayor atractivo es además la capacidad de control del tiempo de ejecución, que nos permite desarrollar aplicaciones en las cuales este factor juegue un papel importante y sea indispensable la robustez, como puede ser por ejemplo la activación del airbag de un coche.

Estos dispositivos se encuentran encapsulados en un único chip integrado por lo que es fácilmente integrar toda la solución en una única PCB, incluyendo la parte de alimentación, el control y hasta a veces los actuadores y sensores en un mismo dispositivo. Esta característica hace de la plataforma la solución que integran los fabricantes de productos a gran escala como pueden ser por ejemplo los automovilísticos o de electrodomésticos. Este uso se debe al coste que se obtiene desarrollando una solución propietaria para la aplicación a gran escala, se trata

de la plataforma más económica en términos de material y teniendo en cuenta que una vez se ha desarrollado se puede replicar tantas veces como se requiera sin tener que invertir de nuevo en mano de obra de desarrollo.

Cabe mencionar que estas plataformas son complementarias y pueden ser utilizadas en el mismo proyecto de distintas maneras posibles:

- En muchas aplicaciones puede ser oportuno utilizar la Raspberry para producir un prototipo de una manera rápida y sencilla, actualizando las especificaciones de la misma hasta quedar satisfecho con el producto, para posteriormente implementar el producto final con el microcontrolador ahorrando costes de material y amplificando el beneficio económico además de los consumos energéticos.
- Otra opción es realizar un desarrollo conjunto, obteniendo el beneficio de las fortalezas de cada una de las plataformas. De este modo, la interacción con el hardware será llevada a cabo por el microcontrolador que proporciona un control minucioso del mismo pudiendo implementar aplicaciones en tiempo real. Siendo este el caso, la Raspberry se ocupa de la parte que requiere tanto la parte gráfica como la capacidad de cómputo con los datos obtenidos desde el microcontrolador.

Para poder integrar ambas plataformas en la misma solución es posible utilizar un protocolo de comunicación estandarizado que sea compatible con las dos, como por ejemplo: SPI, I2C, CAN, etc.

De esta forma se podrá compartir la información precisa para un correcto funcionamiento de la aplicación y se establecerá una unión de las distintas partes del proyecto.

Trabajos futuros

En este trabajo se ha realizado una comparativa entre únicamente dos de las plataformas disponibles para el desarrollo de proyectos de sistemas empotrados, pero existen muchas alternativas más en el mercado.

De aquí en adelante me gustaría seguir desarrollando aplicaciones empotradas y seguir formándome en este sector. Este estudio se podrá complementar con nuevas aplicaciones y con la implementación en otras plataformas y lenguajes como pueden ser: FPGAs, otros SBC y nuevos dispositivos que aparezcan en el mercado.

Bibliografía

[1] https://support.touchgfx.com/4.19/docs/introduction/welcome (TouchGFX)

[2] https://support.ultimaker.com/hc/en-us/categories/360002327600 (Ultimaker Cura)

[3] https://doc.qt.io/qt-6/qtdesigner-manual.html (Qt Designer)

[4] https://docs.python.org/3/ (Python)

[5] https://docs.microsoft.com/es-es/cpp/?view=msvc-170 (C/C++)

[6] https://jlcpcb.com/ (JLCPCB)

[7]https://www.st.com/resource/en/user_manual/um1905-description-of-stm32f7-hal-and-lowlayer-drivers-stmicroelectronics.pdf (STM32F7 HAL)

[8] https://www.st.com/resource/en/user_manual/dm00276557-discovery-kit-with-stm32f769ni-mcu-stmicroelectronics.pdf (STM32F769NI-DISC0 datasheet)

[9] Francisco Javier Ceballos (2019). Curso de programación C/C++ (C/C++)

[10] Dr Peter Dalmaris (2018). KiCad like a pro (KiCad)

Anexos

Anexo 1: Código de aplicación Python

```
import sys
import recursos
from PyQt5 import uic
from PyQt5.QtWidgets import QMainWindow, QApplication
from random import randint
from datetime import datetime
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

Lista_pin_in = [2,4,17,27,23,10,11,5]
Lista_pin_in_kids = [2,4,17,27]
Lista_pin_out = [3,14,15,22,24,9,8,6]
Lista_pin_out_kids = [3,14,15,22]
numero_leds = len(Lista_pin_in)
numero_leds_kids = len(Lista_pin_in_kids)

for i in Lista_pin_in:
    GPIO.setup(i,GPIO.IN)

for i in Lista_pin_out:
    GPIO.setup(i,GPIO.OUT)

def inicio():
    for i in Lista_pin_out:
        GPIO.output(i,0)
        time.sleep(0.2)
        GPIO.output(i,1)
        time.sleep(0.2)

    for i in Lista_pin_out:
         GPIO.output(i,0)


class MainApp(QMainWindow):

    def __init__(self) -> None:
        super().__init__()
        uic.loadUi('First.ui', self)

        self.setWindowFlags(Qt.FramelessWindowHint)

        self.btn_encender.clicked.connect(self.encender)
        self.btn_apagar.clicked.connect(self.apagar)
        self.btn_jugar.clicked.connect(self.jugar)
        self.btn_jugar_kids.clicked.connect(self.jugar_kids)


    def encender(self):
        for i in Lista_pin_out:
            GPIO.output(i, 1)

    def apagar(self):
        for i in Lista_pin_out:
            GPIO.output(i,0)

    def jugar(self):
        inicio = datetime.now()
        anterior = -1
        for j in range(25):
            nuevo = randint(0,numero_leds-1)

            while anterior == nuevo:
                nuevo = randint(0,numero_leds-1)

            anterior = nuevo
            GPIO.output(Lista_pin_out[nuevo],0)
            while GPIO.input(Lista_pin_in[nuevo]) == 1:
                GPIO.output(Lista_pin_out[nuevo],1)

```

```
73                 GPIO.output(Lista_pin_out[nuevo],0)
74
75             final = datetime.now()
76             tiempo = final - inicio
77             self.lbl_count.setText(str(tiempo.total_seconds()))
78
79         def jugar_kids(self):
80             inicio = datetime.now()
81             anterior = -1
82             for j in range(25):
83                 nuevo = randint(0,numero_leds_kids-1)
84
85                 while anterior == nuevo:
86                     nuevo = randint(0,numero_leds_kids-1)
87
88                 anterior = nuevo
89                 GPIO.output(Lista_pin_out_kids[nuevo],0)
90                 while GPIO.input(Lista_pin_in_kids[nuevo]) == 1:
91                     GPIO.output(Lista_pin_out_kids[nuevo],1)
92
93                 GPIO.output(Lista_pin_out_kids[nuevo],0)
94
95             final = datetime.now()
96             tiempo = final - inicio
97             self.lbl_count.setText(str(tiempo.total_seconds()))
98
99     if __name__ == '__main__':
100        app = QApplication([])
101        window = MainApp()
102        window.show()
103        app.exec_()
```

Anexo 2: Código de la aplicación C/C++

```
#include <gui/main_screen/MainView.hpp>
#include <touchgfx/utils.hpp>
#include "main.h"
#include "stm32f7xx_hal.h"

uint32_t get_aleatorio;
volatile int aleatorio_8;
volatile int viejo_aleatorio_8=0;
extern RNG_HandleTypeDef hrng;

MainView::MainView()
{


}

void MainView::setupScreen()
{
    MainViewBase::setupScreen();
}

void MainView::tearDownScreen()
{
    MainViewBase::tearDownScreen();
}

void MainView::jugar()
{
    for(int i=0;i<20;i++)
    {
        HAL_RNG_GenerateRandomNumber(&hrng, &get_aleatorio);
        aleatorio_8 = hrng.RandomNumber%8 +1;
        while(viejo_aleatorio_8 == aleatorio_8)
        {
            HAL_RNG_GenerateRandomNumber(&hrng, &get_aleatorio);
            aleatorio_8 = hrng.RandomNumber%8 +1;
        }
        viejo_aleatorio_8 = aleatorio_8;
            switch (aleatorio_8)
            {
                case 1:
                    HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_RESET);
                    HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, GPIO_PIN_RESET);
                    HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, GPIO_PIN_RESET);
                    HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, GPIO_PIN_RESET);
                    HAL_GPIO_WritePin(LED5_GPIO_Port, LED5_Pin, GPIO_PIN_RESET);
                    HAL_GPIO_WritePin(LED6_GPIO_Port, LED6_Pin, GPIO_PIN_RESET);
                    HAL_GPIO_WritePin(LED7_GPIO_Port, LED7_Pin, GPIO_PIN_RESET);
                    HAL_GPIO_WritePin(LED8_GPIO_Port, LED8_Pin, GPIO_PIN_RESET);
                    HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_SET);
                    while(HAL_GPIO_ReadPin(S01_GPIO_Port,S01_Pin)==GPIO_PIN_SET)
                    {
                        if(HAL_GPIO_ReadPin(S01_GPIO_Port,S01_Pin)==GPIO_PIN_RESET)
                            {
                                break;
                            }
                    }
                    break;

                case 2:
                    HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_RESET);
                    HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, GPIO_PIN_RESET);
                    HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, GPIO_PIN_RESET);
                    HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, GPIO_PIN_RESET);
                    HAL_GPIO_WritePin(LED5_GPIO_Port, LED5_Pin, GPIO_PIN_RESET);
                    HAL_GPIO_WritePin(LED6_GPIO_Port, LED6_Pin, GPIO_PIN_RESET);
                    HAL_GPIO_WritePin(LED7_GPIO_Port, LED7_Pin, GPIO_PIN_RESET);
                    HAL_GPIO_WritePin(LED8_GPIO_Port, LED8_Pin, GPIO_PIN_RESET);
                    HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, GPIO_PIN_SET);
                    while(HAL_GPIO_ReadPin(S02_GPIO_Port,S02_Pin)==GPIO_PIN_SET)
                    {
                        if(HAL_GPIO_ReadPin(S02_GPIO_Port,S02_Pin)==GPIO_PIN_RESET)
                            {
```

```
                                    break;
                                }
                            }
                            break;

                        case 3:
                            HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED5_GPIO_Port, LED5_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED6_GPIO_Port, LED6_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED7_GPIO_Port, LED7_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED8_GPIO_Port, LED8_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, GPIO_PIN_SET);
                            while(HAL_GPIO_ReadPin(S03_GPIO_Port,S03_Pin)==GPIO_PIN_SET)
                            {
                                if(HAL_GPIO_ReadPin(S03_GPIO_Port,S03_Pin)==GPIO_PIN_RESET)
                                    {
                                        break;
                                    }
                            }
                            break;

                        case 4:
                            HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED5_GPIO_Port, LED5_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED6_GPIO_Port, LED6_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED7_GPIO_Port, LED7_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED8_GPIO_Port, LED8_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, GPIO_PIN_SET);
                            while(HAL_GPIO_ReadPin(S04_GPIO_Port,S04_Pin)==GPIO_PIN_SET)
                            {
                                if(HAL_GPIO_ReadPin(S04_GPIO_Port,S04_Pin)==GPIO_PIN_RESET)
                                    {
                                        break;
                                    }
                            }
                            break;

                        case 5:
                            HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED5_GPIO_Port, LED5_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED6_GPIO_Port, LED6_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED7_GPIO_Port, LED7_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED8_GPIO_Port, LED8_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED5_GPIO_Port, LED5_Pin, GPIO_PIN_SET);
                            while(HAL_GPIO_ReadPin(S05_GPIO_Port,S05_Pin)==GPIO_PIN_SET)
                            {
                                if(HAL_GPIO_ReadPin(S05_GPIO_Port,S05_Pin)==GPIO_PIN_RESET)
                                    {
                                        break;
                                    }
                            }
                            break;

                        case 6:
                            HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED5_GPIO_Port, LED5_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED6_GPIO_Port, LED6_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED7_GPIO_Port, LED7_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED8_GPIO_Port, LED8_Pin, GPIO_PIN_RESET);
                            HAL_GPIO_WritePin(LED6_GPIO_Port, LED6_Pin, GPIO_PIN_SET);
                            while(HAL_GPIO_ReadPin(S06_GPIO_Port,S06_Pin)==GPIO_PIN_SET)
```

```
147                         {
148                             if(HAL_GPIO_ReadPin(S06_GPIO_Port,S06_Pin)==GPIO_PIN_RESET)
149                                 {
150                                     break;
151                                 }
152                         }
153                         break;
154 
155                     case 7:
156                         HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_RESET);
157                         HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, GPIO_PIN_RESET);
158                         HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, GPIO_PIN_RESET);
159                         HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, GPIO_PIN_RESET);
160                         HAL_GPIO_WritePin(LED5_GPIO_Port, LED5_Pin, GPIO_PIN_RESET);
161                         HAL_GPIO_WritePin(LED6_GPIO_Port, LED6_Pin, GPIO_PIN_RESET);
162                         HAL_GPIO_WritePin(LED7_GPIO_Port, LED7_Pin, GPIO_PIN_RESET);
163                         HAL_GPIO_WritePin(LED8_GPIO_Port, LED8_Pin, GPIO_PIN_RESET);
164                         HAL_GPIO_WritePin(LED7_GPIO_Port, LED7_Pin, GPIO_PIN_SET);
165                         while(HAL_GPIO_ReadPin(S07_GPIO_Port,S07_Pin)==GPIO_PIN_SET)
166                         {
167                             if(HAL_GPIO_ReadPin(S07_GPIO_Port,S07_Pin)==GPIO_PIN_RESET)
168                                 {
169                                     break;
170                                 }
171                         }
172                         break;
173 
174                     case 8:
175                         HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_RESET);
176                         HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, GPIO_PIN_RESET);
177                         HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, GPIO_PIN_RESET);
178                         HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, GPIO_PIN_RESET);
179                         HAL_GPIO_WritePin(LED5_GPIO_Port, LED5_Pin, GPIO_PIN_RESET);
180                         HAL_GPIO_WritePin(LED6_GPIO_Port, LED6_Pin, GPIO_PIN_RESET);
181                         HAL_GPIO_WritePin(LED7_GPIO_Port, LED7_Pin, GPIO_PIN_RESET);
182                         HAL_GPIO_WritePin(LED8_GPIO_Port, LED8_Pin, GPIO_PIN_RESET);
183                         HAL_GPIO_WritePin(LED8_GPIO_Port, LED8_Pin, GPIO_PIN_SET);
184                         while(HAL_GPIO_ReadPin(S08_GPIO_Port,S08_Pin)==GPIO_PIN_SET)
185                         {
186                             if(HAL_GPIO_ReadPin(S08_GPIO_Port,S08_Pin)==GPIO_PIN_RESET)
187                                 {
188                                     break;
189                                 }
190                         }
191                         break;
192 
193                 }
194         }
195         HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_RESET);
196         HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, GPIO_PIN_RESET);
197         HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, GPIO_PIN_RESET);
198         HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, GPIO_PIN_RESET);
199         HAL_GPIO_WritePin(LED5_GPIO_Port, LED5_Pin, GPIO_PIN_RESET);
200         HAL_GPIO_WritePin(LED6_GPIO_Port, LED6_Pin, GPIO_PIN_RESET);
201         HAL_GPIO_WritePin(LED7_GPIO_Port, LED7_Pin, GPIO_PIN_RESET);
202         HAL_GPIO_WritePin(LED8_GPIO_Port, LED8_Pin, GPIO_PIN_RESET);
203     }
```